

Entwicklerhandbuch

AWS SDK for .NET



AWS SDK for .NET: Entwicklerhandbuch

Copyright © 2024 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 Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist die AWS SDK for .NET	1
Über diese Version	1
Wartung und Support für SDK-Hauptversionen	2
Häufige Anwendungsfälle	2
Weitere Themen in diesem Abschnitt	2
VerwandteAWSWerkzeuge	3
-Tools für Windows PowerShell und Tools for PowerShell Core	3
Toolkit for VS Code	3
-Toolkit for Visual Studio	3
Toolkit für Azure DevOps	4
SDKs- und Tools-Referenz	4
Weitere Ressourcen	4
Erste Schritte	7
Installieren und konfigurieren Sie Ihre Toolchain	8
Plattformübergreifende Entwicklung	8
Windows mit Visual Studio und .NET Core	8
Nächster Schritt	9
Konfigurieren Sie die SDK-Authentifizierung	9
Aktivieren und Konfigurieren von IAM Identity Center	9
Konfigurieren Sie das SDK für die Verwendung von IAM Identity Center.	10
Starten einer AWS-Zugriffsportalsitzung	11
Zusätzliche Informationen	12
Machen Sie einen kurzen Rundgang	12
Einfache plattformübergreifende App	13
Einfache Windows-basierte App	18
Nächste Schritte	25
Starte ein neues Projekt	25
AWSRegion konfigurieren	26
Erstellen Sie einen Service-Client mit einer bestimmten Region	27
Geben Sie eine Region für alle Service-Clients an	28
Auflösung der Region	29
Besondere Informationen über die Region China (Peking)	29
Besondere Informationen zu neuen Diensten AWS	30
AWSSDK Pakete installieren mit NuGet	30

Verwendung NuGet über die Befehlszeile oder das Terminal	31
Verwenden NuGet im Visual Studio Solution Explorer	31
NuGet Von der Package Manager Console aus verwenden	32
Installieren Sie AWSSDK-Baugruppen ohne NuGet	33
Auflösung von Anmeldeinformationen und Profilen	34
Auflösung des Profils	35
Verwenden von Anmeldeinformationen für Verbundbenutzerkonten	36
Angaben von Rollen oder temporären Anmeldeinformationen	37
Verwendung von Proxy-Anmeldeinformationen	37
Benutzer und Rollen	38
Benutzer und Berechtigungssätze	38
Servicerollen	38
Erweiterte Konfiguration	39
AWSSDK.extensions.netCore.Setup und IConfiguration	40
Konfigurieren von anderen Anwendungsparametern	45
Referenz der Konfigurationsdateien für AWS SDK for .NET	53
Verwenden von Legacy-Anmeldeinformationen	65
Wichtige Warnhinweise und Richtlinien für Anmeldeinformationen	66
Die Datei mit den gemeinsam genutzten AWS Anmeldeinformationen verwenden	67
Den SDK Store verwenden (nur Windows)	71
SDK-Funktionen	76
Asynchrone APIs	76
Wiederholungen und Timeouts	78
Wiederholversuche	78
Timeouts	80
Beispiel	81
Paginatoren	82
Wo finde ich Paginatoren?	82
Was geben mir Paginatoren?	82
Synchrone und asynchrone Paginierung	83
Beispiel	83
Zusätzliche Überlegungen für Paginatoren	87
Zusätzliche Tools	88
AWS Bereitstellen des Tools	88
AWS Framework zur Nachrichtenverarbeitung für .NET	88
Advanced Auth	89

Single Sign-On	89
Voraussetzungen	90
Ein SSO-Profil einrichten	90
SSO-Token generieren und verwenden	92
Weitere Ressourcen	97
Tutorials	97
Tutorial: Ausschließlich.NET-Anwendung	98
Tutorial: AWS CLI und .NET-Anwendung	107
Bereitstellen in AWS	116
Bereitstellen über die .NET-CLI	116
Bereitstellen aus den IDE-Toolkits	116
Anwendungsfälle	117
ASP.NET Core-Apps	117
.NET-Konsolen-Apps	118
Blazor- WebAssembly Apps	119
AWS Lambda-Projekte	120
Voraussetzungen	120
Verfügbare Lambda-Befehle	121
Schritte zur Bereitstellung	121
Migrieren Sie Ihr Projekt	123
Was ist neu?	123
Unterstützte Plattformen	124
.NET Core	125
.NET Standard 2.0	125
.NET Framework 4.5	125
.NET Framework 3.5	125
Mobile Class Library und Xamarin	126
Unity-Unterstützung	126
Weitere Informationen	126
Migration auf Version 3	126
Über die Versionen des AWS SDK for .NET	126
Neugestaltung der Architektur für das SDK	127
Abwärtskompatible Änderungen	127
Migration auf Version 3.5	129
Was hat sich für Version 3.5 geändert	129
Migrieren von synchronem Code	131

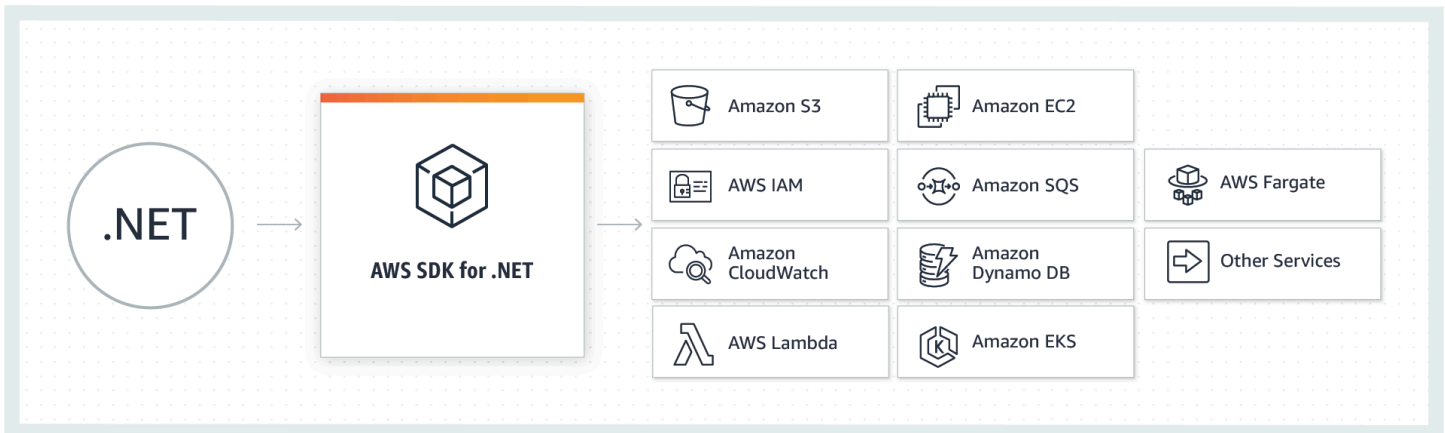
Migration auf Version 3.7	132
Migration von .NET-Standard 1.3	132
Arbeiten mit - AWS Services	134
Codebeispiele mit Anleitungen	134
AWS CloudFormation	135
Amazon Cognito	139
DynamoDB	148
Amazon EC2	178
IAM	242
Amazon S3	261
Amazon SNS	272
Amazon SQS	276
AWS Lambda	310
APIs	310
Voraussetzungen	310
Themen	310
Lambda-Anmerkungen	310
Allgemeine Bibliotheken und Frameworks	312
Framework zur Nachrichtenverarbeitung	312
AWS OpsWorks	324
APIs	324
Voraussetzungen	325
Sonstige Services und Konfigurationen	325
Codebeispiele	326
Aktionen und Szenarien	326
ACM	328
Aurora	332
Auto Scaling	374
Amazon Bedrock	456
Amazon-Bedrock-Laufzeit	460
AWS CloudFormation	480
CloudWatch	483
CloudWatch Protokolle	539
Amazon Cognito Identity Provider	554
Amazon Comprehend	579
DynamoDB	591

Amazon EC2	683
Amazon ECS	783
Elastic Load Balancing	796
EventBridge	850
AWS Glue	890
IAM	922
Amazon Keyspaces	1050
Kinesis	1078
AWS KMS	1096
Lambda	1108
MediaConvert	1146
Organisationen	1159
Amazon Polly	1178
Amazon RDS	1190
Amazon Rekognition	1225
Route-53-Domainregistrierung	1255
Amazon S3	1282
S3 Glacier	1411
SageMaker	1422
Secrets Manager	1455
Amazon SES	1458
Amazon SNS	1471
Amazon SQS	1518
Step Functions	1563
AWS STS	1591
AWS Support	1594
Amazon Transcribe	1621
Amazon Translate	1634
Serviceübergreifende Beispiele	1645
Erstellen einer Amazon-SNS-Anwendung	1646
Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos	1646
Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten	1647
Erstellen Sie einen Tracker für Aurora-Serverless-Arbeitsaufgaben	1647
Erstellen einer Anwendung zum Analysieren von Kundenfeedback	1648
Erkennen von Objekten in Bildern	1649
Sicherheit	1650

Datenschutz	1651
Identitäts- und Zugriffsverwaltung	1652
Zielgruppe	1652
Authentifizierung mit Identitäten	1653
Verwalten des Zugriffs mit Richtlinien	1657
Funktionsweise AWS-Services von mit IAM	1659
Fehlerbehebung für AWS Identität und Zugriff	1660
Compliance-Validierung	1662
Ausfallsicherheit	1663
Sicherheit der Infrastruktur	1664
Erzwingen einer Mindest-TLS-Version	1664
.NET Core	1665
.NET Framework	1666
AWS Tools for PowerShell	1666
Xamarin	1667
Unity	1668
Browser (für Blazor WebAssembly)	1668
S3-Verschlüsselungs-Client-Migration	1669
Migrationsübersicht	1669
Aktualisieren vorhandener Clients auf V1-transitional-Clients zum Lesen neuer Formate ...	1670
Migrieren von V1-transitional-Clients zu V2-Clients zum Schreiben neuer Formate	1670
Aktualisieren von V2-Clients auf V1-Formate, die nicht länger gelesen werden	1674
Besondere Überlegungen	1675
Abrufen von AWSSDK Baugruppen	1675
Zip-Dateien herunterladen und extrahieren	1675
Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung	1676
Beispiele für den Unterricht CredentialProfileStoreChain	1677
Beispiele für Klassen SharedCredentialsFile und AWSCredentialsFactory	1679
Unity-Unterstützung	1679
Xamarin Unterstützung	1681
API-Referenz	1682
Dokumentverlauf	1683
.....	mdclxxxviii

Was ist die AWS SDK for .NET

Die AWS SDK for .NET erleichtert die Erstellung von .NET-Anwendungen, die auf kostengünstige, skalierbare und zuverlässige AWS Services wie Amazon Simple Storage Service (Amazon S3) und Amazon Elastic Compute Cloud (Amazon EC2) zurückgreifen. Das SDK vereinfacht die Verwendung von AWS -Services, indem es eine Reihe von Bibliotheken bereitstellt, die für .NET-Entwickler konsistent und vertraut sind.



(OK, hat es erhalten! Ich bin bereit, [einzurichten](#) und [eine kurze Einführung zu unternehmen](#).)

Über diese Version

Note

Diese Dokumentation bezieht sich auf Version 3.0 und höher von AWS SDK for .NET. Es konzentriert sich hauptsächlich auf .NET Core und ASP.NET Core, enthält aber auch Informationen über .NET Framework und ASP.NET 4.x. Zusätzlich zu Windows und Visual Studio berücksichtigt es die plattformübergreifende Entwicklung gleich.

Weitere Informationen zur Migration finden Sie unter [Migrieren Sie Ihr Projekt](#).

Informationen zum Auffinden veralteter Inhalte für frühere Versionen von AWS SDK for .NET finden Sie im folgenden Element(e):

- [AWS SDK for .NET \(Version 2, veraltet\) Entwicklerhandbuch](#)

Wartung und Support für SDK-Hauptversionen

Informationen zu Wartung und Support für SDK-Hauptversionen und deren zugrunde liegende Abhängigkeiten finden Sie im [AWS -Referenzhandbuch zu SDKs und Tools](#):

- [AWS Wartungsrichtlinie für SDKs und Tools](#)
- [AWS Support-Matrix für SDKs und Tools](#)

Häufige Anwendungsfälle

Die AWS SDK for .NET hilft Ihnen dabei, mehrere hervorragende Anwendungsfälle zu finden, 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.
- Verwalten Sie [Amazon Simple Notification Service \(Amazon SNS\)](#) HTTP-Abonnements für Themen.
- Verwenden Sie das [S3-Übertragungsprogramm](#), 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 [Amazon EC2](#)-Instances und konfigurieren und fordern Sie Amazon EC2-[Spot-Instances an](#).

Weitere Themen in diesem Abschnitt

- [AWS-Tools im Zusammenhang mit AWS SDK for .NET](#)
- [AWS SDKs- und Tools-Referenzhandbuch](#)
- [Weitere Ressourcen](#)

AWS-Tools im Zusammenhang mit AWS SDK for .NET

-Tools für Windows PowerShell und Tools for PowerShell Core

AWS Tools for Windows PowerShell und AWS Tools for PowerShell Core sind PowerShell-Module, die auf der Funktionalität basieren, die vom AWS SDK for .NET bereitgestellt werden. Die AWS-Mithilfe der PowerShell-Tools können Sie Skriptoperationen auf Ihren AWS-Ressourcen der PowerShell-Eingabeaufforderung. Obwohl die Cmdlets unter Verwendung der Service-Clients und -Methoden aus dem SDK implementiert wurden, machen die Cmdlets eine idiomatische PowerShell zum Angeben der Parameter und zum Handhaben der Ergebnisse verfügbar.

Lesen Sie zum Einstieg [AWS Tools for Windows PowerShell](#).

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 diese anschließend auf einem AWS CloudFormation-Stack bereitstellen.
- Arbeiten Sie mit Amazon EventBridge EventBridge-Schemata.
- Verwenden Sie IntelliSense, wenn Sie mit Amazon ECS -Aufgabendefinitionsdateien arbeiten.
- Visualisieren einer AWS Cloud Development Kit (AWS CDK)-Anwendung

-Toolkit for Visual Studio

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 Services wie Lambda und Bereitstellungsassistenten für Webanwendungen und serverlose Anwendungen. Sie können die AWS-Mithilfe von Explorer können Sie Amazon EC2 EC2-Instances verwalten, mit Amazon DynamoDB DynamoDB-Tabellen arbeiten, Nachrichten in Amazon Simple Notification Service (Amazon SNS) -Warteschlangen veröffentlichen und vieles mehr. Dies alles ist in Visual Studio möglich.

Informationen zu den ersten Schritten finden Sie unter [Einrichten von AWS Toolkit for Visual Studio](#) aus.

Toolkit für Azure DevOps

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

So sehen die ersten Schritte mit AWS Toolkit for Azure DevOps, finden Sie unter [AWS Toolkit for Microsoft Azure DevOps-Benutzerhandbuch](#) aus.

AWSSDKs- und Tools-Referenzhandbuch

Die [AWSSDKs- und Tools-Referenzhandbuch](#) enthält Informationen, die für viele der AWSSDKs und Toolkits und das AWS CLI aus. Im Folgenden sehen Sie einige Beispiele für Informationen, die die Referenz enthält:

- Informationen zu dem [geteilt AWS config und credentials Dateien](#) und ihre [Standort](#) aus.
- [Einrichten von AWS Konten, Benutzer und Rollen](#)
- [Referenz für Konfigurations- und Authentifizierungseinstellungen](#)
- [AWS Gemeinsame Runtime \(CRT\) -Bibliotheken](#)
- [AWS-SDKs- und Tools-Wartung-Richtlinie](#)
- [Support-Matrix für AWS-SDKs- und Tools-Version](#)

Weitere Ressourcen

Unterstützte Dienste

Das AWS SDK for .NET unterstützt die meisten AWS-Infrastrukturprodukte und es werden laufend weitere Services hinzugefügt. Eine Liste der vom SDK unterstützten AWS-Services finden Sie in der [SDK LIESMICH-Datei](#).

Versionsverlauf

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

- [SDK-Änderungsprotokoll](#)
- [Was ist neu im AWS SDK for .NET](#)
- [Dokumentverlauf](#)

Homepage für AWS SDK for .NET

Für weitere Informationen über AWS SDK for .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 for .NET-API-Referenz](#).

AWSRe:post (früher AWSForen)

Besuch [AWSRe:Post](#), speziell die [Thema für die AWS SDK for .NET](#), um Fragen zu stellen oder Feedback zu geben AWS. Jede Dokumentationsseite hat eine [Versuche AWSRe:Post](#) Link am Ende der Seite, der Sie zum zugehörigen re:POST-Thema führt. 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 den zu folgen AWS SDK for .NET, gehe zum [Alle ThemenSeite](#), finde „.NET“ auf AWS, und wählen Sie [Folgenknopf](#).

Toolkits

- AWS Toolkit for Visual Studio: Wenn Sie die Microsoft Visual Studio IDE verwenden, sollten Sie sich die [AWS Toolkit for Visual Studio Benutzerleitfaden](#).
- AWS Toolkit for Visual Studio Code: Wenn Sie die Microsoft Visual Studio IDE verwenden, sollten Sie sich die [AWS Toolkit for Visual Studio Code Benutzerleitfaden](#).

Hilfreiche Bibliotheken, Erweiterungen und Tools

Besuchen Sie die [aws/dotnet](#) und [aws/aws-sdk-net](#) Repositorien auf dem GitHub Website mit 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](#)
- [AWSErweiterungen.NET Core-Setup](#)
- [AWS.NET protokollieren](#)
- [Amazon Cognito Authentication Extension Library](#)
- [AWS X-Ray SDK for .NET](#)

Andere Ressourcen

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

- [Entwicklernetzwerk](#)
- [.NET-Entwicklungsumgebung auf derAWS Cloud — Schnellstart-Referenzbereitstellung](#)
- [Hallo, Cloud! Blog](#)
- AWSWeißes Papier:[Entwicklung und Bereitstellung von.NET-Anwendungen aufAWS](#)
- [AWS Microservice Extractor for .NET](#)
- [Portierungsassistent für .NET](#)
- [AWSReferenzhandbuch zu SDKs und Tools](#)

Erste Schritte mit der AWS SDK for .NET

Um das verwenden zu können AWS SDK for .NET, müssen Sie Ihre Toolchain installieren und eine Reihe wichtiger Dinge konfigurieren, 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
- 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

- [Installieren und konfigurieren Sie Ihre Toolchain](#)
- [Konfigurieren Sie die SDK-Authentifizierung mit AWS](#)
- [Machen Sie einen kurzen Rundgang durch die AWS SDK for .NET](#)
- [Starte ein neues Projekt](#)
- [AWSRegion konfigurieren](#)
- [AWSSDK Pakete installieren mit NuGet](#)
- [Installieren Sie AWSSDK-Baugruppen ohne NuGet](#)
- [Auflösung von Anmeldeinformationen und Profilen](#)
- [Zusätzliche Informationen über Benutzer und Rollen](#)
- [Erweiterte Konfiguration für Ihre AWS SDK for .NET Projekt](#)
- [Verwenden von Legacy-Anmeldeinformationen](#)

Installieren und konfigurieren Sie Ihre Toolchain

Um die verwenden zu können AWS SDK for .NET, müssen Sie bestimmte Entwicklungstools installiert haben.

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, falls eines für den von Ihnen ausgewählten Editor und Ihr Betriebssystem verfügbar ist.

Zu den Beispielen gehören [AWS Toolkit for Visual Studio Code](#), [AWS 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

[Konfigurieren Sie die SDK-Authentifizierung mit AWS](#)

Konfigurieren Sie die SDK-Authentifizierung mit AWS

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

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

In diesem Thema wird davon ausgegangen, dass ein neuer Benutzer Projekte lokal entwickelt, von seinem Arbeitgeber noch keine Authentifizierungsmethode erhalten hat und AWS IAM Identity Center zum Abrufen temporärer Anmeldeinformationen verwendet. 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 einer AWS-Zugriffsportalsitzung](#)

Aktivieren und Konfigurieren von IAM Identity Center

Um IAM Identity Center verwenden zu können, muss es zunächst 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 für AWSSDKs und Tools. 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 Referenzhandbuch für AWSSDKs und Tools. Nachdem Sie diese Konfiguration abgeschlossen haben, sollte Ihr System die folgenden Elemente enthalten:

- Die AWS CLI, mit der Sie eine AWS-Zugriffssportalsitzung starten, bevor Sie Ihre Anwendung ausführen.
- Die gemeinsam genutzte AWS config Datei, die ein [\[default\]Profil](#) mit einer Reihe von Konfigurationswerten enthält, auf die im SDK verwiesen werden kann. Den Speicherort dieser Datei finden Sie unter [Speicherort der freigegebenen Dateien](#) im Referenzhandbuch für AWS SDKs und Tools. Der AWS SDK for .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-Zugriffssportalsitzung.

```
[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.SSO0IDC`

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

Starten einer AWS-Zugriffssitzung

Bevor Sie eine Anwendung ausführen, die zugreift AWS-Services, benötigen Sie eine aktive AWS Access-Portalsitzung, 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. Führen Sie den folgenden Befehl in der AWS CLI aus, um sich beim AWS-Zugriffportal anzumelden.

```
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.

ℹ Note

Wenn Sie bereits über eine aktive AWS-Zugriffssitzung verfügen und `aws sso login` ausführen, müssen Sie keine Anmeldeinformationen angeben.

Während des Anmeldevorgangs werden Sie möglicherweise aufgefordert, der AWS CLI Zugriff auf Ihre Daten zu gewähren. Da die AWS CLI auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des Namens `botocore` enthalten.

Zusätzliche Informationen

- Weitere Informationen zur Verwendung von IAM Identity Center und SSO in einer Entwicklungsumgebung finden Sie [Single Sign-On](#) im [Advanced Auth](#) Abschnitt. Diese Informationen umfassen alternative und fortgeschrittenere Methoden sowie Tutorials, die Ihnen zeigen, wie Sie diese Methoden verwenden.
- Weitere Authentifizierungsoptionen für das SDK, z. B. die Verwendung von Profilen und Umgebungsvariablen, finden Sie im Kapitel [Konfiguration](#) im Referenzhandbuch für AWS SDKs und 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 Sicherheitsanmeldeinformationen](#) im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen Anbietern von Anmeldeinformationen finden Sie unter [Standardisierte Anbieter von Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Machen Sie einen kurzen Rundgang durch die AWS SDK for .NET

Dieser Abschnitt enthält grundlegende Tutorials für Entwickler, die noch nicht mit dem vertraut sind AWS SDK for .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 [Arbeiten mit - AWS Services](#). Weitere Codebeispiele finden Sie unter [AWS SDK for .NET Codebeispiele](#).

Themen

- [Einfache plattformübergreifende Anwendung mit dem AWS SDK for .NET](#)
- [Einfache Windows-basierte Anwendung mithilfe des AWS SDK for .NET](#)
- [Nächste Schritte](#)

Einfache plattformübergreifende Anwendung mit dem AWS SDK for .NET

In diesem Tutorial werden und.NET Core für die AWS SDK for .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 unter [Installieren und konfigurieren Sie Ihre Toolchain](#).

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.SSO0IDC
```

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.SSO0IDC
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\n dotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
                "\n\n Usage: 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 ersetzen Sie dabei **BUCKET-NAME** durch den Namen des ausgewählten Buckets.

```
dotnet run BUCKET-NAME
```

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 mithilfe des AWS SDK for .NET

In diesem Tutorial wird die AWS SDK for .NET unter Windows mit Visual Studio und .NET Core verwendet. Das Tutorial zeigt Ihnen, wie Sie das SDK verwenden, um die [Amazon 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 zum Konfigurieren Ihrer Entwicklungsumgebung finden Sie unter [Installieren und konfigurieren Sie Ihre Toolchain](#).

Erforderlich für die Entwicklung unter Windows mit Visual Studio und .NET Core:

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1, 3.1 oder höher

Dies ist in der Regel standardmäßig bei der Installation einer aktuellen Version von Visual Studio enthalten.

Note

Bevor Sie diese Tutorials verwenden, 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 Console-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. Verwenden Sie in diesem Fall unbedingt .NET Framework 4.6.2 oder höher.

2. Wenn das neu erstellte Projekt geladen ist, wählen Sie Tools , NuGet Package Manager , NuGet Pakete für Lösung verwalten aus.
3. Suchen Sie nach den folgenden NuGet Paketen und installieren Sie sie im Projekt: AWSSDK.S3, AWSSDK.SecurityTokenAWSSDK.SSO, und AWSSDK.SSO0IDC

Dieser Prozess installiert die NuGet Pakete aus dem [NuGet Paketmanager](#) . Da wir genau wissen, welche NuGet Pakete wir für dieses Tutorial benötigen, können wir diesen Schritt jetzt ausführen. Es ist auch üblich, dass die erforderlichen Pakete während der Entwicklung bekannt werden. Wenn dies geschieht, folgen Sie einem ähnlichen Prozess, um sie zum jeweiligen Zeitpunkt zu installieren.

4. Wenn Sie die Anwendung über die Eingabeaufforderung ausführen möchten, öffnen Sie jetzt eine Eingabeaufforderung und navigieren Sie zu dem Ordner, der die Build-Ausgabe enthalten wird. Dies ist in der Regel ähnlich wie 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.SSO0IDC
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\nndotnet_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, erhalten Sie möglicherweise einen Build-Fehler ähnlich dem folgenden:

„Feature 'asynchronc main' ist in C# 7.0 nicht verfügbar. Bitte verwenden Sie die Sprachversion 7.1 oder höher.“

Wenn dieser Fehler angezeigt wird, richten Sie Ihr Projekt so ein, dass es eine neuere Version der Sprache verwendet. Dies geschieht 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 Eingabeaufforderung (falls Sie zuvor eine geöffnet haben) oder über die IDE.
2. Untersuchen Sie die Ausgabe, um die Anzahl der Amazon S3-Buckets, die Sie besitzen, sofern vorhanden, und deren Namen zu sehen.
3. Wählen Sie einen Namen für einen neuen Amazon S3-Bucket aus. Verwenden Sie „dotnet-quicktour-s3-1-winvS-“ als Basis und fügen Sie etwas Eindeutiges hinzu, z. B. eine GUID oder Ihren Namen. Befolgen Sie unbedingt die Regeln für Bucket-Namen, wie unter [Regeln für die Bucket-Benennung](#) im [Amazon S3-Benutzerhandbuch](#) beschrieben.
4. Führen Sie die Anwendung erneut aus und geben Sie diesmal den Bucket-Namen an.

Ersetzen Sie in der Befehlszeile **BUCKET-NAME** im folgenden Befehl durch den Namen des ausgewählten Buckets.

```
S3CreateAndList BUCKET-NAME
```

Wenn Sie die Anwendung in der IDE ausführen, wählen Sie Projekt, S3CreateAndList Properties, Debug genund 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-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 [Schnellmenü](#) zurück oder gehen Sie direkt zum [Ende dieser Schnellreise](#).

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 for .NET](#), möchten Sie vielleicht [mit Ihrem Projekt beginnen](#).

Starte ein neues Projekt

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 for .NET](#), finden Sie vollständige Beispiele unter [Machen Sie einen kurzen Rundgang](#). 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 enthält C#-Projektvorlagen für eine Reihe von AWS-Services. 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 unter [Codebeispiele mit Anleitungen](#).

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.SSO0IDC`

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

AWSRegion konfigurieren

AWSRegionen 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 [Amazon EC2 EC2-Client](#) in einer vorhandenen Region instanziiieren. Es verwendet definierte Felder. [RegionEndpoint](#)

```
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, wie Sie die ServiceURL Eigenschaft der Service-Client-Konfigurationsklasse verwenden, um die Region anzugeben. In diesem Fall verwenden Sie die [AmazonEC2Config-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 for .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 `set x`), wirkt sich dies auf alle SDKs und Toolkits aus, nicht nur auf AWS SDK for .NET

AWSConfigs Klasse

Als festgelegt [AWSConfigs.AWSRegion](#)Eigentum.

```
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 wirdAWS-Region, wird AWS SDK for .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. Amazon EC2 EC2-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 [Endgeräte 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.

AWSSDK Pakete installieren mit NuGet

[NuGet](#) ist 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. NuGet ist 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 unter [Abrufen von AWSSDK Baugruppen](#). Informationen zur Verwendung der lokal gespeicherten Assemblys finden Sie unter [Installieren 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

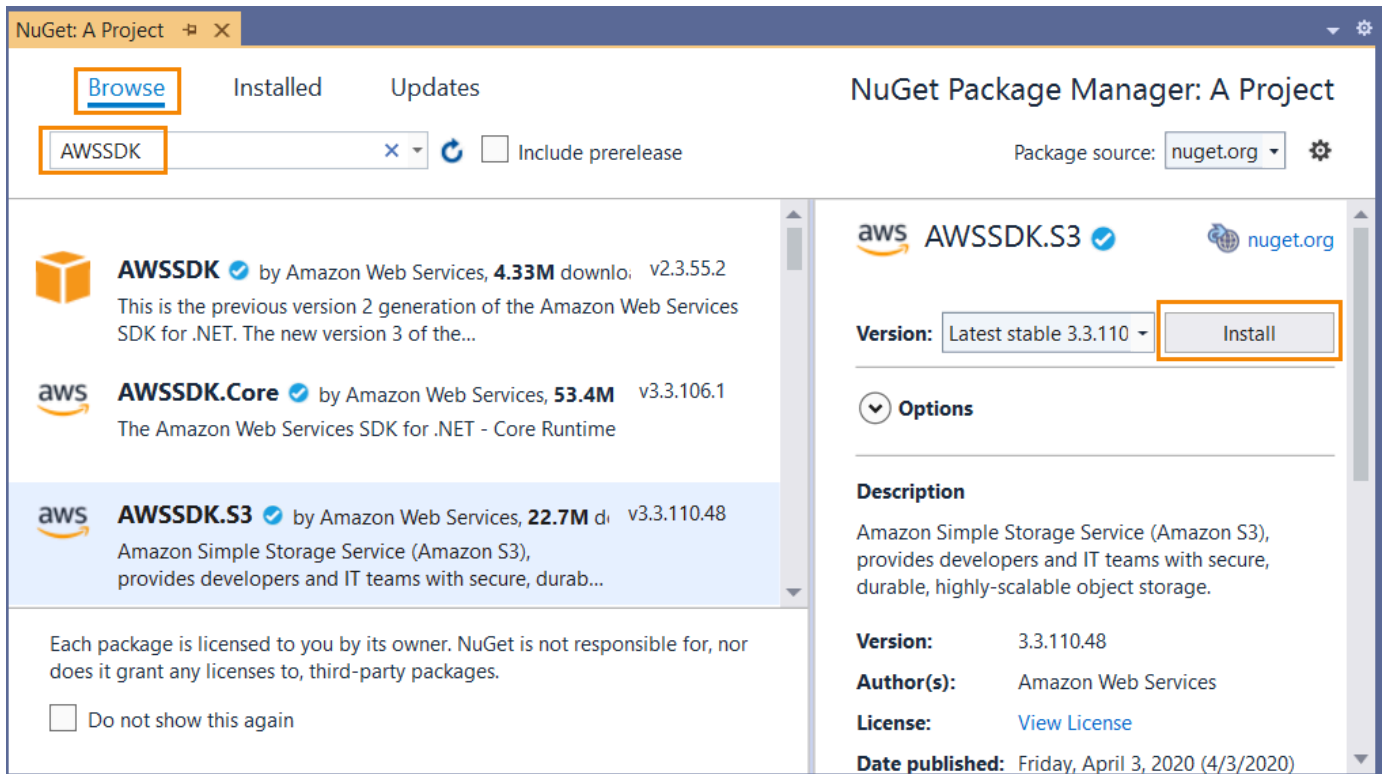
Wenn Sie nur die neueste Version eines NuGet Pakets benötigen, können Sie Versionsinformationen aus dem Befehl ausschließen, wie im folgenden Beispiel gezeigt.

```
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-Assemblies verwenden können, die Sie lokal (oder vor Ort) erhalten und gespeichert haben, wie unter [Abrufen von AWSSDK Baugruppen](#) aus. Das ist nicht die empfohlene Methode zur Handhabung von SDK-Referenzen, ist jedoch in einigen Umgebungen erforderlich.

Note

Die empfohlene Methode zur Handhabung von SDK-Referenzen besteht darin, nur die NuGet-Pakete herunterzuladen und zu installieren, die jedes Projekt benötigt. Diese Methode wird unter [AWSSDK Pakete installieren mit NuGet](#) aus.

So installieren Sie AWSSDK-Baugruppen

1. Erstellen Sie in Ihrem Projektbereich einen Ordner für die erforderlichen AWSSDK-Assemblies. Sie können diesen Ordner beispielsweise aufrufen `AwsAssemblies` aus.
2. Wenn Sie dies noch nicht getan haben, [erhalten Sie die AWSSDK-Assemblies](#), der die Assemblys in einem lokalen Download- oder Installationsordner ablegt. Kopieren Sie die DLL-Dateien für die erforderlichen Assemblys aus diesem Download-Ordner in Ihr Projekt (in die `AwsAssemblies` Ordner, in unserem Beispiel).

Kopieren Sie auch alle Abhängigkeiten. Informationen zu Abhängigkeiten finden Sie auf [GitHub](#) Website.

3. Verweisen Sie wie folgt auf die erforderlichen Baugruppen.

Cross-platform development

1. Öffnen Sie Ihre Projekte `.csproj` Datei und füge ein `<ItemGroup>` element.
2. In der `<ItemGroup>` Element, füge ein `<Reference>` Element mit einem `Include`-Attribut für jede erforderliche Baugruppe.

Für Amazon S3 würden Sie beispielsweise die folgenden Zeilen zu Ihrem Projekt hinzufügen `.csproj` file.

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 Ihre Projekte .csprojfile.

Windows with Visual Studio and .NET Core

1. Laden Sie Ihr Projekt in Visual Studio und öffnen Sie Projekt, Referenz hinzufügen aus.
2. Wählen Sie das Symbol Durchsuchen Schaltfläche unten im Dialogfeld. Navigieren Sie zum Ordner Ihres Projekts und zu dem Unterordner, in den Sie die erforderlichen DLL-Dateien kopiert haben (AwsAssemblies zum Beispiel).
3. Wählen Sie alle DLL-Dateien aus, wählen Sie Add, und wählen OKAY aus.
4. Speichern Sie Ihr Projekt.

Auflösung von Anmeldeinformationen und Profilen

Der AWS SDK for .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 [beschrieben](#) [Zugreifen 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. Ein Anmeldeinformationsprofil mit dem Namen, der durch einen Wert in angegeben wird [AWSConfigs. AWSProfileName](#).
3. Ein Anmeldeinformationsprofil mit dem in der AWS_PROFILE Umgebungsvariablen angegebenen Namen.
4. Das [default]-Anmeldeinformationsprofil.
5. [Sitzungen AWSCredentials](#), die aus den AWS_SESSION_TOKEN Umgebungsvariablen AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY, und erstellt werden, sofern sie alle nicht leer sind.
6. [Basic AWSCredentials](#), die aus den Umgebungsvariablen AWS_ACCESS_KEY_ID und AWS_SECRET_ACCESS_KEY Umgebungsvariablen erstellt werden, wenn sie beide nicht leer sind.
7. [IAM-Rollen für Aufgaben](#) für Amazon ECS-Aufgaben.
8. Metadaten der Amazon EC2 EC2-Instanz.

Wenn Ihre Anwendung auf einer Amazon EC2 EC2-Instanz ausgeführt wird, z. B. in einer Produktionsumgebung, verwenden Sie eine IAM-Rolle, wie unter beschrieben. [Gewähren des Zugriffs mithilfe einer IAM-Rolle](#) Andernfalls, z. B. bei Tests vor einer Vorabversion, speichern Sie Ihre Anmeldeinformationen in einer Datei, die das Format der AWS Anmeldeinformationsdatei verwendet, auf das Ihre Webanwendung auf dem Server Zugriff hat.

Auflösung des Profils

Da es zwei verschiedene Speichermechanismen für Anmeldeinformationen gibt, ist es wichtig zu verstehen, wie diese konfiguriert werden müssenAWS SDK for .NET, um sie zu verwenden. Das [AWSConfigs. AWSProfilesLocation](#)Die Eigenschaft steuert, wie die Anmeldeinformationsprofile AWS SDK for .NET findet.

AWSProfilesLocation	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 Anmeldeinformationen. Wenn sich das Profil an keinem dieser Speicherorte befindet, suchen Sie ~/ .aws/co

AWSProfilesLocation	Verhalten bei der Profilauflösung nfig (Linux oder macOS) oder %USERPROFILE%.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 for .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 non-domain-joined Konten 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.AWSProfileName](#)Eigenschaft auf die gleiche Weise, wie Sie es mit anderen Zugangsprofilen 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 Amazon EC2 EC2-Instances ausgeführt werden, ist die sicherste Methode zur Verwaltung von Anmeldeinformationen die Verwendung von IAM-Rollen, wie unter beschrieben.

[Gewähren des Zugriffs mithilfe einer IAM-Rolle](#)

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: [AmazonCloudDirectoryConfig](#) im [Amazon. CloudDirectory](#) Namespace und [AmazonGameLiftConfig](#) im [Amazon. GameLift](#) Namespace.

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 [NetworkCredential](#) Objekt 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.

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 bei der Entwicklung hat es sich bewährt, Benutzer und Berechtigungssätze in AWS IAM Identity Center zu erstellen 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 [Konfigurieren Sie die SDK-Authentifizierung](#) erhalten haben. Wenn Sie über die entsprechenden AWS Management Console-Berechtigungen verfügen, können Sie auch verschiedene Berechtigungssätze mit der geringsten Berechtigung für diesen Benutzer erstellen oder neue Benutzer speziell für Entwicklungsprojekte erstellen, indem Sie Berechtigungssätze mit der geringsten Berechtigung 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 Referenzhandbuch für AWS SDKs und Tools sowie unter [Erste Schritte](#) im Benutzerhandbuch für AWS IAM Identity Center.

Servicerollen

Sie können eine AWS-Servicerolle einrichten, um im Namen von Benutzern auf AWS-Services zuzugreifen. Diese Art des Zugriffs ist geeignet, wenn mehrere Personen Ihre Anwendung remote ausführen, z. B. auf einer Amazon-EC2-Instance, die Sie für diesen 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 AWS Management Console an 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 aus, suchen und wählen Sie (zum Beispiel) EC2 und dann (zum Beispiel) den EC2-Anwendungsfall aus.
4. Wählen Sie Weiter: Berechtigungen und wählen Sie die [entsprechenden Richtlinien](#) für die AWS Dienste aus, die Ihre Anwendung verwenden wird.

 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 Amazon-EC2-Instances ausgeführt werden.

Erweiterte Konfiguration für Ihre AWS SDK for .NET Projekt

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

Themen

- [Verwenden von AWSSDK .extensions.netCore.Setup und der IConfiguration-Schnittstelle](#)
- [Konfigurieren von anderen Anwendungsparametern](#)
- [Referenz der Konfigurationsdateien für AWS SDK for .NET](#)

Verwenden von AWSSDK .extensions.netCore.Setup und der IConfiguration-Schnittstelle

(Dieses Thema trug früher den Titel „Konfiguration AWS SDK for .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 for .NET mit .NET Core zu vereinfachen, können Sie das Paket [AWSSDK NuGet .Extensions.NETCore.Setup](#) verwenden. Wie viele andere .NET Core-Bibliotheken fügt es der IConfiguration Schnittstelle Erweiterungsmethoden hinzu, um eine reibungslose Konfiguration zu gewährleisten. AWS

Verwenden von AWSSDK .extensions.NETCore.Setup

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

```
public Startup(IConfiguration configuration)
```



```
{  
    Configuration = configuration;  
}
```

Um das `Configuration` Objekt zum Abrufen der `AWSOptions` 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 Amazon EC2 EC2-Instance bereitstellen, die auf `Production` **EnvironmentName** eingestellt ist, wird diese Datei ignoriert und es werden die AWS SDK for .NET IAM-Anmeldeinformationen und die Region verwendet, die für die Amazon EC2 EC2-Instance konfiguriert sind.

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

```
{  
  "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 `GetAWSSOptions` 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.GetAWSSOptions();
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.GetAWSSOptions("service1");
```

Zulässige Werte in der Appsettings-Datei

In der `appsettings.Development.json`-Datei können Sie die folgenden Werte für die Anwendungskonfiguration festlegen. Die Feldnamen müssen die angegebene Groß- und Kleinschreibung verwenden. 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

Das AWSSDK NuGet `.Extensions.NETCore.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 Pakete [AWSSDK.S3](#) und [AWSSDK.DynamoDBv2](#) NuGet 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](#)
- [AWSVom Dienst generierte Endpunkte](#)

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

Weitere Informationen zur Verwendung des <aws> Elements, wie später in diesem Thema beschrieben, finden Sie unter [Referenz für Konfigurationsdateien. AWS SDK for .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 for .NET API die Werte der [LoggingOptions](#)-Aufzählung 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.

AWSLogMetrics

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 for .NET API einzurichten, legen Sie alternativ den Wert fest. `AWSConfigs.LogMetrics`Eigenschaft:](#)

```
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 for .NET API festzulegen, legen Sie alternativ den `AWSConfigs.AWSRegion`Eigenschaft:](#)

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

Weitere Informationen zum Erstellen eines AWS Kunden für eine bestimmte Region finden Sie unter [AWSRegionauswahl](#). Änderungen an dieser Einstellung werden nur für neue AWS-Client-Instances wirksam.

AWSResponseLogging

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

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

Um die Serviceprotokollierung mit der AWS SDK for .NET API einzurichten, legen Sie alternativ die fest [AWSConfigs.ResponseLogging](#)Eigenschaft auf einen der Werte der [ResponseLoggingOption](#)Aufzählung:

```
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 for .NET API festzulegen, legen Sie alternativ die Eigenschaft [AWSConfigs.dynamoDB ContextTableNamePrefix](#) 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 for .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 `AWSEndpointDefinition` Schlüssel im folgenden `<appSettings>` Abschnitt festlegen:

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

Um die Endpunktdefinitionsdatei mit der AWS SDK for .NET API festzulegen, legen Sie alternativ den fest [AWSConfigs.EndpointDefinition](#) Eigenschaft:

```
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 Datei `endpoint.json` ist verfügbar unter <https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json>

AWSVom 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 [CreateDomainRequest](#) Objekt und stellen Sie die `DomainName` Eigenschaft bereit. Erstellen Sie ein [AmazonCloudSearchClient](#) Objekt mithilfe des Anforderungsobjekts. Rufen Sie die [CreateDomain](#)-Methode auf. Das vom Aufruf zurückgegebene [CreateDomainResponse](#) Objekt enthält eine `DomainStatus` Eigenschaft, die `DocService` sowohl den als auch den `SearchService` Endpunkt hat. Erstellen Sie ein [AmazonCloudSearchDomainConfig](#) Objekt und verwenden Sie es, um `SearchService` Instanzen der `DocService` [AmazonCloudSearchDomainClient](#) Klasse 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);
}
```

Beispiel für AWS IoT-Endpunkte

Um den Endpunkt für zu ermitteln AWS IoT, erstellen Sie ein [AmazonIoTClient-Objekt](#) und rufen Sie die Methode auf [DescribeEndPoint](#). Das zurückgegebene [DescribeEndPointResponse](#) Objekt enthält die `EndpointAddress`. Erstellen Sie ein [AmazonIoTDataConfig](#) Objekt, legen Sie die `ServiceURL` Eigenschaft fest und verwenden Sie das Objekt, um die Klasse zu instanziiieren [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 der Konfigurationsdateien für AWS SDK for .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 EC2 und Amazon S3 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 for .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, 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 for .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](#)
- [dynamoDBContext](#)
- [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 for .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 for .NET.

toTable

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

profileName

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

profilesLocation

Der absolute Pfad zum Speicherort der Anmeldeinformationsdatei, die mit anderen AWS SDKs gemeinsam genutzt wird. Standardmäßig wird die Anmeldeinformationsdatei im `.aws`-Verzeichnis im Basisordner des aktuellen Benutzers gespeichert. Dieses Attribut entspricht der `Amazon.AWSConfigs.AWSProfilesLocation`-Eigenschaft im AWS SDK for .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 for .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.Util.DynamoDBv2.DynamoDBEntryConversion`-Klasse im AWS SDK for .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>
```

dynamoDBContext

Das `<dynamoDBContext>`-Element repräsentiert eine Sammlung von kontextspezifischen Einstellungen für Amazon DynamoDB. Dieses Element kann das `tableNamePrefix`-Attribut 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 for .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 repräsentiert eine Sammlung von Einstellungen für Amazon EC2. 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 Signaturversion 4 nicht für alle Anfragen verwendet wird (`false`, Standard). Dieses Attribut entspricht der `Amazon.Util.EC2Config.UseSignatureVersion4`-Eigenschaft aus der `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4`-Eigenschaft im AWS SDK for .NET.

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

Legt fest, wohin das Protokoll geschrieben werden soll (entspricht der `LogTo`-Eigenschaft aus der `Amazon.AWSConfigs.LoggingConfig.LogTo`-Eigenschaft im AWS SDK for .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 for .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 for .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 for .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 for .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 for .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 for .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 for .NET`.

name

Der Name der Eigenschaft. Dieses Attribut entspricht der `Amazon.Util.PropertyConfig.Name`-Eigenschaft im `AWS SDK for .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 for .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 for .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.AWSCfgs.ProxyConfig.Host`-Eigenschaft im AWS SDK for .NET.

password

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

port

Die Portnummer des Proxys. Dieses Attribut entspricht der `Amazon.Util.ProxyConfig.Port`-Eigenschaft aus der `Amazon.AWSCfgs.ProxyConfig.Port`-Eigenschaft im AWS SDK for .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 for .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 for .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 eigens entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie [AWS IAM Identity Center](#).

ℹ 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 für AWS SDKs und Tools. Als bewährte Sicherheitsmethode verwenden Sie AWS IAM Identity Center, wie unter [Konfigurieren Sie die SDK-Authentifizierung](#) beschrieben.

Wichtige Warnhinweise und Richtlinien für Anmeldeinformationen

Warnhinweise für Anmeldeinformationen

- Verwenden Sie NICHT die Root-Anmeldeinformationen Ihres Kontos, um auf Ihre AWS-Ressourcen zuzugreifen. Diese Anmeldeinformationen bieten uneingeschränkten Zugriff auf Konten und können nur schwer widerrufen werden.
- Fügen 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 die Anmeldeinformationen in der freigegebenen AWS-Datei `credentials` im Klartext gespeichert werden.

Zusätzliche Hinweise zur sicheren Verwaltung von Anmeldeinformationen

Eine allgemeine Erläuterung der sicheren Verwaltung von AWS Anmeldeinformationen finden Sie unter [AWSSicherheitsanmeldedaten](#) im [Allgemeine AWS-ReferenzIAM-Benutzerhandbuch](#) und

[Bewährte Sicherheitsmethoden und Anwendungsfälle](#). Zusätzlich zu diesen Diskussionen sollten Sie Folgendes berücksichtigen:

- 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 Amazon-EC2-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 den anderen AWS SDKs verwendet werden. Sie kann auch von den [AWS CLI/AWS Tools for Windows PowerShell](#), und den AWS Toolkits für [Visual Studio](#) und [VS Code](#) [JetBrains](#) verwendet werden.

⚠ Warning

Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie eigens entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie [AWS IAM Identity Center](#).

ℹ 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 für AWS SDKs und Tools.

Als bewährte Sicherheitsmethode verwenden Sie AWS IAM Identity Center, wie unter [Konfigurieren Sie die SDK-Authentifizierung](#) beschrieben.

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 im [Referenzhandbuch für AWS SDKs und Tools](#) unter [Speicherort der gemeinsam genutzten Dateien](#). Lesen Sie auch [Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung](#).

Die Datei mit den gemeinsam genutzten AWS Anmeldeinformationen ist eine Klartextdatei und folgt einem bestimmten Format. Informationen zum Format von AWS Anmeldeinformationsdateien finden Sie unter [Format der Anmeldeinformationsdatei](#) im Referenzhandbuch für AWS SDKs und Tools.

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.CredentialManagement](#) Namespace der AWS SDK for .NET API, wie später in diesem Thema gezeigt wird.

- Verwenden Sie Befehle und Verfahren für die [AWS Tools for PowerShell](#) und die AWS Toolkits für [Visual Studio](#) und [VS Code](#). [JetBrains](#)
- Verwenden Sie [AWS CLI](#) Befehle, z. B. `aws configure set aws_access_key_id undaws 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 for .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.CredentialManagement](#) Namespace: [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.CredentialManagement](#)Namespace: und. [CredentialProfileSharedCredentialsFile](#) Es verwendet auch die [RegionEndpoint](#)Klasse des [Amazon-Namespaces](#).

```
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 [AWSRegion konfigurieren](#).

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 for .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 eigens 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 für AWS SDKs und Tools.

Als bewährte Sicherheitsmethode verwenden Sie AWS IAM Identity Center, wie unter [Konfigurieren Sie die SDK-Authentifizierung](#) beschrieben.

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 PowerShell](#) und den [AWS Toolkit for Visual Studio](#) bereit.

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.CredentialManagement](#) Namespace der AWS SDK for .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 Ihnen, 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.CredentialManagement](#) [Namespace: CredentialProfileOptions](#), und [NetSDK CredentialProfile.CredentialsFile](#)

```
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.

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

```
"[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: CredentialProfile](#) und [NetSDK.CredentialsFile](#) Es verwendet auch die [RegionEndpoint](#) Klasse des [Amazon-Namespaces](#).


```
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 [AWSRegion konfigurieren](#).

Funktionen des AWS SDK for .NET

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

Stellen Sie sicher, dass Sie [Ihr Projekt zuerst eingerichtet](#) haben.

Informationen zur Entwicklung von Software für bestimmte -AWSServices sowie Codebeispiele finden Sie unter [Arbeiten mit - AWS Services](#). Weitere Codebeispiele finden Sie unter [AWS SDK for .NET Codebeispiele](#).

Themen

- [AWS-asynchrone APIs für .NET](#)
- [Wiederholungen und Timeouts](#)
- [Paginatoren](#)
- [Zusätzliche Tools](#)

AWS-asynchrone APIs für .NET

Das AWS SDK for .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 for .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: Task-Klasse, Task-Klasse.](#) <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 `async` 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 [Machen Sie einen kurzen Rundgang](#) Abschnitten ([Einfache plattformübergreifende App](#) und [Einfache Windows-basierte App](#)) und in [Codebeispiele mit Anleitungen](#).

Wiederholungen und Timeouts

Mit der AWS SDK for .NET können Sie die Anzahl der Wiederholungen und die Timeout-Werte für HTTP-Anfragen an -AWSServices konfigurieren. 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 sollten die AWS SDK for .NET und Ihre Anwendung reagieren, wenn die Netzwerkkonnektivität nachlässt oder ein -AWS-Service 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 bereitgestellt oder wird sie an einem Remote-Standort mit unzuverlässiger Konnektivität bereitgestellt?

Wiederholversuche

Übersicht

Der AWS SDK for .NET kann Anforderungen wiederholen, die aufgrund serverseitiger Drosselung oder unterbrochener Verbindungen fehlschlagen. Es gibt zwei Eigenschaften von Service-Konfigurationsklassen, mit denen Sie das Wiederholungsverhalten eines Service-Clients angeben können. Service-Konfigurationsklassen erben diese Eigenschaften von der abstrakten [Amazon.Runtime.ClientConfig](#)-Klasse der [AWS SDK for .NET API-Referenz](#) :

- `RetryMode` gibt einen von drei Wiederholungsmodi an, die in der [Amazon.Runtime.RequestRetryMode](#)-Aufzählung definiert sind.

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

- `MaxErrorRetry` gibt die Anzahl der auf Service-Client-Ebene zulässigen Wiederholungsversuche an. Das SDK wiederholt den Vorgang mit der angegebenen Anzahl, bevor es fehlschlägt und eine Ausnahme auslöst.

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

Detaillierte Beschreibungen dieser Eigenschaften finden Sie in der abstrakten [Amazon.Runtime.ClientConfig](#)-Klasse der [AWS SDK for .NET API-Referenz zu](#) . Jeder Wert von `RetryMode` entspricht standardmäßig einem bestimmten Wert von `MaxErrorRetry`, wie in der folgenden Tabelle gezeigt.

RetryMode	Corresponding MaxErrorRetry (Amazon DynamoDB)	Corresponding MaxErrorRetry (all others)
Legacy	10	4
Standard	10	2
Adaptive (experimental)	10	2

Behavior

Wenn Ihre Anwendung gestartet wird

Wenn Ihre Anwendung gestartet wird, `MaxErrorRetry` werden die Standardwerte für `RetryMode` und vom SDK konfiguriert. Diese Standardwerte werden verwendet, wenn Sie einen Service-Client erstellen, es sei denn, Sie geben andere Werte an.

- Wenn die Eigenschaften in Ihrer Umgebung nicht festgelegt sind, `RetryMode` ist der Standardwert für als Legacy und der Standardwert für mit dem entsprechenden Wert aus der vorherigen Tabelle `MaxErrorRetry` konfiguriert.

- Wenn der Wiederholungsmodus in Ihrer Umgebung festgelegt wurde, wird dieser Wert als Standard für `verwendetRetryMode`. Der Standardwert für `MaxErrorRetry` wird mit dem entsprechenden Wert aus der vorherigen Tabelle konfiguriert, es sei denn, der Wert für maximale Fehler wurde auch in Ihrer Umgebung festgelegt (als Nächstes beschrieben).
- Wenn der Wert für maximale Fehler in Ihrer Umgebung festgelegt wurde, wird dieser Wert als Standard für `verwendetMaxErrorRetry`. Amazon DynamoDB ist die Ausnahme von dieser Regel. Der standardmäßige DynamoDB-Wert 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 `verwendenMaxErrorRetry`, wie zuvor beschrieben, oder Sie können andere Werte angeben. Um andere Werte anzugeben, erstellen Sie ein Servicekonfigurationsobjekt wie [AmazonDynamoDBConfig](#) oder [AmazonSQSConfig](#) und fügen Sie es ein, wenn Sie den Service-Client erstellen.

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

Überlegungen

Wenn ein Wiederholungsversuch stattfindet, wird die Latenz Ihrer Anfrage erhöht. Sie sollten Ihre Wiederholversuche basierend auf den Anwendungsbeschränkungen für die Gesamtanfragenlatenz und die Fehlerraten konfigurieren.

Timeouts

Mit dem AWS SDK for .NET können Sie die Timeout-Werte für die Anforderung und das Socket-Lese-/Schreib-Timeout auf Service-Client-Ebene konfigurieren. Diese Werte werden in der `Timeout` und den `ReadWriteTimeout` Eigenschaften der abstrakten [Amazon.Runtime.ClientConfig](#)-Klasse angegeben. Diese Werte werden als die `ReadWriteTimeout` Eigenschaften `Timeout` und der [HttpRequest](#) Objekte übergeben, 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 for .NET, die auf die portable Klassenbibliothek (PCL) abzielt, verwendet die [HttpClient](#)-Klasse 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 maximalen Werte gesetzt, wenn die aufgerufene Methode einen Stream hochlädt, z. B. [AmazonS3Client.PutObjectAsync\(\)](#), [AmazonS3Client.UploadPartAsync\(\)](#), [AmazonGlacierClient.UploadArchiveAsync\(\)](#) usw.
- Die Versionen des AWS SDK for .NET, die auf .NET Framework abzielen, setzen `Timeout` und `ReadWriteTimeout` auf die Höchstwerte für alle [AmazonS3Client](#)- und [AmazonGlacierClient](#)-Objekte.
- Die Versionen von AWS SDK for .NET, die auf die portable Klassenbibliothek (PCL) und .NET Core abzielen, sind `Timeout` und `ReadWriteTimeout` auf den Höchstwert für alle [AmazonS3Client](#)- und [AmazonGlacierClient](#)-Objekte festgelegt.

Beispiel

Das folgende Beispiel zeigt, wie Sie den Standard-Wiederholungsmodus, maximal 3 Wiederholungen, ein `Timeout` von 10 Sekunden und ein `Timeout` für Lese-/Schreibvorgänge 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 for .NET that target .NET Core.  
        ReadWriteTimeout = TimeSpan.FromSeconds(10),  
        RetryMode = RequestRetryMode.Standard,  
        MaxErrorRetry = 3  
    });
```

Paginatoren

Einige -AWSservices sammeln und speichern eine große Datenmenge, die Sie mithilfe der API-Aufrufe abrufen können. 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 (in der Regel als `NextToken` bezeichnet) bereit. Einige dieser Service-Clients stellen auch Paginatoren bereit.

Mit Paginatoren können Sie den Overhead des Fortsetzungstokens vermeiden, der Schleifen, Statusvariablen, mehrere API-Aufrufe usw. umfassen kann. Wenn Sie einen Paginator verwenden, können Sie Daten aus einem -AWSservice über eine einzelne Codezeile abrufen, die Deklaration einer `foreach` Schleife. Wenn mehrere API-Aufrufe erforderlich sind, um die Daten abzurufen, übernimmt der Paginator dies für Sie.

Wo finde ich Paginatoren?

Nicht alle Services stellen Paginatoren bereit. Eine Möglichkeit, zu bestimmen, ob ein Service einen Paginator für eine bestimmte API bereitstellt, besteht darin, die Definition einer Service-Client-Klasse in der [AWS SDK for .NET API-Referenz](#) zu anzusehen.

Wenn Sie beispielsweise die Definition für die [AmazonCloudWatchLogsClient](#) Klasse untersuchen, wird eine `-Paginators`Eigenschaft angezeigt. Dies ist die Eigenschaft, die einen Paginator für Amazon CloudWatch Logs bereitstellt.

Was geben mir Paginatoren?

Paginatoren enthalten Eigenschaften, mit denen Sie vollständige Antworten sehen können. Sie enthalten in der Regel auch eine oder mehrere Eigenschaften, mit denen Sie auf die interessantesten Teile der Antworten zugreifen können, die wir die Schlüsselergebnisse nennen werden.

In der oben [AmazonCloudWatchLogsClient](#) genannten enthält das `Paginator` Objekt beispielsweise eine `-Responses`Eigenschaft mit dem vollständigen [DescribeLogGroupsResponse](#) Objekt aus dem API-Aufruf. Diese `Responses` Eigenschaft enthält unter anderem eine Sammlung der Protokollgruppen.

Das Paginator-Objekt enthält auch ein Schlüsselergebnis mit dem Namen `LogGroups`. Diese Eigenschaft enthält nur den Teil der Antwort für Protokollgruppen. Mit diesem Schlüsselergebnis können Sie Ihren Code unter vielen Umständen reduzieren und vereinfachen.

Synchrone und asynchrone Paginierung

Paginatoren bieten sowohl synchrone als auch asynchrone Mechanismen für die Paginierung. Die synchrone Paginierung ist in .NET Framework 4.5 (oder höher)-Projekten verfügbar. Die 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 nächste Beispiel die asynchrone Paginierung. Informationen zum Ausführen derselben Aufgaben mit synchroner Paginierung und .NET Framework 4.5 (oder höher) finden Sie nach dem Beispiel unter [Zusätzliche Überlegungen für Paginatoren](#).

Beispiel

Das folgende Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um eine Liste von Protokollgruppen anzuzeigen. Im Gegensatz dazu zeigt das Beispiel, wie dies sowohl mit als auch ohne Paginatoren geschieht. Bevor Sie sich den vollständigen Code ansehen, der später gezeigt wird, sollten Sie die folgenden Ausschnitte berücksichtigen.

Abrufen von CloudWatch Protokollgruppen ohne Paginatoren

```
// 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 Ausschnitte sind genau identisch, sodass der Vorteil bei 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 [Microsoft.Bcl.AsyncInterfaces](#) NuGet package, da asynchrone Paginatoren die `-IAsyncEnumerable` Schnittstelle verwenden.

Vollständiger Code

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

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.CloudWatch](#)

Programmierelemente:

- Namespace [Amazon.CloudWatch](#)
 - Klasse [AmazonCloudWatchLogsClient](#)
- Namespace [AmazonCloudWatchLogs.Modell](#)
 - Klasse [DescribeLogGroupsRequest](#)
 - Klasse [DescribeLogGroupsResponse](#)
 - Klasse [LogGroup](#)

Vollcode

```
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}");
                }
            } while (response.NextToken != null);
        }
    }
}
```

```
    }
    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 für Paginatoren

- Paginatoren können nicht mehr als einmal verwendet werden

Wenn Sie die Ergebnisse eines bestimmten AWS Paginators an mehreren Speicherorten in Ihrem Code benötigen, dürfen Sie ein Paginatorobjekt nicht mehr als einmal verwenden. Erstellen Sie stattdessen bei jedem Bedarf einen neuen Paginator. Dieses Konzept wird im vorherigen Beispielcode in der `DisplayLogGroupsWithPaginators` Methode gezeigt.

- Synchroner Paginierung

Die synchrone Paginierung ist für .NET Framework 4.5 (oder höher)-Projekte verfügbar.

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

Um dies zu sehen, erstellen Sie ein .NET Framework 4.5 (oder höher)-Projekt und kopieren Sie den vorherigen Code hinein. Entfernen Sie dann einfach das `await` Schlüsselwort aus den beiden `foreach` Paginatorkaufrufen, 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 sehen, die Sie mit asynchroner Paginierung erzielt haben.

Zusätzliche Tools

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

AWS Bereitstellen des Tools

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 in bereitzustellenAWS.

Weitere Informationen finden Sie unter [Bereitstellen von Anwendungen in AWS](#).

AWS Framework zur Nachrichtenverarbeitung für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Wenn Sie Services 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 Framework zur Nachrichtenverarbeitung für .NET](#).

Erweiterte Authentifizierung und -Autorisierung mit AWS SDK for .NET

Die Themen in diesem Abschnitt enthalten Informationen über erweiterte Techniken für die Authentifizierung und -Autorisierung in Ihrer AWS SDK for .NET Anwendung.

Themen

- [Single Sign-On mit dem AWS SDK for .NET](#)

Single Sign-On mit dem AWS SDK for .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

Auf einer übergeordneten Ebene interagieren SDKs mit IAM Identity Center auf ähnliche Weise wie das folgende Muster:

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 wie das, das kein SDK ist 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 unter [Konfigurieren Sie die SDK-Authentifizierung](#). 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 [AWSZugriffsport](#) 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 Zugriffportal. 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 [AWSZugriffportal](#) 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-Konten Seite. Die Konto-ID, die Sie für diese Einstellung wählen, 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 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.SSO0IDC`

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 unter [Tutorial 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.SSO0IDC`

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

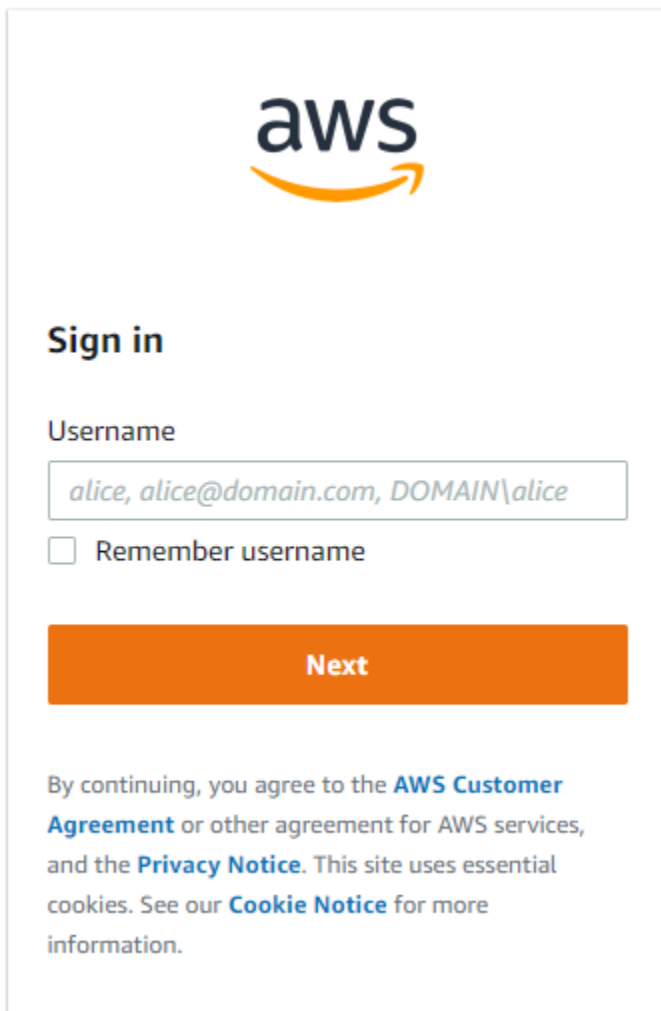
```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    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
        });
    };

    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:



aws

Sign in

Username

Remember username

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

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 Prozess 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-fähigen 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.SSO0IDC`

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.
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 for .NET Vollständige Informationen zur Verwendung von IAM Identity Center mit dem AWS SDK for .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 [Installieren und konfigurieren Sie Ihre Toolchain](#) und beschrieben [Erste 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.

Einrichten von 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. Gehen Sie dann wie folgt vor:

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 OrganizationsKonsole](#) 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 ConsoleAWS-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 aus.
 - c. Wählen Sie auf der Seite für Gruppen die Option Weiter aus, überprüfen Sie die Informationen und wählen Sie Benutzer hinzufügen aus.
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:

```
{
  "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 AWSKontoseite 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 den Tab „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.S3` und `AWSSDK.SecurityToken` zusätzlich zu `AWSSDK.SSO` und `AWSSDK.SSO0IDC` ein.

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

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSO0IDC
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
        });
    };

    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.SSO0IDC` 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.SSOIDC
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
        });
    };

    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 in der Eigenschaft `Options` eines SSO-Objekts eine Callback-Methode bereitstellen. `AWSCredentials`](#)

Weisen Sie den SSO-Benutzer an

Bitte 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 `.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. Dabei kann es sich AWS um Ressourcen oder Ressourcen in Ihrer Entwicklungsumgebung handeln, 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 for .NET. Vollständige Informationen zur Verwendung von IAM Identity Center mit dem AWS SDK for .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 [Installieren und konfigurieren Sie Ihre Toolchain](#) und beschrieben [Erste Schritte](#).
- Identifizieren oder erstellen Sie mindestens ein AWS-Konto, das 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.

Einrichten von 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. Gehen Sie dann wie folgt vor:

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 IAM

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 ConsoleAWS-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 aus.
 - 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:

```
{
  "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.
 - f. Notieren Sie sich den Namen des Berechtigungssatzes für die spätere Verwendung in der `sso_role_name` Einstellung.
6. Gehen Sie zur AWSKontoseite 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.S3` und `AWSSDK.SecurityToken` zusätzlich zu `AWSSDK.S3` und `AWSSDK.SecurityToken` ein.

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

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSO0IDC
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.SSO0IDC` zusätzlich zu `AWSSDK.IdentityManagement` und `AWSSDK.SecurityToken` ein.

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

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
AWSSDK.SSO0IDC
```



```
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

Bitte 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.

- 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
```

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

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

- 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.
- Führen Sie die Amazon S3 S3-Anwendung erneut aus. Die Anwendung zeigt jetzt die Liste der S3-Buckets an.
- 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.

Bereitstellen von Anwendungen in AWS

Nachdem Sie Ihre cloudnative .NET Core-Anwendung oder Ihren cloudnativen .NET Core-Service auf einem Entwicklungscomputer entwickelt haben, möchten Sie sie auf bereitstellenAWS. Sie können dies tun, indem Sie die AWS Management Console oder bestimmte Services wie AWS CloudFormation oder verwendenAWS Cloud Development Kit (AWS CDK). Sie können auch AWS Tools verwenden, die für die Bereitstellung erstellt wurden. Mit diesen Tools können Sie Folgendes tun.

Bereitstellen über die .NET-CLI

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

- [AWS Deploy Tool for .NET CLI](#) - Unterstützt Bereitstellungen in [AWS App Runner](#), [Amazon Elastic Container Service \(Amazon ECS\)](#) und [AWS Elastic Beanstalk](#).
- [AWS Lambda Tools for .NET CLI](#) - Unterstützt Bereitstellungen von AWS Lambda Projekten.

Bereitstellen aus den IDE-Toolkits

Sie können AWS Toolkits verwenden, um Ihre Anwendungen direkt über die IDE Ihrer Wahl bereitzustellen:

- [AWS Toolkit for Visual Studio](#)

Note

Die Funktion „Veröffentlichen inAWS“ im Toolkit stellt die gleiche Funktionalität wie die AWS Deploy Tool for .NET CLI bereit. Weitere Informationen finden Sie unter [Veröffentlichen in AWS](#) im AWS Toolkit for Visual Studio -Benutzerhandbuch.

- [AWS Toolkit for JetBrains](#)

Weitere Informationen finden Sie unter [Arbeiten mit AWS Serverless-Anwendungen](#) und [Arbeiten mit AWS App Runner](#).

- [AWS Toolkit für VS-Code](#)

Weitere Informationen finden Sie unter [Arbeiten mit Serverless-Anwendungen](#) und [Verwenden von AWS App Runner](#).

- [AWS Toolkit for Azure DevOps](#)

Anwendungsfälle

Die folgenden Abschnitte enthalten Anwendungsfallsszenarien für bestimmte Arten von Anwendungen, einschließlich Informationen darüber, wie Sie die .NET-CLI verwenden würden, um diese Anwendungen bereitzustellen.

- [ASP.NET Core-Apps](#)
- [.NET-Konsolen-Apps](#)
- [Blazor- WebAssembly Apps](#)
- [AWS Lambda-Projekte](#)

ASP.NET Core-Apps

Das [AWS Deploy Tool](#) für die .NET CLI hilft Ihnen bei der Bereitstellung Ihrer ASP.NET-Anwendungen und führt Sie durch einen Bereitstellungsprozess. Es handelt sich um ein interaktives Tool für die .NET-CLI, mit dem .NET-Anwendungen mit minimalem AWS Wissen bereitgestellt werden können.

Das Tool Deploy verfügt über die folgenden Funktionen:

- Computing-Empfehlungen für Ihre Anwendung – Holen Sie sich die Computing-Empfehlungen und erfahren Sie, welche AWS Datenverarbeitung für Ihre Anwendung am besten geeignet ist.
- Dockerfile-Generierung – Das Tool generiert bei Bedarf ein Dockerfile oder verwendet ein vorhandenes Dockerfile.
- Automatische Paketierung und Bereitstellung – Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf der ausgewählten AWS Rechenleistung 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 für wiederholbare Bereitstellungen mit Ihrem Team teilen.

- Hilfe beim Lernen AWS CDK für .NET – Das Tool hilft Ihnen dabei, schrittweise die zugrunde liegenden AWS Tools zu erlernen, auf denen es aufbaut, z. B. AWS CDK.

Das [AWS Deploy Tool](#) unterstützt die Bereitstellung von ASP.NET Core-Anwendungen für die folgenden AWS Services:

- [Amazon ECS Service](#) mit [AWS Fargate](#) - Unterstützt Bereitstellungen von Webanwendungen für Amazon Elastic Container Service (Amazon ECS) mit Rechenleistung, die von einer AWS Fargate Serverless-Rechen-Engine verwaltet wird.
- [AWS App Runner](#) – Unterstützt Bereitstellungen für einen vollständig verwalteten Service, der es Entwicklern erleichtert, containerisierte Webanwendungen und APIs in großem Umfang bereitzustellen. Es ist keine vorherige Infrastrukturerfahrung erforderlich.
- [AWS Elastic Beanstalk](#) – Unterstützt Bereitstellungen für einen Service, der es Entwicklern erleichtert, Webanwendungen und APIs in einer vollständig verwalteten Umgebung in großem Umfang bereitzustellen. Es ist keine vorherige Infrastrukturerfahrung erforderlich.

Weitere Informationen finden Sie in der [Toolübersicht](#) . Um von dort aus zu beginnen, navigieren Sie zu Dokumentation , Erste Schritte und wählen [Sie Installieren von](#) für Installationsanweisungen aus.

.NET-Konsolen-Apps

Das [AWS Deploy Tool](#) für die .NET-CLI hilft Ihnen bei der Bereitstellung Ihrer .NET-Konsolenanwendungen als Service oder als geplante Aufgabe als Container-Image unter Linux und führt Sie durch einen Bereitstellungsprozess. Wenn Ihre Anwendung über kein Dockerfile verfügt, generiert das Tool es automatisch. Andernfalls wird ein vorhandenes Dockerfile verwendet.

Das Tool Deploy verfügt über die folgenden Funktionen:

- Computing-Empfehlungen für Ihre Anwendung – Holen Sie sich die Computing-Empfehlungen und erfahren Sie, welche AWS Datenverarbeitung für Ihre Anwendung am besten geeignet ist.
- Dockerfile-Generierung – Das Tool generiert bei Bedarf ein Dockerfile oder verwendet ein vorhandenes Dockerfile.
- Automatische Paketierung und Bereitstellung – Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf der ausgewählten AWS Rechenleistung 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 für wiederholbare Bereitstellungen mit Ihrem Team teilen.
- Hilfe beim Lernen AWS CDK für .NET – Das Tool hilft Ihnen dabei, schrittweise die zugrunde liegenden AWS Tools zu erlernen, auf denen es aufbaut, z. B. AWS CDK.

Das [AWS Deploy Tool](#) unterstützt die Bereitstellung von .NET-Konsolenanwendungen für die folgenden AWS Services:

- [Amazon ECS Service](#) mit [AWS Fargate](#) - Unterstützt Bereitstellungen von .NET-Anwendungen als Service (z. B. einen Hintergrundprozessor) für Amazon Elastic Container Service (Amazon ECS) mit Rechenleistung, die von AWS Fargate Serverless-Compute-Engine verwaltet wird.
- [Geplante Amazon-ECS-Aufgabe](#) mit [AWS Fargate](#) - Unterstützt Bereitstellungen von .NET-Anwendungen als geplante Aufgabe (z. B. end-of-day Prozess) für Amazon ECS mit Rechenleistung, die von AWS Fargate Serverless-Compute-Engine verwaltet wird.

Weitere Informationen finden Sie in der [Toolübersicht](#) . Um von dort aus zu beginnen, navigieren Sie zu Dokumentation , Erste Schritte und wählen [Sie Installieren von](#) für Installationsanweisungen aus.

Blazor- WebAssembly Apps

Das [AWS Deploy Tool](#) für die .NET-CLI unterstützt Sie beim Hosten Ihrer Blazor- WebAssembly Anwendung in Amazon S3, wobei Amazon CloudFront für die Bereitstellung von Inhaltsnetzwerken verwendet wird. 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:

- Automatische Paketierung und Bereitstellung – Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf der ausgewählten AWS Rechenleistung 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 für wiederholbare Bereitstellungen mit Ihrem Team teilen.

- Hilfe beim Lernen AWS CDK für .NET – Das Tool hilft Ihnen dabei, schrittweise die zugrunde liegenden AWS Tools zu erlernen, 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 Installieren von](#) für Installationsanweisungen aus.

AWS Lambda-Projekte

AWS Lambda ist ein Datenverarbeitungsservice, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Der Service führt Ihren Code auf einer hoch verfügbaren Datenverarbeitungsinfrastruktur aus und erledigt die gesamte Administration der Datenverarbeitungsressourcen. Weitere Informationen zu Lambda finden Sie unter [Was ist ? AWSLambda?](#) 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 die .NET CLI für die Bereitstellung von Lambda-Funktionen verwenden, müssen die folgenden Voraussetzungen erfüllt sein:

- Vergewissern Sie sich, dass Sie die .NET CLI installiert haben. Zum Beispiel: `dotnet --version`. Gehen Sie bei Bedarf zu <https://dotnet.microsoft.com/download> um es zu installieren.
- Richten Sie die .NET CLI für die Arbeit mit Lambda ein. Eine Beschreibung dazu finden Sie unter [.NET Core-CLI](#) im AWS Lambda Entwicklerhandbuch. In diesem Verfahren ist das Folgende der Deployment-Befehl:

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

Wenn Sie nicht sicher sind, wie Sie für diese Übung eine IAM-Rolle erstellen können, schließen Sie die `--function-role role` Teil. 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 Eingabeaufforderung oder ein Terminal und geben `dotnet lambda --help`. Die Befehlsausgabe sieht folgendermaßen oder ähnlich aus:

```
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

Bei den folgenden Anleitungen wird davon ausgegangen, dass Sie einen erstellt habenAWS Lambda.NET-Projekt. Für die Zwecke dieses Verfahrens wird das Projekt benanntDotNetCoreLambdaTest.

1. Öffnen Sie eine Eingabeaufforderung oder ein Terminal und navigieren Sie zum Ordner, der die .NET Lambda-Projektdatei enthält.
2. Geben Sie `dotnet lambda deploy-function` ein.
3. Wenn Sie dazu aufgefordert werden, geben SieAWSRegion (die Region, für die Ihre Lambda-Funktion bereitgestellt wird).
4. Geben Sie auf Aufforderung den Namen der bereitzustellenden Funktion ein, z. B.DotNetCoreLambdaTest. Dies kann der Name einer Funktion sein, die bereits in IhremAWS-Konto oder eine, die dort noch nicht eingesetzt wurde.
5. Wählen Sie auf Aufforderung die IAM-Rolle aus bzw. erstellen Sie diese, die Lambda für die Ausführung der Funktion übernimmt.

Nach erfolgreichem Abschluss wird die angezeigte neue Lambda-Funktion erstellt angezeigt werden.

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

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

Nachdem Ihre Lambda-Funktion bereitgestellt wurde, kann sie verwendet werden. Weitere Informationen finden Sie unter [Beispiele für die Verwendung von AWS Lambda](#).

Lambda überwacht automatisch Lambda-Funktionen für Sie und meldet Metriken über Amazon CloudWatch. Informationen zur Überwachung und Problembehandlung Ihrer Lambda-Funktion finden Sie unter [Überwachung von und Fehlerbehebung bei Lambda-Anwendungen](#).

Migrieren Sie Ihr Projekt für die AWS SDK for .NET

Dieser Abschnitt enthält Informationen zu Migrationsaufgaben, die für Sie relevant sein könnten, sowie Anweisungen zur Durchführung dieser Aufgaben.

Themen

- [Was ist neu im AWS SDK for .NET](#)
- [Vom unterstützte Plattformen AWS SDK for .NET](#)
- [Migrieren auf AWS SDK for .NET-Version 3](#)
- [Migration auf -Version 3.5AWS SDK for .NET](#)
- [Migration auf Version 3.7AWS SDK for .NET](#)
- [Migration von .NET-Standard 1.3](#)

Was ist neu im AWS SDK for .NET

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

Im Folgenden finden Sie die Neuerungen in der AWS SDK for .NET.

23. Februar 2024: Unterstützung für .NET 8 hinzugefügt

Unterstützung für .NET 8 wurde hinzugefügt AWS SDK for .NET. Verwenden Sie die neuesten [NuGet Pakete](#) oder die [Baugruppen, die .NET 8 und höher unterstützen](#). Weitere Informationen zu diesem Support, einschließlich [Support für Lambda](#), finden Sie im Blogbeitrag [.NET 8 Support auf AWS](#).

18. Februar 2024: Bevorstehende Änderungen am .NET Framework-Support

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

2023-07-17: Das AWS Lambda Annotations-Framework wurde für die allgemeine Verfügbarkeit veröffentlicht

Das [AWS Lambda Annotations-Framework](#) macht die Erfahrung beim Schreiben von Lambda-Funktionen in C# für .NET-Entwickler natürlicher, indem es C#-Quellgeneratortechnologie verwendet. Sie ist jetzt allgemein verfügbar.

2023-07-15: Der Distributed Cache Provider für DynamoDB wurde in der Vorschau veröffentlicht

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Mit der Bibliothek Distributed Cache Provider kann Amazon DynamoDB als Speicher für das verteilte Cache-Framework von ASP.NET Core verwendet werden. Weitere Informationen finden Sie im Blogbeitrag [Introducing the AWS .NET Distributed Cache Provider for DynamoDB \(Preview\)](#) and the [GitHub repository](#) .

2022-07-13: Das AWS Deploy Tool wurde veröffentlicht

Das AWS Deploy Tool wurde veröffentlicht. Dieses Tool ist ein interaktives Tool für die .NET-CLI und die AWS Toolkit for Visual Studio , mit dem .NET-Anwendungen mit minimalen AWS Kenntnissen und mit den wenigsten Klicks oder Befehlen bereitgestellt werden können. Weitere Informationen finden Sie unter [Bereitstellen von Anwendungen in AWS](#).

2020-08-24: Version 3.5 des SDK wurde veröffentlicht

- Die .NET-Erfahrung wurde standardisiert, indem die Unterstützung für alle Nicht-Framework-Varianten des SDK auf .NET Standard 2.0 umgestellt wurde. Weitere Informationen finden Sie unter [Migration auf Version 3.5](#).
- Paginatoren wurden zu vielen Service-Clients hinzugefügt, wodurch die Paginierung von API-Ergebnissen bequemer wird. Weitere Informationen finden Sie unter [Paginatoren](#).

Vom unterstützte Plattformen AWS SDK for .NET

Das AWS SDK for .NET stellt verschiedene Gruppen von Komponenten für Entwickler bereit, die auf unterschiedliche 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

unterstützt AWS SDK for .NET Anwendungen, die für .NET Core (.NET Core 3.1, .NET 5, .NET 6 usw.) geschrieben wurden. -AWSService-Clients unterstützen nur asynchrone Aufrufmuster im .NET Core. Dies betrifft auch viele der High-Level-Abstraktionen, 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 for .NET entsprechen [.NET Standard 2.0](#). Die AWS SDK for .NET bietet nur asynchrone Methoden für Anwendungen, die auf .NET Standard geschrieben wurden.

.NET Framework 4.5

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

Diese Version von AWS SDK for .NET wurde mit .NET Framework 4.5 kompiliert und läuft in der .NET 4.0-Laufzeit. -AWSServiceclients unterstützen synchrone und asynchrone Aufrufmuster und verwenden die in [C# 5.0](#) eingeführten Schlüsselwörter [asynchron](#) und [await](#).

.NET Framework 3.5

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

Diese Version von AWS SDK for .NET wurde mit .NET Framework 3.5 kompiliert und läuft entweder in der Laufzeit .NET 2.0 oder .NET 4.0. -AWSServiceclients unterstützen synchrone und asynchrone Aufrufmuster und verwenden das ältere Start- und Endmuster.

Note

Das AWS SDK for .NET ist nicht FIPS-konform (Federal Information Processing Standard), wenn es von Anwendungen verwendet wird, die für Version 2.0 der CLR erstellt wurden. Einzelheiten dazu, wie Sie eine FIPS-konforme Implementierung in dieser Umgebung ersetzen können, finden Sie unter [CryptoConfig](#) im Microsoft-Blog und in der HMACSHA256-Klasse des [CLR-Sicherheitsteams](#) (HMACSHA256Cng) in Security.Cryptography.dll.

Mobile Class Library und Xamarin

Das AWS SDK for .NET enthält auch eine Portable Class Library-Implementierung. Die Implementierung der portablen Klassenbibliothek kann auf mehrere Plattformen abzielen, darunter Universal Windows Platform (UWP) und Xamarin auf iOS und Android. Weitere Informationen finden Sie unter [Mobile SDK for .NET and Xamarin](#). -AWSServiceclients unterstützen nur asynchrone Anrufmuster.

Unity-Unterstützung

Weitere Informationen zur Unity-Unterstützung finden Sie unter [Besondere Überlegungen zur Unity-Unterstützung](#).

Weitere Informationen

[Migration auf -Version 3.5AWS SDK for .NET](#)

Migrieren auf AWS SDK for .NET-Version 3

In diesem Thema werden Änderungen in Version 3 des AWS SDK for .NET beschrieben und wie Sie Ihren Code auf diese Version des SDKs migrieren.

Über die Versionen des AWS SDK for .NET

Das AWS SDK for .NET, das ursprünglich im November 2009 veröffentlicht wurde, wurde für .NET Framework 2.0 entwickelt. Seit dieser Veröffentlichung wurde .NET mit dem .NET Framework 4.0

und .NET Framework 4.5 verbessert und um neue Zielplattformen erweitert: WinRT und Windows Phone.

AWS SDK for .NET Version 2 wurde aktualisiert, um die neuen Funktionen der .NET-Plattform zu nutzen und auf WinRT und Windows Phone abzielen.

AWS SDK for .NET Version 3 wurde aktualisiert, um die Komponenten modular zu gestalten.

Neugestaltung der Architektur für das SDK

Die gesamte Version 3 des AWS SDK for .NET wurde modular überarbeitet. Jeder Service ist jetzt in einer eigenen Komponente anstatt in einer globalen Komponente implementiert. Sie müssen Ihrer Anwendung nicht mehr das gesamte AWS SDK for .NET hinzufügen. Sie können jetzt Baugruppen nur für die AWS-Dienste, die Ihre Anwendung verwendet.

Abwärtskompatible Änderungen

In den folgenden Abschnitten werden Änderungen an Version 3 des AWS SDK for .NET beschrieben.

AWSClientFactory entfernt

Die `Amazon.AWSClientFactory`-Klasse wurde entfernt. Jetzt müssen Sie zum Erstellen eines Service-Clients den Konstruktor des Service-Clients verwenden. So erstellen Sie zum Beispiel einen `AmazonEC2Client`:

```
var ec2Client = new Amazon.EC2.AmazonEC2Client();
```

Amazon.Runtime.AssumeRoleAWSCredentials entfernt

Die `Amazon.Runtime.AssumeRoleAWSCredentials`-Klasse wurde entfernt, weil sie sich in einem Core-Namespace befand, aber eine Abhängigkeit im AWS Security Token Service hatte, und weil sie im SDK seit einiger Zeit überflüssig war. Verwenden Sie stattdessen die `Amazon.SecurityToken.AssumeRoleAWSCredentials`-Klasse.

SetACL-Methode vom S3Link entfernt

Die `S3Link`-Klasse ist Teil des `Amazon.DynamoDBv2`-Paket und wird für das Speichern von Objekten in Amazon S3 verwendet, die Referenzen in einem DynamoDB -Artikel sind. Dies ist eine nützliche Funktion, aber wir wollten keine Kompilierungsabhängigkeit im `Amazon.S3`-Paket für DynamoDB. Aus diesem Grund haben wir die bereitgestellten `Amazon.S3`-Methoden aus der `S3Link`-Klasse

vereinfacht und die `SetACL`-Methode durch die `MakeS3ObjectPublic`-Methode ersetzt. Um mehr Kontrolle über die Zugriffskontrollliste (ACL) für das Objekt zu haben, verwenden Sie das `Amazon.S3`-Paket direkt.

Überflüssige Ergebnisklassen entfernt

Für die meisten Services im AWS SDK for .NET geben Operationen ein Antwortobjekt zurück, das Metadaten für die Operation enthält, z. B. die Anforderungs-ID und ein Ergebnisobjekt. Eine separate Antwort- und Ergebnisklasse zu haben, war redundant und erzeugte zusätzliche Schreibarbeit für Entwickler. In Version 2 des AWS SDK for .NET haben wir sämtliche Informationen der Ergebnisklasse in die Antwortklasse gestellt. Wir haben auch die Ergebnisklassen als überflüssig markiert, da wir von ihrer Verwendung abraten. In Version 3 des AWS SDK for .NET haben wir diese überflüssigen Ergebnisklassen entfernt, um die Größe des SDKs zu reduzieren.

AWSÄnderungen im Abschnitt Config

Eine erweiterte Konfiguration des AWS SDK for .NET ist über die `App.config`- oder `Web.config`-Datei möglich. Dies können Sie durch einen `<aws>`-Konfigurationsabschnitt wie den folgenden, der auf den SDK-Komponentennamen verweist, vornehmen.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

In Version 3 des AWS SDK for .NET existiert die `AWSSDK`-Komponente nicht mehr. Der allgemeine Code wurde in die `AWSSDK.Core`-Komponente gestellt. Daher müssen Sie die Verweise auf die `AWSSDK`-Komponente in der `App.config`- oder `Web.config`-Datei wie folgt auf die `AWSSDK.Core`-Komponente ändern.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```



```
</aws>  
</configuration>
```

Sie können die Konfigurationseinstellungen mithilfe der `Amazon.AWSConfigs`-Klasse auch bearbeiten. In Version 3 des AWS SDK for .NET haben wir die Konfigurationseinstellungen für DynamoDB aus der `Amazon.AWSConfigs`-Klasse in die `Amazon.AWSConfigsDynamoDB`-Klasse.

Migration auf -Version 3.5 AWS SDK for .NET

Version 3.5 des AWS SDK for .NET standardisiert die .NET-Erfahrung weiter, indem die Unterstützung für alle nicht dem Framework zugehörigen Varianten des SDK auf [.NET Standard 2.0](#) übertragen werden. Je nach Umgebung und Codebasis müssen Sie möglicherweise bestimmte Migrationsarbeiten ausführen, um die Funktionen von Version 3.5 zu nutzen.

In diesem Thema werden die Änderungen in Version 3.5 und mögliche Arbeiten beschrieben, die Sie eventuell durchführen müssen, wenn Sie Ihre Umgebung oder den Codes von Version 3 migrieren möchten.

Was hat sich für Version 3.5 geändert

Im Folgenden wird beschrieben, was sich in AWS SDK for .NET-Version 3.5 geändert hat und was nicht.

.NET Framework und .NET Core

Unterstützung für .NET Framework und .NET Core hat sich nicht geändert.

Xamarin

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](#).

Unity

Unity-Apps müssen auf .NET Standard 2.0- oder .NET 4.x-Profilen mit Unity 2018.1 oder höher ausgerichtet werden. Weitere Informationen finden Sie unter [.NET-Profilunterstützung](#). Zusätzlich, wenn Sie verwenden IL2CPP um zu bauen, müssen Sie das Code-Stripping deaktivieren, indem Sie ein `link.xml`-Datei, wie unter [Verweisen auf AWS SDK for .NET Standard 2.0 von Unity, Xamarin](#)

[oder UWP](#). Nachdem Sie Ihren Code auf eine der empfohlenen Codebasen portiert haben, kann Ihre Unity-App auf alle vom SDK angebotenen Services zugreifen.

Da Unity .NET Standard 2.0 unterstützt, AWS SDK Core-Paket der SDK-Version 3.5 enthält keinen Unity-spezifischen Code mehr. Dies schließt eine gewisse allgemeine Funktionalität ein. Um einen besseren Übergang zu ermöglichen, müssen alle ErbeUnity-Code steht als Referenz in [aw/aws-sdk-unity-net](#) GitHub -Repository. Wenn Sie bemerken, dass fehlende Funktionalität eine Auswirkung auf Ihre Verwendung von AWS mit Unity können Sie unter <https://github.com/aws/dotnet/issues>.

Lesen Sie auch [Besondere Überlegungen zur Unity-Unterstützung](#).

Universal Windows Platform (UWP)

Richten Sie Ihre UWP-Anwendung auf [Version 16299 oder höher](#) (Fall Creators Update, Version 1709, veröffentlicht im Oktober 2017) aus.

Windows Phone und Silverlight

Version 3.5 der AWS SDK for .NET unterstützt diese Plattformen nicht, da Microsoft sie nicht mehr aktiv entwickelt. Weitere Informationen finden Sie unter:

- [Ende der Unterstützung für Windows 10 Mobile](#)
- [Ende der Unterstützung für Silverlight](#)

Ältere tragbare Klassenbibliotheken (profilbasierte PCL)

Ziehen Sie die erneute Ausrichtung Ihrer Bibliothek auf .NET Standard in Erwägung. Weitere Informationen finden Sie unter [Vergleich mit Portable Class Libraries](#) von Microsoft.

Amazon Cognito Sync Manager und Amazon Mobile Analytics Manager

Allgemeine Abstraktionen, die die Verwendung von Amazon Cognito Sync und Amazon Mobile Analytics erleichtern, werden aus -Version 3.5 AWS SDK for .NET AWS AppSync ist der bevorzugte Ersatz für Amazon Cognito Sync. Amazon Pinpoint ist der bevorzugte Ersatz für Amazon Mobile Analytics.

Wenn Ihr Code vom Fehlen von allgemeinem Bibliothekscode für betroffen ist AWS AppSync und Amazon Pinpoint haben, können Sie Ihr Interesse an einer oder beiden der folgenden Aktionen angeben GitHub probleme: <https://github.com/aws/dotnet/issues/20> und <https://github.com/aws/dotnet/issues/19>. Sie können die Bibliotheken für Amazon Cognito Sync Manager und Amazon Mobile

Analytics Manager auch von den folgenden beziehen GitHub Repositorys: [aw/amazon-cognito-sync-manager-net](https://github.com/aws/amazon-cognito-sync-manager-net) und [aw/aws-mobile-analytics-manager-net](https://github.com/aws/aws-mobile-analytics-manager-net).

Migrieren von synchronem Code

Version 3.5 des AWS SDK for .NET unterstützt sowohl .NET Framework als auch .NET Standard (über .NET Core-Versionen wie .NET Core 3.1, .NET 5 usw.). Varianten des SDK, die dem .NET-Standard entsprechen, bieten nur asynchrone Methoden. Wenn Sie also .NET Standard nutzen möchten, müssen Sie den synchronen Code so ändern, dass er asynchron ausgeführt wird.

Die folgenden Codeausschnitte zeigen, wie Sie synchronen Code in asynchronen Code ändern können. Der Code in diesen Ausschnitten wird verwendet, um die Anzahl der Amazon S3 S3-Buckets anzuzeigen.

Der ursprüngliche Code ruft [ListBuckets](#).

```
private static ListBucketsResponse MyListBuckets()
{
    var s3Client = new AmazonS3Client();
    var response = s3Client.ListBuckets();
    return response;
}

// From the calling function
ListBucketsResponse response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
```

Zur Verwendung von Version 3.5 des SDK rufen Sie [ListBucketsAsync](#) stattdessen.

```
private static async Task<ListBucketsResponse> MyListBuckets()
{
    var s3Client = new AmazonS3Client();
    var response = await s3Client.ListBucketsAsync();
    return response;
}

// From an asynchronous calling function
ListBucketsResponse response = await MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");

// OR From a synchronous calling function
```

```
Task<ListBucketsResponse> response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
```

Migration auf Version 3.7 AWS SDK for .NET

Ab Version 3.7 AWS SDK for .NET .NET-Standard 1.3 nicht mehr unterstützt.

Weitere Informationen zur Migration von .NET-Standard 1.3 finden Sie unter [Migration von .NET-Standard 1.3](#) aus.

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 Beendigung der Unterstützung für .NET-Standard 1.3 auf AWS SDK for .NET am 31. Dezember 2020.

AWS stellte weiterhin Serviceupdates und Sicherheitskorrekturen auf AWS SDK for .NET. Das gilt für .NET Standard 1.3 bis zum 1. Oktober 2020. Danach wurde das .NET-Standard 1.3-Ziel in den Wartungsmodus versetzt, was bedeutete, dass keine neuen Updates veröffentlicht wurden. AWSes wurden nur kritische Bugfixes und Sicherheitspatches angewendet.

Unterstützung für .NET-Standard 1.3 auf AWS SDK for .NET kam zu seinem Lebensende. Danach wurden keine Fehlerbehebungen oder Sicherheitspatches mehr angewendet. Artefakte, die mit diesem Ziel erstellt wurden, bleiben auf NuGet zum Download verfügbar.

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 for .NET zu verwenden.

Bei Fragen oder Bedenken [kontaktieren Sie den AWS-Support](#).

Arbeiten mit - AWS Services in der AWS SDK for .NET

Die folgenden Abschnitte enthalten Beispiele, Tutorials, Aufgaben und Leitfäden, die Ihnen zeigen, wie Sie die AWS SDK for .NET für die Arbeit mit - AWS Services verwenden. Diese Beispiele und Tutorials basieren auf einer API, die von AWS SDK for .NET bereitgestellt wird. Informationen dazu, welche Klassen und Methoden in der API verfügbar sind, finden Sie in der API [AWS SDK for .NET - Referenz zu](#) .

Wenn Sie noch nicht mit dem vertraut sind AWS SDK for .NET, sollten Sie sich zuerst das [Machen Sie einen kurzen Rundgang](#) Thema ansehen. Es bietet Ihnen eine Einführung in das SDK.

Weitere Codebeispiele finden Sie im [AWS Code Examples Repository](#) und im [awslabs-Repository](#) auf GitHub.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

- [Codebeispiele mit Anleitungen für die AWS SDK for .NET](#)
- [AWS LambdaFür den Rechendienst verwenden](#)
- [Allgemeine Bibliotheken und Frameworks für die AWS SDK for .NET](#)
- [Programmierung AWS OpsWorks für die Arbeit mit Stacks und Anwendungen](#)
- [Support für SonstigeAWS-Services und Konfiguration](#)

Codebeispiele mit Anleitungen für die AWS SDK for .NET

Die folgenden Abschnitte enthalten Codebeispiele und bieten Anleitungen für die Beispiele. Sie können Ihnen helfen zu lernen, wie Sie die verwenden, AWS SDK for .NET um mit - AWS Services zu arbeiten.

Wenn Sie noch nicht mit dem vertraut sind AWS SDK for .NET, sollten Sie sich zuerst das [Machen Sie einen kurzen Rundgang](#) Thema ansehen. Es bietet Ihnen eine Einführung in das SDK.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

- [Zugreifen auf AWS CloudFormation mit der AWS SDK for .NET](#)
- [Authentifizieren von Benutzern mit Amazon Cognito](#)
- [Verwenden von Amazon-DynamoDB-NoSQL-Datenbanken](#)
- [Arbeiten mit Amazon EC2](#)
- [Zugreifen auf AWS Identity and Access Management \(IAM\) mit der AWS SDK for .NET](#)
- [Verwenden des Internetspeichers von Amazon Simple Storage Service](#)
- [Senden von Benachrichtigungen aus der Cloud mit Amazon Simple Notification Service](#)
- [Messaging mit Amazon SQS](#)

Zugreifen auf AWS CloudFormation mit der AWS SDK for .NET

Die AWS SDK for .NET unterstützt [AWS CloudFormation](#), wodurch - AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt erstellt und bereitgestellt werden.

APIs

stellt APIs für AWS CloudFormation Clients AWS SDK for .NET bereit. Mit den APIs können Sie mit AWS CloudFormation Funktionen wie Vorlagen und Stacks arbeiten. Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen die Muster zeigen, die Sie bei der Arbeit mit diesen APIs befolgen können. Um den vollständigen Satz von APIs anzuzeigen, sehen Sie sich die [AWS SDK for .NET API-Referenz](#) an (und scrollen Sie zu „AmazonCloudFormation“).

Die AWS CloudFormation APIs werden vom [AWSSDK-CloudFormation](#) Paket bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

Themen

- [Auflisten von AWS Ressourcen mit AWS CloudFormation](#)

Auflisten von AWS Ressourcen mit AWS CloudFormation

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um die Ressourcen in AWS CloudFormation Stacks aufzulisten. Das Beispiel verwendet die Low-Level-API. Die Anwendung verwendet keine Argumente, sondern sammelt einfach 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](#)

Programmirelemente:

- Namespace [Amazon.CloudFormation](#)

Klasse [AmazonCloudFormationClient](#)

- Namespace [AmazonCloudFormation.Modell](#)

Klasse [I CloudFormationPaginatorFactory.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.SSO0IDC 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;
    }
}
```

Authentifizieren von Benutzern mit Amazon Cognito

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

Mit Amazon Cognito Identity können Sie eindeutige Identitäten für Ihre Benutzer erstellen und sie für einen sicheren Zugriff auf Ihre - AWS Ressourcen wie Amazon S3 oder Amazon DynamoDB authentifizieren. Amazon Cognito Identity unterstützt öffentliche Identitätsanbieter wie Amazon, Facebook, Bol/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 Amazon Cognito Sync weiterhin verwenden, um Benutzerdaten zu synchronisieren und auf Ressourcen zuzugreifen AWS .

Weitere Informationen zu [Amazon Cognito](#) finden Sie im [Amazon Cognito-Entwicklerhandbuch](#).

Die folgenden Codebeispiele zeigen, wie Sie Amazon Cognito Identity einfach verwenden können. Das [Anmeldeinformationsanbieter](#) Beispiel zeigt, wie Sie Benutzeridentitäten erstellen und authentifizieren. Das [CognitoAuthentication -Erweiterungsbibliothek](#) Beispiel zeigt, wie Sie die CognitoAuthentication Erweiterungsbibliothek verwenden, um Amazon Cognito-Benutzerpools zu authentifizieren.

Themen

- [Anbieter von Amazon Cognito-Anmeldeinformationen](#)
- [Beispiele für Amazon CognitoAuthentication -Erweiterungsbibliotheken](#)

Anbieter von Amazon Cognito-Anmeldeinformationen

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, das sich im [AWSSDK.CognitoIdentity](#) NuGet-Paket befindet, ist ein Anmeldeinformationsobjekt, das Amazon Cognito und die AWS Security Token Service (AWS STS) verwendet, um Anmeldeinformationen für AWS Aufrufe 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 einen neuen Identitäten-Pool über die Amazon Cognito-Konsole 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.

Einrichten von `CognitoAWSCredentials`

Das folgende Codebeispiel zeigt, wie Sie `einrichtenCognitoAWSCredentials`, mit dem Sie dann einen Aufruf an Amazon S3 als nicht authentifizierter Benutzer tätigen können. 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();  
}
```

Als nicht AWS authentifizierten Benutzer verwenden

Das folgende Codebeispiel zeigt, wie Sie AWS als nicht authentifizierter Benutzer verwenden können, 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 for .NET ist. Wenn Sie sowohl als auch verwenden `AmazonCognitoSyncClient` `CognitoAWSCredentials`, müssen Sie die `IdentityId` Eigenschaften `IdentityPoolId` und nicht angeben, wenn Sie Aufrufe mit tätigen `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 für Amazon CognitoAuthentication -Erweiterungsbibliotheken

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

Die `CognitoAuthentication` Erweiterungsbibliothek im [Amazon.Extensions.CognitoAuthentication](#) NuGet package vereinfacht den Authentifizierungsprozess von Amazon Cognito-Benutzerpools

für .NET-Core- und Xamarin-Entwickler. Die Bibliothek baut auf der API des Amazon Cognito-Identitätsanbieters auf, um API-Aufrufe zur Benutzerauthentifizierung zu erstellen und zu senden.

Verwenden der CognitoAuthentication Erweiterungsbibliothek

Amazon Cognito verfügt über einige integrierte - AuthFlow und -ChallengeNameWerte 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 der Basisauthentifizierung

Erstellen Sie ein [AmazonCognitoIdentityProviderClient](#) mit [Anonymous AWSCredentials](#), für das keine signierten Anforderungen 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;
}
}
```

Authentifizierung mit Herausforderungen

Die Fortsetzung des Authentifizierungsablaufs mit Herausforderungen, z. B. mit `NewPasswordRequired` und `Multi-Factor Authentication (MFA)` ist ebenfalls einfacher. Die einzigen Anforderungen sind die `CognitoAuthentication` Objekte, das Passwort des Benutzers für SRP und die erforderlichen Informationen für die nächste Aufforderung, die abgerufen werden, nachdem der Benutzer zur Eingabe aufgefordert wurde. Der folgende Code zeigt eine Möglichkeit, den Aufforderungstyp zu überprüfen und die entsprechenden Antworten auf MFA und `NewPasswordRequired` Aufforderungen während des Authentifizierungsablaufs 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 von - 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 einen Identitäten-Pool über die Amazon Cognito-Konsole für Verbundidentitäten erstellen. Indem Sie den Amazon Cognito-Benutzerpool angeben, den Sie als Anbieter erstellt haben, können Sie Ihren Amazon Cognito-Benutzerpool-Benutzern mithilfe seiner poolID und clientID den Zugriff AWS auf 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-Benutzerinformationen Aufrufe an verschiedene AWS Ressourcen tätigen. Das folgende Beispiel zeigt einen Benutzer, der mit SRP authentifiziert wurde und auf die verschiedenen Amazon S3-Buckets zugreift, die von der Rolle des zugehörigen Identitätspools zugelassen werden

```
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 – Initiiert mit einem Aufruf von `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP – Initiiert mit einem Aufruf von `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP – Initiiert mit einem Aufruf von `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 im .NET Framework als auch im .NET (Core) vorhanden, aber die Aufrufkonventionen unterscheiden sich, unabhängig davon, ob synchron oder asynchron.

unterstützt AWS SDK for .NET 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 Dokumentmodell und das Object-Persistence-Modell.

Die folgenden Informationen führen diese Modelle und ihre APIs ein, enthalten Beispiele für ihre Verwendung und wann sie verwendet werden sollen, und enthalten Links zu zusätzlichen DynamoDB-Programmierressourcen in der AWS SDK for .NET.

Themen

- [Low-Level-Modell](#)
- [Dokument-Modell](#)
- [Object Persistence-Modell](#)
- [Weitere Informationen](#)
- [Verwenden von Ausdrücken mit Amazon DynamoDB und der AWS SDK for .NET](#)
- [JSON-Unterstützung in Amazon DynamoDB](#)

Low-Level-Modell

Das Low-Level-Programmiermodell umschließt direkte Aufrufe an den DynamoDB-Service. Sie greifen auf dieses Modell über den [Amazon.DynamoDBv2](#)-Namespace zu.

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

Die folgenden Beispiele zeigen Ihnen, 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 Instanz der `CreateTableRequest`-Klasse, die Merkmale wie erforderliche Elementattributnamen, Primärschlüsseldefinition und Durchsatzkapazität enthält. Die `CreateTable`-Methode gibt eine Instanz 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 Sie mit dem Low-Level-Modell 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 Instanz der `PutItemRequest`-Klasse eingefügt. Jede der beiden Instanzen 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 Dokumentprogrammierungsmodell 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 den [Amazon.DynamoDBv2.DocumentModel](#)-Namespace auf dieses Modell zu.

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-Dokumentenprogrammierungsmodell finden Sie unter [.NET: Dokumentmodell](#) im [Amazon-DynamoDB-Entwicklerhandbuch](#).

In den folgenden Abschnitten finden Sie 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 einer Darstellung der Tabelle

Um Datenoperationen mit dem Dokumentmodell durchzuführen, müssen Sie zunächst eine Instance der `Table` Klasse erstellen, die eine bestimmte Tabelle darstellt. Es gibt zwei Hauptmethoden, dies zu tun.

LoadTable -Methode

Der erste Mechanismus besteht darin, eine der statischen LoadTable Methoden der [Table](#) Klasse zu verwenden, ähnlich wie im folgenden Beispiel:

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

Note

Dieser Mechanismus funktioniert zwar unter bestimmten Bedingungen, kann jedoch manchmal zu zusätzlicher Latenz oder Deadlocks aufgrund von Kaltstart- und Threadpool-Verhalten führen. Weitere Informationen zu diesen Verhaltensweisen finden Sie im Blogbeitrag [Verbesserte DynamoDB-Initialisierungsmuster für die AWS SDK for .NET](#).

TableBuilder

Ein alternativer Mechanismus, die [TableBuilder](#) Klasse, wurde in [Version 3.7.203 des AWSSDK.DynamoDBv2 NuGet package](#) eingeführt. Dieser Mechanismus kann die oben genannten Verhaltensweisen beheben, indem bestimmte implizite Methodenaufrufe entfernt werden, insbesondere die `-DescribeTable` Methode. Dieser Mechanismus wird auf ähnliche Weise wie das folgende 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 for .NET](#).

Einfügen eines Elements in eine Tabelle

Im folgenden Beispiel wird mithilfe der `-PutItemAsync` Methode der `-Table` Klasse eine Antwort in die Tabelle Antworten 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);
```

Abrufen eines Elements aus einer Tabelle

Im folgenden Beispiel wird eine Antwort über die `-GetItemAsync` Methode der `-Table` Klasse abgerufen. Um die abzurufende Antwort zu bestimmen, 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 Zeichenfolgen für die `WriteLine` Methode konvertiert. Sie können explizite Konvertierungen durchführen, indem Sie die verschiedenen Methoden der `DynamoDBEntry` Klasse „As[type]“ verwenden. Sie können beispielsweise den

Wert für explizit Id von einem `Primitive` Datentyp in eine GUID konvertieren, indem Sie die `-AsGuid()` Methode verwenden:

```
var guid = reply["Id"].AsGuid();
```

Object Persistence-Modell

Das Object-Persistence-Programmiermodell wurde speziell für das Speichern, Laden und Abfragen von .NET-Objekten in DynamoDB entwickelt. Sie greifen über den [Amazon.DynamoDBv2.DataModel](#)-Namespace auf dieses Modell zu.

Von den drei Modellen ist das Object-Persistence-Modell am einfachsten zu codieren, 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-Objektpersistenz-Programmiermodell finden Sie unter [.NET: Object Persistence Model](#) im [Amazon DynamoDB-Entwicklerhandbuch](#).

Die folgenden Beispiele zeigen Ihnen, wie Sie eine .NET-Klasse definieren, die ein DynamoDB-Element darstellt, eine Instance der .NET-Klasse verwenden, um ein Element in eine DynamoDB-Tabelle einzufügen, und eine Instance 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 einer 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 für 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 Kontexts für das Object-Persistence-Modell

Um das Object-Persistence-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 auszuführen und Abfragen auszuführen.

Grundlegender Kontext

Das folgende Beispiel zeigt, wie Sie den einfachsten Kontext erstellen.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Kontext mit -DisableFetchingTableMetadata Eigenschaft

Das folgende Beispiel zeigt, wie Sie zusätzlich die -DisableFetchingTableMetadataEigenschaft der -DynamoDBContextConfigKlasse festlegen können, um implizite Aufrufe der -DescribeTableMethode 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 weglassen, die die Schlüssel- und Indexstruktur von Tabellenelementen aus der `Reply` Klasse beschreiben. Diese Attribute werden stattdessen durch einen impliziten Aufruf der `DescribeTable` Methode abgeleitet. Wenn auf `DisableFetchingTableMetadata` `true` ist, wie im zweiten Beispiel gezeigt, QueryAsync basieren Methoden des Object-Persistence-Modells wie `SaveAsync` und vollständig auf den in der `Reply` Klasse definierten Attributen. In diesem Fall erfolgt kein Aufruf der `DescribeTable` Methode .

Note

Unter bestimmten Bedingungen können Aufrufe der `-DescribeTable` Methode manchmal zu zusätzlicher Latenz oder Deadlocks aufgrund von Kaltstart- und Threadpool-Verhalten führen. Aus diesem Grund ist es manchmal von Vorteil, Aufrufe dieser Methode zu vermeiden. Weitere Informationen zu diesen Verhaltensweisen finden Sie im Blogbeitrag [Verbesserte DynamoDB-Initialisierungsmuster für die AWS SDK for .NET](#).

Verwenden einer Instance 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 Instance 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 Instance einer .NET-Klasse zum Abrufen von Elementen aus einer Tabelle

In diesem Beispiel wird eine Abfrage erstellt, um alle Datensätze von „Author1“ mithilfe der `-QueryAsync` Methode der `-DynamoDBContext` Klasse zu finden. Anschließend werden Elemente über die `-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 Object-Persistence-Modell

Die oben gezeigten Beispiele und Erklärungen enthalten manchmal eine Eigenschaft der `DynamoDBContext` Klasse namens `DisableFetchingTableMetadata`. Mit dieser Eigenschaft, die in [Version 3.7.203 des AWSSDK.DynamoDBv2 NuGet package eingeführt wurde](#), können Sie

bestimmte Bedingungen vermeiden, die aufgrund von Kaltstart- und Threadpool-Verhaltensweisen zu zusätzlicher Latenz oder Deadlocks führen können. Weitere Informationen finden Sie im Blogbeitrag [Verbesserte DynamoDB-Initialisierungsmuster für die AWS SDK for .NET](#).

Im Folgenden finden Sie einige zusätzliche Informationen zu dieser Eigenschaft.

- Diese Eigenschaft kann global in Ihrer `-app.config` oder `-web.config` Datei festgelegt werden, wenn Sie .NET Framework verwenden.
- Diese Eigenschaft kann global festgelegt werden, indem die [AWSConfigsDynamoDB](#) Klasse verwendet wird, 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 möglich, weiterhin die `-DisableFetchingTableMetadata` Eigenschaft zu nutzen. Verwenden Sie dazu zusätzlich zur `-DisableFetchingTableMetadata` Eigenschaft die `-TableBuilder` Klasse. Die `TableBuilder` Klasse wurde auch in [Version 3.7.203 des AWSSDK.DynamoDBv2 NuGet - 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

Verwenden der AWS SDK for .NET zum Programmieren von DynamoDB-Informationen und -Beispielen**

- [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 der AWS SDK for .NET](#)
- [JSON-Unterstützung in Amazon DynamoDB](#)

Low-Level-Modell – Informationen und Beispiele

- [Arbeiten mit Tabellen mithilfe der AWS SDK for .NET Low-Level-API](#)
- [Arbeiten mit Elementen unter Verwendung der AWS SDK for .NET Low-Level-API](#)
- [Abfragen von Tabellen mit der AWS SDK for .NET Low-Level-API](#)
- [Scannen von Tabellen mit der AWS SDK for .NET Low-Level-API](#)
- [Arbeiten mit lokalen sekundären Indizes unter Verwendung der AWS SDK for .NET Low-Level-API](#)
- [Arbeiten mit globalen sekundären Indizes unter Verwendung der AWS SDK for .NET Low-Level-API](#)

Dokument-Modell – Informationen und Beispiele

- [DynamoDB-Datentypen](#)
- [DynamoDB-Eintrag](#)
- [.NET: Dokumentmodell](#)

Informationen und Beispiele zu Object-Persistence-Modellen

- [.NET: "Object Persistence"-Modell](#)

Verwenden von Ausdrücken mit Amazon DynamoDB und der AWS SDK for .NET

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

Die folgenden Codebeispiele veranschaulichen, wie Sie mit DynamoDB mit Ausdrücken AWS SDK for .NET programmieren. 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 zwei Beispielementen 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 [Fallbeispiel: Ein ProductCatalog Element](#). 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.WriteLine("Binary data");
}
// Binary set attribute value.
else if (value.BS.Count > 0)
{
    foreach (var bValue in value.BS)
    {
        Console.WriteLine("\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.WriteLine("\n");
    PrintItem(value.M);
}
// Number attribute value.
else if (value.N != null)
{
    Console.WriteLine(value.N);
}
// Number set attribute value.
else if (value.NS.Count > 0)
{
    Console.WriteLine("{0}", string.Join("\n", value.NS.ToArray()));
}
// Null attribute value.
else if (value.NULL)
{
    Console.WriteLine("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 zum Drucken des Attributs zu ermitteln. 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, MNS, und SS, wenn größer als Null Count ist, dann ist das Attribut vom 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 Beispiелеlement 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 Codebeispiele finden Sie unter

- [DynamoDB Series - Expressions](#)
- [Angeben von Elementattributen](#)
- [Ausdrucksattributnamen](#)
- [Bedingungsausdrücke](#)
- [Aktualisierungsausdrücke](#)
- [Arbeiten mit Elementen unter Verwendung der AWS SDK for .NET Low-Level-API](#)
- [Abfragen von Tabellen mit der AWS SDK for .NET Low-Level-API](#)
- [Scannen von Tabellen mit der AWS SDK for .NET Low-Level-API](#)
- [Arbeiten mit lokalen sekundären Indizes unter Verwendung der AWS SDK for .NET Low-Level-API](#)
- [Arbeiten mit globalen sekundären Indizes unter Verwendung der AWS SDK for .NET Low-Level-API](#)

JSON-Unterstützung in Amazon DynamoDB

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

Die AWS SDK for .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 diese einfügen.

Themen

- [Abrufen von Daten aus einer DynamoDB-Tabelle im JSON-Format](#)
- [Einfügen von JSON-Formatdaten in eine DynamoDB-Tabelle](#)
- [DynamoDB-Datentypkonvertierungen zu JSON](#)
- [Weitere Infos](#)

Abrufen von Daten aus einer DynamoDB-Tabelle im JSON-Format

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 bestimmen, aus der das Element abgerufen werden soll, verwendet

die `-LoadTable` Methode der `-Table` Klasse eine Instance der `-AmazonDynamoDBClient` Klasse und den Namen der Zieltabelle in DynamoDB .

Einfügen von JSON-Formatdaten in eine DynamoDB-Tabelle

Das folgende Beispiel zeigt, wie Sie das JSON-Format verwenden, 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, die eine Instance der `-AmazonDynamoDBClient` Klasse und den Namen der Zieltabelle in DynamoDB angibt.

DynamoDB-Datentypkonvertierungen zu JSON

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 Instance einer `Document` Klasse zu konvertieren, werden einige DynamoDB-Datentypen nicht wie erwartet konvertiert. Das heißt:

- DynamoDB-Sets (die BS Typen NS, und SS) werden in JSON-Arrays konvertiert.
- DynamoDB-Binärskalare und -Sets (die BS Typen B und) werden in Base64-kodierte JSON-Zeichenfolgen oder Listen von Zeichenfolgen 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 das Programmieren von JSON mit DynamoDB mit der finden Sie AWS SDK for .NET unter:

- [JSON-Unterstützung in DynamoDB](#)
- [Amazon DynamoDB-Aktualisierungen - JSON, Erweitertes kostenloses Kontingent, Flexible Skalierung, Größere Elemente](#)

Arbeiten mit Amazon EC2

AWS SDK for .NET unterstützt [Amazon EC2](#), einen Webservice, der anpassbare Rechenkapazität bereitstellt. Sie verwenden diese Rechenkapazität, um Ihre Softwaresysteme zu erstellen und zu hosten.

APIs

stellt APIs für Amazon EC2-Clients AWS SDK for .NET bereit. Mit den APIs können Sie mit EC2-Funktionen wie Sicherheitsgruppen und Schlüsselpaaren arbeiten. Mit den APIs können Sie auch Amazon EC2 steuern. Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen die Muster zeigen, die Sie bei der Arbeit mit diesen APIs befolgen können. Um den vollständigen Satz von APIs anzuzeigen, lesen Sie die [AWS SDK for .NET API-Referenz](#) (und scrollen Sie zu „Amazon.EC2“).

Die Amazon EC2-APIs werden vom [AWSSDK.EC2](#) NuGet package bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Informationen zu den Beispielen

Die Beispiele in diesem Abschnitt zeigen Ihnen, wie Sie mit Amazon EC2Clients arbeiten und Amazon EC2 verwalten.

Das [EC2-Spot-Instance-Tutorial](#) zeigt Ihnen, wie Sie Amazon EC2-Spot-Instances anfordern. Spot-Instances ermöglichen Ihnen den Zugriff auf ungenutzte EC2-Kapazität für weniger als den On-Demand-Preis.

Themen

- [Arbeiten mit Sicherheitsgruppen in Amazon EC2](#)
- [Arbeiten mit Amazon EC2-Schlüsselpaaren](#)
- [Anzeigen Ihrer Amazon EC2-Regionen und Availability Zones](#)
- [Arbeiten mit Amazon EC2](#)
- [Amazon EC2 Spot Instance-Tutorial](#)

Arbeiten mit Sicherheitsgruppen in Amazon EC2

In Amazon EC2 fungiert eine Sicherheitsgruppe als virtuelle Firewall, die den Netzwerkverkehr für eine oder mehrere EC2-Instances steuert. Standardmäßig ordnet EC2 Ihre Instances einer Sicherheitsgruppe zu, die keinen eingehenden Datenverkehr zulässt. Sie können eine Sicherheitsgruppe erstellen, die es den EC2-Instances ermöglicht, eingehenden Datenverkehr zu akzeptieren. Wenn Sie beispielsweise eine Verbindung zu einer EC2 Windows-Instance herstellen müssen, muss die Sicherheitsgruppe so konfiguriert werden, dass RDP-Datenverkehr möglich ist.

Weitere Informationen zu Sicherheitsgruppen finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) und im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Wenn Sie die verwenden AWS SDK for .NET, können Sie eine Sicherheitsgruppe für die Verwendung in EC2 in einer VPC oder EC2-Classic erstellen. Weitere Informationen zu EC2 in einer VPC im Vergleich zu EC2-Classic finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Warning

EC2-Classic wird am 15. August 2022 eingestellt. Wir empfehlen Ihnen die Migration von EC2-Classic zu einer VPC. Weitere Informationen finden Sie unter Migration von EC2-Classic zu einer VPC im [Amazon-EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon-](#)

[EC2-Benutzerhandbuch für Windows-Instances](#). Lesen Sie außerdem den Blogbeitrag [EC2-Classic Networking is Retiring – Here's How to Prepare](#) (EC2-Classic Networking wird außer Betrieb genommen – so bereiten Sie sich vor).

Informationen zu den APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt ([Arbeiten mit Amazon EC2](#)).

Themen

- [Aufzählen von Sicherheitsgruppen](#)
- [Erstellen von Sicherheitsgruppen](#)
- [Aktualisieren von Sicherheitsgruppen](#)

Aufzählen von Sicherheitsgruppen

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um Sicherheitsgruppen aufzuzählen. Wenn Sie eine [Amazon Virtual Private Cloud](#)-ID angeben, zählt die Anwendung die Sicherheitsgruppen für diese bestimmte VPC. Andernfalls zeigt die Anwendung einfach eine Liste aller verfügbaren Sicherheitsgruppen an.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Aufzählen von Sicherheitsgruppen](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Aufzählen von Sicherheitsgruppen

Der folgende Codeausschnitt zählt Ihre Sicherheitsgruppen. Es werden alle Gruppen oder die Gruppen für eine bestimmte VPC aufgezählt, falls eine angegeben ist.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

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 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 erstellt wird, dass der Name Teil des Name-Wert-Paares auf „vpc-id“ gesetzt ist. Dieser Name stammt aus der Beschreibung für die `-Filters`Eigenschaft der [DescribeSecurityGroupsRequest](#)-Klasse.
- Um die vollständige Liste Ihrer Sicherheitsgruppen zu erhalten, können Sie auch [DescribeSecurityGroupsAsync ohne Parameter](#) verwenden.
- Sie können die Ergebnisse überprüfen, indem Sie die Liste der Sicherheitsgruppen in der [Amazon EC2-Konsole](#) überprüfen.

Erstellen von Sicherheitsgruppen

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um eine Sicherheitsgruppe zu erstellen. Sie können die ID einer vorhandenen VPC angeben, um eine Sicherheitsgruppe für EC2 in einer VPC zu erstellen. Wenn Sie keine solche ID angeben, wird die neue Sicherheitsgruppe für EC2-Classic verwendet, wenn 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. Weitere Informationen finden Sie in den Referenzen für EC2 in einer VPC im Vergleich zu EC2-Classic im übergeordneten Abschnitt ([Arbeiten mit Sicherheitsgruppen in Amazon EC2](#)).

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Suchen vorhandener Sicherheitsgruppen](#)

- [Eine Sicherheitsgruppe erstellen](#)
- [Vollständiger Code](#)

Suchen vorhandener Sicherheitsgruppen

Der folgende Codeausschnitt sucht nach vorhandenen Sicherheitsgruppen mit dem angegebenen Namen in der angegebenen VPC.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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

Der folgende Codeausschnitt erstellt eine neue Sicherheitsgruppe, wenn in der angegebenen VPC keine Gruppe mit diesem Namen vorhanden ist. Wenn keine VPC angegeben ist und eine oder mehrere Gruppen mit diesem Namen vorhanden sind, gibt der Ausschnitt einfach die Liste der Gruppen zurück.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
```

```
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

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
{
    // = = = = =
    // = = =
    // Class to create a security group
    class Program
    {
        private const int MaxArgs = 2;
```

```
static async Task Main(string[] args)
{
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
        PrintHelp();
        return;
    }
    if(parsedArgs.Count > MaxArgs)
        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 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($"Not 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($"GroupName: {group.GroupName}");
        Console.WriteLine($"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(
        $"One or more security groups with name {groupName} already exist.\n");
    return securityGroups;
}

// If the security group doesn't already exist, 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.");
}
}

// = = = = =
// 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);
}
}
}
```

Aktualisieren von Sicherheitsgruppen

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe der eine Regel zu einer Sicherheitsgruppe AWS SDK for .NET hinzufügen. Insbesondere fügt das Beispiel eine Regel hinzu, die eingehenden Datenverkehr auf einem bestimmten TCP-Port zulässt, der beispielsweise für Remote-Verbindungen zu einer EC2-Instance 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 wird der angegebenen Sicherheitsgruppe eine Regel für eingehenden Datenverkehr hinzugefügt.

Note

Um dieses Beispiel verwenden zu können, benötigen Sie eine IP-Adresse (oder einen Adressbereich) im CIDR-Format. Weitere Informationen zu Methoden zum Abrufen der IP-Adresse Ihres lokalen Computers finden Sie unter [Zusätzliche Überlegungen](#) am Ende dieses Themas.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Hinzufügen einer Regel für eingehenden Datenverkehr](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Hinzufügen einer Regel für eingehenden Datenverkehr

Der folgende Codeausschnitt 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 diesen verwendeten Ausschnitt.

```
//
// 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($"New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}
```

Vollständiger Code

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

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

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
{
    // = = = = =
    // Class to add a rule that allows inbound traffic on TCP a port
    class Program
    {
        private const int DefaultPort = 3389;
    }
}
```

```
static async Task Main(string[] args)
{
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
        PrintHelp();
        return;
    }

    // Get the application arguments from the parsed list
    var groupID = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
    var ipAddress = CommandLine.GetArgument(parsedArgs, null, "-i", "--ip-address");
    var portStr = CommandLine.GetArgument(parsedArgs, DefaultPort.ToString(), "-p",
"--port");
    if(string.IsNullOrEmpty(ipAddress))
        CommandLine.ErrorExit("\nYou must supply an IP address in CIDR format.");
    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($"The 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 e2Client, 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.");
}
}

// = = = = =
// 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 keine Portnummer angeben, verwendet die Anwendung standardmäßig Port 3389. Dies ist der Port für Windows RDP, mit dem Sie eine Verbindung zu einer EC2-Instance herstellen können, auf der Windows ausgeführt wird. Wenn Sie eine EC2-Instance mit Linux starten, 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 für die `-IpProtocol`Eigenschaft der [-IpPermission](#)Klasse.
- Wenn Sie dieses Beispiel verwenden, möchten Sie möglicherweise die IP-Adresse Ihres lokalen Computers. Im Folgenden finden Sie einige Möglichkeiten, 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 diese Adresse mit einem Service abrufen. Ein solcher Service ist <http://checkip.amazonaws.com/>. Weitere Informationen zum Autorisieren von eingehendem Datenverkehr finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).
 - Eine andere Möglichkeit, die IP-Adresse Ihres lokalen Computers abzurufen, ist die Verwendung der [Amazon EC2-Konsole](#) .

Wählen Sie eine Ihrer Sicherheitsgruppen aus, wählen Sie die Registerkarte Regeln für eingehenden Datenverkehr und wählen Sie Regeln für eingehenden Datenverkehr bearbeiten aus. Öffnen Sie in einer Regel für eingehenden Datenverkehr das Dropdown-Menü in der Spalte

Quelle und wählen Sie Meine IP aus, um die IP-Adresse Ihres lokalen Computers im CIDR-Format anzuzeigen. Stellen Sie sicher, dass Sie den Vorgang abbrechen.

- Sie können die Ergebnisse dieses Beispiels überprüfen, indem Sie sich die Liste der Sicherheitsgruppen in der [Amazon EC2-Konsole](#) ansehen.

Arbeiten mit Amazon EC2-Schlüsselpaaren

Amazon EC2 verwendet Kryptografie für öffentliche Schlüssel, um Anmeldeinformationen zu ver- und entschlüsseln. Die Kryptografie mit öffentlichem Schlüssel verwendet einen öffentlichen Schlüssel, um Daten zu verschlüsseln, und dann verwendet der Empfänger 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 Start ein Schlüsselpaar angeben und dann den privaten Schlüssel des Paares angeben, wenn Sie eine Verbindung zu ihm herstellen.

Wenn Sie eine EC2-Instance starten, können Sie ein Schlüsselpaar dafür erstellen oder ein Schlüsselpaar verwenden, das Sie bereits beim Starten anderer Instances verwendet haben. Weitere Informationen zu Amazon EC2-Schlüsselpaaren finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Informationen zu den APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt ([Arbeiten mit Amazon EC2](#)).

Themen

- [Erstellen und Anzeigen von Schlüsselpaaren](#)
- [Löschen von Schlüsselpaaren](#)

Erstellen und Anzeigen von Schlüsselpaaren

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um ein Schlüsselpaar zu erstellen. Die Anwendung übernimmt den Namen für das neue Schlüsselpaar und den Namen einer PEM-Datei (mit der Erweiterung „.pem“). Es 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 Ausschnitte 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

Der folgende Ausschnitt erstellt ein Schlüsselpaar und speichert dann den privaten Schlüssel in der angegebenen PEM-Datei.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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

Der folgende Codeausschnitt zeigt eine Liste der verfügbaren Schlüsselpaare an.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}
```

Vollständiger Code

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

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmierelemente:

- Namespace [Amazon.EC2](#)
 - Klasse [AmazonEC2Client](#)
- Namespace [Amazon.EC2.Model](#)
 - Klasse [CreateKeyPairRequest](#)
 - Klasse [CreateKeyPairResponse](#)
 - Klasse [DescribeKeyPairsResponse](#)
 - Klasse [KeyValuePair](#)

Der Code

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;
```

```
namespace EC2CreateKeyPair
{
    // = = = = =
    // Class to create and store a key pair
    class Program
    {
        static async Task Main(string[] args)
        {
            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            // 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($"Created 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 (KeyValuePair 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.");
}
}
```

```
}

// = = = = =
// = = =
// 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

- Nachdem Sie das Beispiel ausgeführt haben, können Sie das neue Schlüsselpaar in der [Amazon EC2-Konsole](#) sehen.

- Wenn Sie ein Schlüsselpaar erstellen, müssen Sie den zurückgegebenen privaten Schlüssel speichern, da Sie den privaten Schlüssel später nicht abrufen können.

Löschen von Schlüsselpaaren

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um ein Schlüsselpaar zu löschen. Die Anwendung übernimmt den Namen eines Schlüsselpaars. Es löscht das Schlüsselpaar und zeigt dann alle verfügbaren Schlüsselpaare an. Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach alle verfügbaren Schlüsselpaare an.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Löschen des Schlüsselpaars](#)
- [Verfügbare Schlüsselpaare anzeigen](#)
- [Vollständiger Code](#)

Löschen des Schlüsselpaars

Der folgende Codeausschnitt löscht ein Schlüsselpaar.

Das Beispiel [am Ende dieses Themas](#) zeigt dieses verwendete Snippet.

```
//  
// Method to delete a key pair  
private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)  
{  
    await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{  
        KeyName = keyName});  
    Console.WriteLine($"{keyName} has been deleted (if it existed).");  
}
```

Verfügbare Schlüsselpaare anzeigen

Der folgende Codeausschnitt zeigt eine Liste der verfügbaren Schlüsselpaare an.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
```

```
// 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 (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}
```

Vollständiger Code

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

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmirelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

Klasse [DeleteKeyPairRequest](#)

Klasse [DescribeKeyPairsResponse](#)

Klasse [KeyValuePair](#)

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($"Key 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 (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
    }
}
```

Anzeigen Ihrer Amazon EC2-Regionen und Availability Zones

Amazon EC2 wird an mehreren Standorten weltweit gehostet. Diese Standorte bestehen aus -Regionen und Availability Zones. Jede -Region ist ein separater geografischer Bereich mit mehreren isolierten Standorten, die als Availability Zones bezeichnet werden.

Weitere Informationen zu Regionen und Availability Zones finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um Details zu den Regionen und Availability Zones abzurufen, die sich auf einen EC2-Client beziehen. Die Anwendung zeigt Listen der Regionen und Availability Zones an, die einem EC2-Client zur Verfügung stehen.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

Klasse [DescribeAvailabilityZonesResponse](#)

Klasse [DescribeRegionsResponse](#)

Klasse [AvailabilityZone](#)

[Klassenregion](#)

```
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);
        }
    }
}
```

```
}  
}  
}
```

Arbeiten mit Amazon EC2

Sie können die verwenden AWS SDK for .NET , um Amazon EC2-Instances mit Operationen wie Erstellen, Starten und Beenden zu steuern. Die Themen in diesem Abschnitt enthalten einige Beispiele dafür. Weitere Informationen zu EC2-Instances finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Informationen zu den APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt ([Arbeiten mit Amazon EC2](#)).

Themen

- [Starten einer Amazon EC2](#)
- [Beenden einer Amazon EC2-Instance](#)

Starten einer Amazon EC2

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um eine oder mehrere identisch konfigurierte Amazon EC2-Instances aus demselben Amazon Machine Image (AMI) zu starten. Mithilfe [mehrerer Eingaben](#), die Sie bereitstellen, startet die Anwendung eine EC2-Instance und überwacht dann die Instance, bis sie den Status „Ausstehend“ nicht mehr hat.

Wenn Ihre EC2-Instance ausgeführt wird, können Sie eine Remote-Verbindung zu ihr herstellen, wie unter beschrieben([optional](#)) [Herstellen einer Verbindung mit der Instance](#).

Sie können eine EC2-Instance in einer VPC oder in EC2-Classic starten (sofern Ihr AWS Konto dies unterstützt). Weitere Informationen zu EC2 in einer VPC im Vergleich zu EC2-Classic finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Warning

EC2-Classic wird am 15. August 2022 eingestellt. Wir empfehlen Ihnen die Migration von EC2-Classic zu einer VPC. Weitere Informationen finden Sie unter Migration von EC2-Classic zu einer VPC im [Amazon-EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon-](#)

[EC2-Benutzerhandbuch für Windows-Instances](#). Lesen Sie außerdem den Blogbeitrag [EC2-Classic Networking is Retiring – Here's How to Prepare](#) (EC2-Classic Networking wird außer Betrieb genommen – so bereiten Sie sich vor).

Die folgenden Abschnitte enthalten Ausschnitte und andere Informationen für dieses Beispiel. Der [vollständige Code für das Beispiel](#) wird nach den Codeausschnitten angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Sammeln Sie, was Sie benötigen](#)
- [Starten einer -Instance](#)
- [Überwachen der Instance](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)
- [\(optional\) Herstellen einer Verbindung mit der Instance](#)
- [Bereinigen](#)

Sammeln Sie, was Sie benötigen

Um eine EC2-Instance zu starten, benötigen Sie mehrere Dinge.

- Eine [VPC](#), in der die Instance gestartet wird. Wenn es sich um eine Windows-Instance handelt und Sie eine Verbindung über RDP herstellen, muss der VPC höchstwahrscheinlich ein Internet-Gateway sowie ein Eintrag für das Internet-Gateway in der Routing-Tabelle zugeordnet sein. Weitere Informationen finden Sie unter [Internet-Gateways](#) im Amazon-VPC-Benutzerhandbuch.
- Die ID eines vorhandenen Subnetzes in der VPC, in der die Instance gestartet wird. Eine einfache Möglichkeit, dies zu finden oder zu erstellen, besteht darin, sich bei der [Amazon-VPC-Konsole](#) anzumelden, aber Sie können es auch programmgesteuert mithilfe der [DescribeSubnetsAsync](#) Methoden [CreateSubnetAsync](#) und abrufen.

Note

Wenn Ihr AWS Konto EC2-Classic unterstützt und es sich um den Instance-Typ handelt, den Sie starten möchten, ist dieser Parameter nicht erforderlich. Wenn Ihr Konto jedoch

EC2-Classic nicht unterstützt und 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 Instance herstellen möchten, muss die zuvor erwähnte Sicherheitsgruppe über eine entsprechende eingehende Regel verfügen, die SSH-Datenverkehr auf Port 22 (Linux-Instance) oder RDP-Datenverkehr auf Port 3389 (Windows-Instance) zulässt. Informationen dazu finden Sie unter [Aktualisieren von Sicherheitsgruppen](#), einschließlich des [Weitere Überlegungen](#) Endes dieses Themas.
- Das Amazon Machine Image (AMI), das zum Erstellen der Instance verwendet wird. Weitere Informationen zu AMIs finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#). Lesen Sie beispielsweise mehr über freigegebene AMIs im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).
- Der Name eines vorhandenen EC2-Schlüsselpaars, das für die Verbindung mit der neuen Instance verwendet wird. Weitere Informationen finden Sie unter [Arbeiten mit Amazon EC2-Schlüsselpaaren](#).
- Der Name der PEM-Datei, die den privaten Schlüssel des zuvor erwähnten EC2-Schlüsselpaars enthält. Die PEM-Datei wird verwendet, wenn Sie sich [remote mit der Instance verbinden](#).

Starten einer -Instance

Der folgende Codeausschnitt startet eine EC2-Instance.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 der Instance

Der folgende Codeausschnitt überwacht die Instance, bis sie nicht mehr den Status „Ausstehend“ hat.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

Die gültigen Werte der `-Instance.State.CodeEigenschaft` finden Sie in der [-InstanceState](#)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.WriteLine(".");
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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

Klasse [InstanceType](#)

- Namespace [Amazon.EC2.Model](#)

Klasse [DescribeInstancesRequest](#)

Klasse [DescribeInstancesResponse](#)

Klassen-[Instance](#)

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
{
    // = = = = =
    // Class to launch an EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
            }
        }
    }
}
```



```
        return;
    }

    // Get the application arguments from the parsed list
    string groupID =
        CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
    string ami =
        CommandLine.GetArgument(parsedArgs, null, "-a", "--ami-id");
    string keyPairName =
        CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
    string subnetID =
        CommandLine.GetArgument(parsedArgs, null, "-s", "--subnet-id");
    if( (string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
        || (string.IsNullOrEmpty(ami) || !ami.StartsWith("ami-"))
        || (string.IsNullOrEmpty(keyPairName))
        || (!string.IsNullOrEmpty(subnetID) && !subnetID.StartsWith("subnet-")))
        CommandLine.ErrorExit(
            "\nOne or more of the required arguments is missing or incorrect." +
            "\nRun the command with no arguments to see help.");

    // Create an EC2 client
    var ec2Client = new AmazonEC2Client();

    // Create an object with the necessary properties
    RunInstancesRequest request = GetRequestData(groupID, ami, keyPairName,
        subnetID);

    // Launch the instances and wait for them to start running
    var instanceIds = await LaunchInstances(ec2Client, request);
    await CheckState(ec2Client, instanceIds);
}

//
// Method to put together the properties needed to launch the instance.
private static RunInstancesRequest GetRequestData(
    string groupID, string ami, string keyPairName, string subnetID)
{
    // Common properties
    var groupIDs = new List<string>() { groupID };
    var request = new RunInstancesRequest()
    {
        // The first three of these would be additional command-line arguments or
        similar.
    }
}
```

```
        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
    };

    // Properties specifically for EC2 in a VPC.
    if(!string.IsNullOrEmpty(subnetID))
    {
        request.NetworkInterfaces =
            new List<InstanceNetworkInterfaceSpecification>() {
                new InstanceNetworkInterfaceSpecification() {
                    DeviceIndex = 0,
                    SubnetId = subnetID,
                    Groups = groupIDs,
                    AssociatePublicIpAddress = true
                }
            };
    }

    // Properties specifically for EC2-Classic
    else
    {
        request.SecurityGroupIds = groupIDs;
    }
    return request;
}

//
// Method to launch the instances
// Returns a list with the launched instance IDs
private static async Task<List<string>> LaunchInstances(
    IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
{
    var instanceIds = new List<string>();
    RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

    Console.WriteLine("\nNew instances have been created.");
    foreach (Instance item in responseLaunch.Reservation.Instances)
    {
        instanceIds.Add(item.InstanceId);
    }
}
```

```
        Console.WriteLine($" New instance: {item.InstanceId}");
    }

    return instanceIds;
}

//
// Method to wait until the instances are running (or at least not pending)
private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
{
    Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

    int numberRunning;
    DescribeInstancesResponse responseDescribe;
    var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

    // Check every couple of seconds
    int wait = 2000;
    while(true)
    {
        // Get and check the status for each of the instances to see if it's past
        pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.WriteLine(".");
        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-Instance überprüfen, können Sie der `-Filter`Eigenschaft des [-DescribeInstancesRequest](#) Objekts einen Filter hinzufügen. Mit dieser Technik können Sie die Anforderung auf bestimmte Instances beschränken, z. B. Instances mit einem bestimmten benutzerdefinierten Tag.
- Der Kürze halber wurden einigen Eigenschaften typische Werte gegeben. Eine oder alle dieser Eigenschaften können stattdessen programmgesteuert oder durch Benutzereingaben bestimmt werden.
- Die Werte, die Sie für die `MaxCount` Eigenschaften `MinCount` und des [RunInstancesRequest](#) Objekts verwenden können, werden durch die Ziel-Availability Zone und die maximale Anzahl von Instances bestimmt, die für den Instance-Typ zulässig sind. Weitere Informationen finden Sie unter [Wie viele Instances kann ich in Amazon EC2 ausführen?](#) im Amazon EC2 General FAQ.

- Wenn Sie einen anderen Instance-Typ als dieses Beispiel verwenden möchten, stehen Ihnen mehrere Instance-Typen zur Auswahl, die Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#) sehen können.
- Sie können auch eine [IAM-Rolle](#) an eine Instance anfügen, wenn Sie sie starten. Erstellen Sie dazu ein [IamInstanceProfileSpecification](#) Objekt, dessen Name Eigenschaft auf den Namen einer IAM-Rolle festgelegt ist. Fügen Sie dieses Objekt dann der `-IamInstanceProfile` Eigenschaft des `-RunInstancesRequest` Objekts hinzu.

Note

Um eine EC2-Instance zu starten, der eine IAM-Rolle zugeordnet ist, muss die Konfiguration eines IAM-Benutzers bestimmte Berechtigungen enthalten. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

(optional) Herstellen einer Verbindung mit der Instance

Nachdem eine Instance ausgeführt wurde, können Sie sich mithilfe des entsprechenden Remote-Clients `remote` mit ihr verbinden. Sowohl für Linux- als auch für Windows-Instances benötigen Sie die öffentliche IP-Adresse oder den öffentlichen DNS-Namen der Instance. Sie benötigen außerdem Folgendes.

Für Linux-Instances

Sie können einen SSH-Client verwenden, um eine Verbindung zu Ihrer Linux-Instance herzustellen. Stellen Sie sicher, dass die Sicherheitsgruppe, die Sie beim Starten der Instance verwendet haben, SSH-Datenverkehr auf Port 22 zulässt, wie unter beschrieben [Aktualisieren von Sicherheitsgruppen](#).

Sie benötigen auch den privaten Teil des Schlüsselpaars, das Sie zum Starten der Instance verwendet haben, d. h. die PEM-Datei.


Weitere Informationen finden Sie unter Herstellen einer [Verbindung mit Ihrer Linux-Instance](#) im Amazon EC2-Benutzerhandbuch für Linux-Instances.

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 Starten der Instance verwendet haben, RDP-Datenverkehr auf Port 3389 zulässt, wie unter beschrieben [Aktualisieren von Sicherheitsgruppen](#).

Sie benötigen auch das Administratorpasswort. Sie können dies mit dem folgenden Beispielcode erreichen, der die Instance-ID und den privaten Teil des Schlüsselpaars erfordert, das zum Starten der Instance verwendet wird, d. h. die PEM-Datei.

Weitere Informationen finden Sie unter [Herstellen einer Verbindung mit Ihrer Windows-Instance](#) im Amazon EC2-Benutzerhandbuch für Windows-Instances.

 Warning

Dieser Beispielcode gibt das Klartext-Administratorpasswort für Ihre Instance zurück.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

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
{
    // = = = = =
    // Class to get the Administrator password of a Windows EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }

            // 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($"Password: {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.");
}
}

// = = = = =
// 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);
}
}
```

Bereinigen

Wenn Sie Ihre EC2-Instance nicht mehr benötigen, müssen Sie sie beenden, wie unter [beschrieben](#) [Beenden einer Amazon EC2-Instance](#).

Beenden einer Amazon EC2-Instance

Wenn Sie eine oder mehrere Ihrer Amazon EC2-Instances nicht mehr benötigen, können Sie sie beenden.

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um EC2-Instances zu beenden. Es verwendet eine Instance-ID als Eingabe.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

- Namespace [Amazon.EC2.Model](#)

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, sollten Sie sich bei der [Amazon EC2-Konsole](#) anmelden, um zu überprüfen, ob die [EC2](#)-Instance beendet wurde.

Amazon EC2 Spot Instance-Tutorial

In diesem Tutorial erfahren Sie, wie Sie AWS SDK for .NET Amazon Amazon EC2-Spot-Instances mit der verwalten.

Übersicht

Mit Spot-Instances können Sie ungenutzte Amazon EC2-Kapazität für weniger als den On-Demand-Preis anfordern. Dies kann Ihre EC2-Kosten für Anwendungen, die unterbrochen werden können, erheblich senken.

Im Folgenden finden Sie eine allgemeine Übersicht darüber, wie Spot-Instances angefordert und verwendet werden.

1. Erstellen Sie eine Spot-Instance-Anforderung und geben Sie den Höchstpreis an, den Sie zu zahlen bereit sind.
2. Wenn die Anforderung erfüllt ist, führen Sie die Instance wie jede andere Amazon EC2 aus.
3. Führen Sie die Instance so lange wie gewünscht aus und beenden Sie sie dann, es sei denn, der Spot-Preis ändert sich so, dass die Instance für Sie beendet wird.
4. Bereinigen Sie die Spot-Instance-Anforderung, wenn Sie sie nicht mehr benötigen, damit Spot-Instances nicht mehr erstellt werden.

Dies war ein sehr allgemeiner Überblick über Spot-Instances. Sie können Spot-Instances besser verstehen, indem Sie sich im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#) darüber informieren.

Über dieses Tutorial

Wenn Sie diesem Tutorial folgen, verwenden Sie die AWS SDK for .NET für Folgendes:

- Erstellen einer Spot-Instance-Anforderung
- Ermitteln, wann die Spot-Instance-Anforderung erfüllt wurde
- Stornieren der Spot-Instance-Anforderung
- Beenden von dazugehörigen Instances

Die folgenden Abschnitte enthalten Ausschnitte und andere Informationen für dieses Beispiel. Der [vollständige Code für das Beispiel](#) wird nach den Codeausschnitten angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Voraussetzungen](#)
- [Sammeln Sie, was Sie benötigen](#)
- [Erstellen einer Spot-Instance-Anforderung](#)
- [Ermitteln des Status Ihrer Spot-Instance-Anforderung](#)
- [Bereinigen Ihrer Spot-Instance-Anforderungen](#)
- [Bereinigen Ihrer Spot-Instances](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Voraussetzungen

Informationen zu den APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt ([Arbeiten mit Amazon EC2](#)).

Sammeln Sie, was Sie benötigen

Um eine Spot-Instance-Anforderung zu erstellen, benötigen Sie mehrere Dinge.

- Die Anzahl der Instances und deren Instance-Typ. Es stehen mehrere Instance-Typen zur Auswahl, die Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#) sehen können. Die Standardnummer für dieses Tutorial ist 1.
- Das Amazon Machine Image (AMI), das zum Erstellen der Instance verwendet wird. Weitere Informationen zu AMIs finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#). Lesen Sie beispielsweise mehr über freigegebene AMIs im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).
- 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 [Amazon EC2-Preisseite](#) anzeigen. Der Standardpreis für dieses Tutorial wird später erläutert.
- Wenn Sie eine Remote-Verbindung zu einer Instance herstellen möchten, eine Sicherheitsgruppe mit der entsprechenden Konfiguration und den entsprechenden Ressourcen. Dies wird unter [Arbeiten mit Sicherheitsgruppen in Amazon EC2](#) und die Informationen zum [Sammeln von benötigtem](#) und [zum Herstellen einer Verbindung mit einer](#) Instance in beschrieben [Starten einer Amazon EC2](#). Der Einfachheit halber verwendet dieses Tutorial die Sicherheitsgruppe namens default, die alle neueren AWS Konten haben.

Es gibt viele Möglichkeiten zum Anfordern von Spot-Instances. Im Folgenden sind gängige Strategien aufgeführt:

- Stellen Sie Anfragen, die garantiert weniger kosten als On-Demand-Preise.
- Stellen Sie Anfragen basierend auf dem Wert der resultierenden Berechnung.
- Stellen Sie Anfragen, um so schnell wie möglich Rechenkapazität zu erhalten.

Die folgenden Erläuterungen beziehen sich auf den Spot-Preisverlauf im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Reduzieren Sie die Kosten unter On-Demand

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 überprüfen den Spot-Preisverlauf auf Instance-Typen, indem Sie entweder die Amazon EC2-Konsole oder die Amazon EC2-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 Anforderung am oberen Ende des Bereichs von Spot-Preisen an, die immer noch unter dem On-Demand-Preis liegen, wobei davon ausgegangen wird, dass Ihre einmalige Spot-Instance-Anforderung höchstwahrscheinlich erfüllt wird und für genügend aufeinanderfolgende Rechenzeit ausgeführt wird, um den Auftrag 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.

Bezahlen Sie nicht mehr als den Wert des Ergebnisses

Sie haben eine Aufgabe zur Datenverarbeitung, die ausgeführt werden soll. Sie verstehen den Wert der Ergebnisse des Auftrags gut genug, um zu wissen, wie viel sie in Bezug auf die Rechenkosten wert sind.

Nachdem Sie den Spot-Preis-Verlauf für Ihren Instance-Typ analysiert haben, wählen Sie einen Preis aus, zu dem die Kosten der Rechenzeit nicht mehr als der Wert der Ergebnisse des Auftrags betragen. Sie erstellen eine persistente Anforderung und lassen es zwischenzeitlich laufen, sobald der Spot-Preis Ihr Gebot erreicht ist oder darunter sinkt.

Schnelles Erlangen von Rechenkapazität

Sie haben einen unerwarteten, kurzfristigen Bedarf an zusätzlicher Kapazität, die nicht über On-Demand-Instances verfügbar ist. Nachdem Sie den Spot-Preis-Verlauf für Ihren Instance-Typ analysiert haben, wählen Sie einen Preis über dem höchsten historischen Preis aus, um die Wahrscheinlichkeit, dass Ihre Anfrage schnell erfüllt wird, erheblich zu erhöhen und die Datenverarbeitung fortzusetzen, bis sie abgeschlossen ist.

Nachdem Sie sich vergewissert haben, was Sie benötigen, 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 Preis auf diese Weise festgelegt wird, erhöhen sich die Chancen zur Erfüllung der Anfrage.

Erstellen einer Spot-Instance-Anforderung

Der folgende Codeausschnitt zeigt Ihnen, wie Sie eine Spot-Instance-Anforderung mit den Elementen erstellen, die Sie zuvor gesammelt haben.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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 [SpotInstanceRequest](#) Objekts enthalten ist.

Note

Ihnen werden alle gestarteten Spot-Instances in Rechnung gestellt. Um unnötige Kosten zu vermeiden, stellen Sie sicher, dass Sie [alle Anfragen abbrechen](#) und [alle Instances beenden](#).

Ermitteln des Status Ihrer Spot-Instance-Anforderung

Der folgende Codeausschnitt zeigt Ihnen, wie Sie Informationen zu Ihrer Spot-Instance-Anforderung abrufen. Sie können diese Informationen verwenden, um bestimmte Entscheidungen in Ihrem Code zu treffen, z. B. ob Sie weiterhin darauf warten möchten, dass eine Spot-Instance-Anforderung erfüllt wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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-Anforderung zurück, z. B. die Instance-ID, ihren Status und den Statuscode. Die Statuscodes für Spot-Instance-Anforderungen finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Bereinigen Ihrer Spot-Instance-Anforderungen

Wenn Sie keine Spot-Instances mehr anfordern müssen, ist es wichtig, alle ausstehenden Anforderungen abzuberechnen, um zu verhindern, dass diese Anforderungen erneut erfüllt werden. Der folgende Codeausschnitt zeigt Ihnen, wie Sie eine Spot-Instance-Anforderung abbrechen.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 Ihrer Spot-Instances

Um unnötige Kosten zu vermeiden, ist es wichtig, dass Sie alle Instances beenden, die über Spot-Instance-Anforderungen gestartet wurden. Durch das einfache Abbrechen von Spot-Instance-Anforderungen werden Ihre Instances nicht beendet, was bedeutet, dass Ihnen weiterhin Gebühren in Rechnung gestellt werden. Der folgende Codeausschnitt 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 diesen verwendeten Ausschnitt.

```
//  
// 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-Anforderung zu erstellen und abzubrechen und eine Spot-Instance zu beenden.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.EC2](#)

Programmiererelemente:

- Namespace [Amazon.EC2](#)

Klasse [AmazonEC2Client](#)

Klasse [InstanceType](#)

- Namespace [Amazon.EC2.Model](#)

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($"Spot 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, sollten Sie sich bei der [Amazon EC2-Konsole](#) anmelden, um zu überprüfen, ob die [Spot-Instance-Anforderung](#) abgebrochen wurde und ob die [Spot-Instance](#) beendet wurde.

Zugreifen auf AWS Identity and Access Management (IAM) mit der AWS SDK for .NET

unterstützt AWS SDK for .NET [AWS Identity and Access Management](#), einen Webservice, mit dem AWS Kunden Benutzer und Benutzerberechtigungen in verwalten können AWS.

Ein AWS Identity and Access Management (IAM)-Benutzer ist eine Entität, die Sie in erstellen AWS. Die Entität stellt eine Person oder Anwendung dar, die mit interagiert AWS. Weitere Informationen zu IAM-Benutzern finden Sie unter [IAM-Benutzer](#) und [IAM- und STS-Limits](#) im IAM-Benutzerhandbuch.

Sie erteilen einem Benutzer Berechtigungen, indem Sie eine IAM-Richtlinie erstellen. Die Richtlinie enthält ein Richtliniendokument, das die Aktionen auflistet, 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

stellt APIs für IAM-Clients AWS SDK for .NET bereit. Mit den APIs können Sie mit IAM-Funktionen wie Benutzern, Rollen und Zugriffsschlüsseln arbeiten.

Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen die Muster zeigen, die Sie bei der Arbeit mit diesen APIs befolgen können. Um den vollständigen Satz von APIs anzuzeigen, sehen Sie sich die [AWS SDK for .NET API-Referenz](#) an (und scrollen Sie zu „AmazonIdentityManagement“).

Dieser Abschnitt enthält auch [ein Beispiel](#), das Ihnen zeigt, wie Sie eine IAM-Rolle an Amazon EC2 anfügen, um die Verwaltung von Anmeldeinformationen zu vereinfachen.

Die IAM-APIs werden vom [AWSSDK-IdentityManagement](#) NuGetPaket bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

Themen

- [Erstellen von IAM-verwalteten Richtlinien aus JSON](#)
- [Anzeigen des Richtliniendokuments einer von IAM verwalteten Richtlinie](#)
- [Gewähren des Zugriffs mithilfe einer IAM-Rolle](#)

Erstellen von IAM-verwalteten Richtlinien aus JSON

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um eine von [IAM verwaltete Richtlinie](#) aus einem bestimmten Richtliniendokument in JSON zu erstellen. Die Anwendung erstellt ein IAM-Clientobjekt, 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 Ausschnitte 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

Der folgende Codeausschnitt erstellt eine von IAM verwaltete Richtlinie mit dem angegebenen Namen und Richtliniendokument.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.IdentityManagement](#)

Programmiererelemente:

- Namespace [Amazon.IdentityManagement](#)
Klasse [AmazonIdentityManagementServiceClient](#)
- Namespace [AmazonIdentityManagement.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
{
    // = = = = =
    // Class to create an IAM policy with a given policy document
    class Program
    {
        private const int MaxArgs = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
            {
                PrintHelp();
                return;
            }

            // Get the application arguments from the parsed list
            string policyName =
                CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-name");
            string policyFilename =
                CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
            if( string.IsNullOrEmpty(policyName)
                || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
                CommandLine.ErrorExit(
                    "\nOne or more of the required arguments is missing or incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Create an IAM service client
            var iamClient = new AmazonIdentityManagementServiceClient();

            // Create the new policy
```

```

    var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
    Console.WriteLine($"Policy {response.Policy.PolicyName} has been created.");
    Console.WriteLine($"  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.");
}
}

// = = = = =
// 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

- Im Folgenden finden Sie ein Beispiel für ein Richtlinienokument, das Sie in eine JSON-Datei kopieren und als Eingabe für diese Anwendung verwenden können:

```
{
  "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 überprüfen, ob die Richtlinie erstellt wurde, indem Sie sich in der [IAM-Konsole](#) ansehen. Wählen Sie in der Dropdown-Liste Filterrichtlinien die Option Kundenverwaltet aus. Löschen Sie die Richtlinie, wenn Sie sie nicht mehr benötigen.
- Weitere Informationen zur Richtlinienerstellung finden Sie unter [Erstellen von IAM-Richtlinien](#) und in der [IAM-JSON-Richtlinienreferenz](#) im [IAM-Benutzerhandbuch](#).

Anzeigen des Richtliniendokuments einer von IAM verwalteten Richtlinie

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um ein Richtliniendokument anzuzeigen. Die Anwendung erstellt ein IAM-Clientobjekt, findet die Standardversion der angegebenen von IAM verwalteten Richtlinie und zeigt das Richtliniendokument dann in JSON an.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Suchen der Standardversion](#)
- [Anzeigen des Richtliniendokuments](#)
- [Vollständiger Code](#)

Suchen der Standardversion

Der folgende Codeausschnitt findet die Standardversion der angegebenen IAM-Richtlinie.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 responseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
            PolicyArn = policyArn});

    // Find the default version
    foreach(PolicyVersion version in responseVersions.Versions)
    {
        if(version.IsDefaultVersion)
        {
            defaultVersion = version.VersionId;
            break;
        }
    }

    return defaultVersion;
}
```

Anzeigen des Richtliniendokuments

Der folgende Codeausschnitt zeigt das Richtliniendokument in JSON der angegebenen IAM-Richtlinie an.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.IdentityManagement](#)

Programmierelemente:

- Namespace [Amazon.IdentityManagement](#)
 - Klasse [AmazonIdentityManagementServiceClient](#)
- Namespace [AmazonIdentityManagement.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.Modell;  
  
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)}");
}
}
}
```

Gewähren des Zugriffs mithilfe einer IAM-Rolle

Dieses Tutorial zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um IAM-Rollen auf Amazon EC2 zu aktivieren.

Übersicht

Alle Anfragen an AWS müssen kryptografisch signiert werden, indem Anmeldeinformationen verwendet werden, die von ausgestellt wurden AWS. Daher benötigen Sie eine Strategie zur Verwaltung von Anmeldeinformationen für Anwendungen, die auf Amazon EC2 ausgeführt werden.

Sie müssen diese Anmeldeinformationen sicher verteilen, speichern und rotieren, aber auch für die Anwendungen zugänglich halten.

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 an eine EC2-Instance an. Weitere Informationen zu den Vorteilen der Verwendung von IAM-Rollen finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#). Weitere Informationen zu [IAM-Rollen](#) finden Sie auch im IAM-Benutzerhandbuch.

Wenn die Anwendung für eine Anwendung, die mit der erstellt wurde AWS SDK for .NET, ein Clientobjekt für einen - AWS Service erstellt, sucht das Objekt nach Anmeldeinformationen aus mehreren potenziellen Quellen. Die Reihenfolge, in der sie durchsucht, wird in angezeigt [Auflösung von Anmeldeinformationen und Profilen](#).

Wenn das Clientobjekt keine Anmeldeinformationen aus einer anderen Quelle findet, ruft es temporäre Anmeldeinformationen ab, die dieselben Berechtigungen haben wie diejenigen, die in der IAM-Rolle konfiguriert wurden und sich in den Metadaten der EC2-Instance befinden. Diese Anmeldeinformationen werden verwendet, um Aufrufe an AWS vom Client-Objekt aus zu tätigen.

Über dieses Tutorial

Wenn Sie diesem Tutorial folgen, verwenden Sie die AWS SDK for .NET (und andere Tools), um eine Amazon EC2-Instance mit einer angehängten IAM-Rolle zu starten, und sehen dann eine Anwendung auf der Instance mit den Berechtigungen der IAM-Rolle.

Themen

- [Erstellen einer Amazon S3-Beispielanwendung](#)
- [Erstellen einer IAM-Rolle](#)
- [Starten einer EC2-Instance und Anfügen der IAM-Rolle](#)
- [Herstellen einer Verbindung mit der EC2-Instance](#)
- [Ausführen der Beispielanwendung auf der EC2-Instance](#)
- [Bereinigen](#)

Erstellen einer Amazon S3-Beispielanwendung

Diese Beispielanwendung ruft ein Objekt aus Amazon S3 ab. Um die Anwendung auszuführen, benötigen Sie Folgendes:

- Ein Amazon S3-Bucket, der eine Textdatei enthält.
- AWS -Anmeldeinformationen auf Ihrem Entwicklungscomputer, mit denen Sie auf den Bucket zugreifen können.

Informationen zum Erstellen eines Amazon S3-Buckets und zum Hochladen eines Objekts finden Sie im [Benutzerhandbuch für Amazon Simple Storage Service](#). Weitere Informationen zu AWS Anmeldeinformationen finden Sie unter [Konfigurieren Sie die SDK-Authentifizierung mit AWS](#).

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, mit der Sie die Anwendung ausführen können, ohne sie zu veröffentlichen. Wenn Sie später in diesem Tutorial eine EC2-Instance erstellen, können Sie die .NET Core Runtime auf der Instance installieren. Dies bietet Ihnen eine ähnliche Erfahrung und eine kleinere Dateiübertragung. Sie können die .NET Core Runtime jedoch auch nicht auf der Instance installieren. Wenn Sie diese Vorgehensweise wählen, müssen Sie die Anwendung veröffentlichen, damit alle Abhängigkeiten beim Übertragen auf die Instance berücksichtigt werden.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.S3](#)

Programmirelemente:

- 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
{
    // = = = = =
    // Class to retrieve a text file from an S3 bucket and write it to a local file
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            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.");
}
}

// = = = = =
// 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);
}
}
}
```

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 Service aus, suchen und wählen Sie EC2 und wählen Sie Weiter: Berechtigungen aus.
4. Suchen Sie unter Berechtigungsrichtlinien anfügen nach AmazonS3ReadOnlyAccess und wählen Sie es aus. Überprüfen Sie die Richtlinie, wenn Sie möchten, und wählen Sie dann Weiter: Tags aus.
5. Fügen Sie Tags hinzu, wenn Sie möchten, und wählen Sie dann Weiter: Überprüfen aus.

6. Geben Sie einen Namen und eine Beschreibung für die Rolle ein und wählen Sie dann Create role aus. Notieren Sie sich diesen Namen, da Sie ihn benötigen, wenn Sie die EC2-Instance starten.

Starten einer EC2-Instance und Anfügen der IAM-Rolle

Starten Sie eine EC2-Instance mit der IAM-Rolle, die Sie zuvor erstellt haben. Sie können dies wie folgt tun.

- Verwenden der EC2-Konsole

Folgen Sie den Anweisungen zum Starten einer Instance im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Während Sie den Assistenten durchlaufen, sollten Sie mindestens die Seite Configure Instance Details besuchen, damit Sie die zuvor erstellte IAM-Rolle auswählen können.

- Verwenden der AWS SDK for .NET

Weitere Informationen dazu finden Sie unter [Starten einer Amazon EC2](#), einschließlich des [Weitere Überlegungen](#) Endes dieses Themas.

Um eine EC2-Instance zu starten, der eine IAM-Rolle zugeordnet ist, muss die Konfiguration eines IAM-Benutzers bestimmte Berechtigungen enthalten. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

Herstellen einer Verbindung mit der EC2-Instance

Stellen Sie eine Verbindung mit der EC2-Instance her, damit Sie die Beispielanwendung dorthin übertragen und dann die Anwendung ausführen können. Sie benötigen die -Datei, die den privaten Teil des Schlüsselpaars enthält, das Sie zum Starten der Instance verwendet haben, d. h. die PEM-Datei.

Sie können dies tun, indem Sie das Verbindungsverfahren im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#) befolgen. 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 sich auch mit dem Toolkit for Visual Studio mit der Instance verbinden. Weitere Informationen finden Sie unter [Herstellen einer Verbindung mit einer Amazon EC2-Instance](#) im AWS Toolkit for Visual Studio -Benutzerhandbuch.

Ausführen der Beispielanwendung auf der EC2-Instance

1. Kopieren Sie die Anwendungsdateien von Ihrem lokalen Laufwerk auf Ihre Instance.

Welche Dateien Sie übertragen, hängt davon ab, wie Sie die Anwendung erstellt haben und ob auf Ihrer Instance die .NET Core Runtime installiert ist. Informationen zum Übertragen von Dateien auf Ihre Instance finden Sie im [Amazon EC2-Benutzerhandbuch für Linux-Instances](#) oder im [Amazon EC2-Benutzerhandbuch für Windows-Instances](#).

2. Starten Sie die Anwendung und überprüfen Sie, ob 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 [Amazon EC2-Konsole](#).
 - b. Wählen Sie die Instance aus und trennen Sie die IAM-Rolle über Aktionen , Instance-Einstellungen , IAM-Rolle anfügen/ersetzen .
 - c. Führen Sie die Anwendung erneut aus und sehen Sie, dass sie einen Autorisierungsfehler zurückgibt.

Bereinigen

Wenn Sie mit diesem Tutorial fertig sind und die von Ihnen erstellte EC2-Instance nicht mehr benötigen, müssen Sie die Instance beenden, um unerwünschte Kosten zu vermeiden. Sie können dies in der [Amazon EC2-Konsole](#) oder programmgesteuert tun, wie unter beschrieben [Beenden einer Amazon EC2-Instance](#). 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

Die AWS SDK for .NET unterstützt [Amazon S3](#), bei dem es sich um Speicher für das Internet handelt. Der Service ist darauf ausgelegt, Cloud Computing für Entwickler zu erleichtern.

APIs

stellt APIs für Amazon S3-Clients AWS SDK for .NET bereit. Mit den APIs können Sie mit Amazon S3-Ressourcen wie Buckets und Elementen arbeiten. Die vollständigen APIs für Amazon S3 finden Sie im Folgenden:

- [AWS SDK for .NET API-Referenz](#) (und scrollen Sie zu „Amazon.S3“).
- Dokumentation zu [Amazon.Extensions.S3.Encryption](#)

Die Amazon S3-APIs werden von den folgenden NuGet Paketen bereitgestellt:

- [AWSSDK.S3](#)
- [Amazon.Extensions.S3.Verschlüsselung](#)

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Beispiele in diesem Dokument

Die folgenden Themen in diesem Dokument zeigen Ihnen, wie Sie die AWS SDK for .NET für die Arbeit mit Amazon S3 verwenden.

- [Verwenden von KMS-Schlüsseln für die S3-Verschlüsselung](#)

Beispiele in anderen Dokumenten

Die folgenden Links zum [Amazon S3-Entwicklerhandbuch](#) enthalten zusätzliche Beispiele für die Verwendung von AWS SDK for .NET für die Arbeit mit Amazon S3.

Note

Obwohl diese Beispiele und zusätzliche Überlegungen zur Programmierung für Version 3 des AWS SDK for .NET mit .NET Framework erstellt wurden, sind sie auch für spätere Versionen des AWS SDK for .NET mit .NET Core geeignet. Manchmal sind kleine Anpassungen des Codes erforderlich.

Amazon S3-Programmierbeispiele

- [ACLs verwalten](#)
- [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 for .NET für Amazon 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](#)

Verwenden von AWS KMS Schlüsseln für die Amazon S3-Verschlüsselung in der AWS SDK for .NET

Dieses Beispiel zeigt Ihnen, wie Sie - AWS Key Management Service Schlüssel verwenden, um Amazon S3-Objekte zu verschlüsseln. Die Anwendung erstellt einen Kundenmasterschlüssel (CMK) und verwendet ihn, um ein [AmazonS3EncryptionClientV2](#)-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-Bucket zu erstellen. Anschließend entschlüsselt es das Objekt und zeigt seinen Inhalt an.

Warning

Eine ähnliche Klasse namens `AmazonS3EncryptionClient` ist veraltet und weniger sicher als die `AmazonS3EncryptionClientV2` Klasse. Informationen zum Migrieren von vorhandenem Code, der verwendet `AmazonS3EncryptionClient`, finden Sie unter [S3-Verschlüsselungs-Client-Migration](#).

Themen

- [Erstellen von Verschlüsselungsmaterialien](#)
- [Erstellen und Verschlüsseln eines Amazon S3-Objekts](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Erstellen von Verschlüsselungsmaterialien

Der folgende Codeausschnitt erstellt ein `EncryptionMaterials` Objekt, das eine KMS-Schlüssel-ID enthält.

Das Beispiel [am Ende dieses Themas](#) zeigt dieses Snippet in Verwendung.

```
// 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);
```


Erstellen und Verschlüsseln eines Amazon S3-Objekts

Der folgende Ausschnitt erstellt ein `AmazonS3EncryptionClientV2` Objekt, das die zuvor erstellten Verschlüsselungsmaterialien verwendet. Anschließend wird der Client verwendet, um ein neues Amazon S3-Objekt zu erstellen und zu verschlüsseln.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [Amazon.Extensions.S3.Verschlüsselung](#)

Programmiererelemente:

- Namespace [Amazon.Extensions.S3.Verschlüsselung](#)
 - Klasse [AmazonS3EncryptionClientV2](#)
 - Klasse [AmazonS3CryptoConfigurationV2](#)
 - 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 [AmazonKeyManagementService.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
{
    // = = = = =
    // Class to store text in an encrypted S3 object.
    class Program
    {
        private const int MaxArgs = 3;

        public static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
            {
                PrintHelp();
                return;
            }

            // Get the application arguments from the parsed list
            string 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($"The 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.");
}

}

// = = = = =
// 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

- Sie können die Ergebnisse dieses Beispiels überprüfen. Rufen Sie dazu die [Amazon S3-Konsole](#) auf und öffnen Sie den Bucket, den Sie der Anwendung bereitgestellt haben. Suchen Sie dann das neue Objekt, laden Sie es herunter und öffnen Sie es in einem Texteditor.
- Die [AmazonS3EncryptionClientV2](#)-Klasse implementiert dieselbe Schnittstelle wie die `AmazonS3Client` Standardklasse. Dies erleichtert die Portierung Ihres Codes in die `-AmazonS3EncryptionClientV2`-Klasse, sodass Verschlüsselung und Entschlüsselung automatisch und transparent im Client erfolgen.
- Ein Vorteil der Verwendung eines - AWS KMS Schlüssels als Masterschlüssel besteht darin, dass Sie Ihre eigenen Masterschlüssel nicht speichern und verwalten müssen. Dies geschieht durch AWS. Ein zweiter Vorteil besteht darin, dass die `AmazonS3EncryptionClientV2` Klasse von `AmazonS3EncryptionClientV2` Klasse von interoperabel AWS SDK for .NET ist AWS SDK for Java. Das bedeutet, dass Sie mit dem verschlüsseln AWS SDK for Java und mit dem entschlüsseln können AWS SDK for .NET und umgekehrt.

Note

Die `AmazonS3EncryptionClientV2` Klasse von AWS SDK for .NET unterstützt KMS-Masterschlüssel nur, wenn sie im Metadatenmodus ausgeführt werden. Der Befehlsdateimodus der `AmazonS3EncryptionClientV2` Klasse von AWS SDK for .NET ist nicht mit der `AmazonS3EncryptionClientV2` Klasse von kompatibel AWS SDK for Java.

- Weitere Informationen zur clientseitigen Verschlüsselung mit der `-AmazonS3EncryptionClientV2`-Klasse und zur Funktionsweise der Envelope-Verschlüsselung finden Sie unter [Clientseitige Datenverschlüsselung mit AWS SDK for .NET und Amazon S3](#).

Senden von Benachrichtigungen aus der Cloud mit Amazon Simple Notification Service

Note

Die Informationen in diesem Thema beziehen sich auf Projekte, die auf .NET Framework und der AWS SDK for .NET Version 3.3 und früher basieren.

unterstützt AWS SDK for .NET Amazon Simple Notification Service (Amazon SNS), einen Webservice, mit dem Anwendungen, Endbenutzer und Geräte sofort Benachrichtigungen aus der Cloud senden können. Weitere Informationen finden Sie unter [Amazon SNS](#).

Auflisten Ihrer Amazon SNS-Themen

Das folgende Beispiel zeigt, wie Sie Ihre Amazon SNS-Themen, die Abonnements für jedes Thema und die Attribute für jedes Thema auflisten. In diesem Beispiel wird die Standardeinstellung verwendet [AmazonSimpleNotificationServiceClient](#).

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;

var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
    response = client.ListTopics(request);

    foreach (var topic in response.Topics)
    {
        Console.WriteLine("Topic: {0}", topic.TopicArn);

        var subs = client.ListSubscriptionsByTopic(
            new ListSubscriptionsByTopicRequest
            {
                TopicArn = topic.TopicArn
            });

        var ss = subs.Subscriptions;
```



```
if (ss.Any())
{
    Console.WriteLine(" Subscriptions:");

    foreach (var sub in ss)
    {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
    }
}

var attrs = client.GetTopicAttributes(
    new GetTopicAttributesRequest
    {
        TopicArn = topic.TopicArn
    }).Attributes;

if (attrs.Any())
{
    Console.WriteLine(" Attributes:");

    foreach (var attr in attrs)
    {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
    }
}

Console.WriteLine();
}

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

Senden einer Nachricht an ein Amazon SNS-Thema

Das folgende Beispiel zeigt, wie Sie eine Nachricht an ein Amazon SNS-Thema senden. Das Beispiel verwendet ein Argument, den ARN des Amazon 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);
            }
        }
    }
}
```

```
}  
}
```

Sehen Sie sich das [vollständige Beispiel an](#), einschließlich Informationen zum Erstellen und Ausführen des Beispiels über die Befehlszeile auf 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);
    }
}
}
```

Sehen Sie sich das [vollständige Beispiel an](#), einschließlich Informationen zum Erstellen und Ausführen des Beispiels über die Befehlszeile auf GitHub.

Messaging mit Amazon SQS

unterstützt AWS SDK for .NET [Amazon Simple Queue Service \(Amazon SQS\)](#), einen Nachrichtenwarteschlangenservice, der Nachrichten oder Workflows zwischen Komponenten in einem System verarbeitet.

Amazon SQS-Warteschlangen bieten einen Mechanismus, mit dem Sie Nachrichten zwischen Softwarekomponenten wie Microservices, verteilten Systemen und Serverless-Anwendungen senden, speichern und empfangen können. Auf diese Weise können Sie solche Komponenten entkoppeln und Ihr eigenes Messaging-System entwerfen und betreiben. Informationen darüber, wie Warteschlangen und Nachrichten in Amazon SQS funktionieren, finden Sie unter [Amazon SQS-Tutorials](#) und [Grundlegende Amazon SQS-Architektur](#) im [Amazon-Simple-Queue-Service-Entwicklerhandbuch](#).

Important

Aufgrund der verteilten Natur von Warteschlangen kann Amazon SQS nicht garantieren, dass Sie Nachrichten in der genauen Reihenfolge erhalten, in der sie gesendet werden. Wenn Sie die Nachrichtenreihenfolge beibehalten müssen, verwenden Sie eine [Amazon SQS-FIFO-Warteschlange](#).

APIs

stellt APIs für Amazon SQS-Clients AWS SDK for .NET bereit. Mit den APIs können Sie mit Amazon SQS-Funktionen wie Warteschlangen und Nachrichten arbeiten. Dieser Abschnitt enthält eine kleine

Anzahl von Beispielen, die Ihnen die Muster zeigen, die Sie bei der Arbeit mit diesen APIs befolgen können. Um den vollständigen Satz von APIs anzuzeigen, sehen Sie sich die [AWS SDK for .NET API-Referenz](#) an (und scrollen Sie zu „Amazon.SQS“).

Die Amazon SQS-APIs werden vom [AWSSDKSQS](#) NuGet -Paket bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

Themen

- [Erstellen von Amazon SQS-Warteschlangen](#)
- [Aktualisieren von Amazon SQS-Warteschlangen](#)
- [Löschen von Amazon SQS-Warteschlangen](#)
- [Senden von Amazon SQS-Nachrichten](#)
- [Empfangen von Amazon SQS-Nachrichten](#)

Erstellen von Amazon SQS-Warteschlangen

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um eine Amazon SQS-Warteschlange zu erstellen. Die Anwendung erstellt eine [Warteschlange für unzustellbare Nachrichten](#), wenn Sie den ARN für keinen angeben. Anschließend wird eine Standard-Nachrichtenwarteschlange erstellt, die eine Warteschlange für unzustellbare Nachrichten enthält (die von Ihnen angegebene oder die erstellte).

Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach Informationen zu allen vorhandenen Warteschlangen an.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Anzeigen vorhandener Warteschlangen](#)
- [Erstellen der Warteschlange](#)
- [Abrufen des ARN einer Warteschlange](#)

- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Anzeigen vorhandener Warteschlangen

Der folgende Codeausschnitt zeigt eine Liste der vorhandenen Warteschlangen in der Region des SQS-Clients und die Attribute jeder Warteschlange.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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 der Warteschlange

Der folgende Ausschnitt erstellt eine Warteschlange. Der Ausschnitt beinhaltet die Verwendung einer Warteschlange für unzustellbare Nachrichten, aber für Ihre Warteschlangen ist nicht unbedingt eine Warteschlange für unzustellbare Nachrichten erforderlich.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen Ausschnitt.

```

//
// 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;
}

```

Abrufen des ARN einer Warteschlange

Der folgende Codeausschnitt ruft den ARN der Warteschlange ab, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt dieses verwendete Snippet.

```

//
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.SQS](#)

Programmiererelemente:

- Namespace [Amazon.SQS](#)
 - Klasse [AmazonSQSClient](#)
 - 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
```



```
{
  // = = = = =
  // = = =
  // Class to create a queue
  class Program
  {
    private const string MaxReceiveCount = "10";
    private const string ReceiveMessageWaitTime = "2";
    private const int MaxArgs = 3;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count > MaxArgs)
        CommandLine.ErrorExit(
          "\nToo many command-line arguments.\nRun the command with no arguments to see
help.");

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of no command-line arguments, just show help and the existing
queues
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ShowQueues(sqsClient);
        return;
      }

      // Get the application arguments from the parsed list
      string queueName =
        CommandLine.GetArgument(parsedArgs, null, "-q", "--queue-name");
      string deadLetterQueueUrl =
        CommandLine.GetArgument(parsedArgs, null, "-d", "--dead-letter-queue");
      string maxReceiveCount =
        CommandLine.GetArgument(parsedArgs, MaxReceiveCount, "-m", "--max-receive-
count");
    }
  }
}
```

```
string receiveWaitTime =
    CommandLine.GetArgument(parsedArgs, ReceiveMessageWaitTime, "-w", "--wait-
time");

if(string.IsNullOrEmpty(queueName))
    CommandLine.ErrorExit(
        "\nYou must supply a queue name.\nRun the command with no arguments to see
help.");

// If a dead-letter queue wasn't given, create one
if(string.IsNullOrEmpty(deadLetterQueueUrl))
{
    Console.WriteLine("\nNo dead-letter queue was specified. Creating one...");
    deadLetterQueueUrl = await CreateQueue(sqsClient, queueName + "__dlq");
    Console.WriteLine($"Your new dead-letter queue:");
    await ShowAllAttributes(sqsClient, deadLetterQueueUrl);
}

// Create the message queue
string messageQueueUrl = await CreateQueue(
    sqsClient, queueName, deadLetterQueueUrl, maxReceiveCount, receiveWaitTime);
Console.WriteLine($"Your new message queue:");
await ShowAllAttributes(sqsClient, messageQueueUrl);
}

//
// Method to show a list of the existing queues
private static async Task ShowQueues(IAmazonSQS sqsClient)
{
    ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
    Console.WriteLine();
    foreach(string qUrl in responseList.QueueUrls)
    {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
    }
}

//
// Method to create a queue. Returns the queue URL.
private static async Task<string> CreateQueue(
    IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
```

```
    string maxReceiveCount=null, string receiveWaitTime=null)
{
    var attrs = new Dictionary<string, string>();

    // If a dead-letter queue is given, create a message queue
    if(!string.IsNullOrEmpty(deadLetterQueueUrl))
    {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
            $"{{\"deadLetterTargetArn\": \"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\", \" +
            $\"\"maxReceiveCount\": \"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
    }

    // If no dead-letter queue is given, create one of those instead
    //else
    //{
    // // Add attributes for the dead-letter queue as needed
    // attrs.Add();
    //}

    // Create the queue
    CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
        new CreateQueueRequest{QueueName = qName, Attributes = attrs});
    return responseCreate.QueueUrl;
}

//
// Method to get the ARN of a queue
private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
    qUrl, new List<string>{QueueAttributeName.QueueArn});
    return responseGetAtt.QueueARN;
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
```

```

    var attributes = new List<string>{ QueueAttributeName.All };
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
        "\\n -q, --queue-name: The name of the queue you want to create." +
        "\\n -d, --dead-letter-queue: The URL of an existing queue to be used as the
dead-letter queue."+
        "\\n      If this argument isn't supplied, a new dead-letter queue will be
created." +
        "\\n -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
of the queue." +
        $"\\n      Default is {MaxReceiveCount}." +
        "\\n -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
for long polling." +
        $"\\n      Default is {ReceiveMessageWaitTime}.");
}
}

// = = = = =
//
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //

```

```
// Returns:
// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
```

```
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}
```

Weitere Überlegungen

- Ihr Warteschlangenname muss aus alphanumerischen Zeichen, Bindestrichen und Unterstrichen bestehen.
- Bei Warteschlangennamen und Warteschlangen-URLs wird zwischen Groß- und Kleinschreibung unterschieden
- Wenn Sie die Warteschlangen-URL benötigen, aber nur den Warteschlangennamen haben, verwenden Sie eine der `AmazonSQSClient.GetQueueUrlAsync` Methoden.
- Informationen zu den verschiedenen Warteschlangenattributen, die Sie festlegen können, finden Sie unter [CreateQueueRequest](#) in der [AWS SDK for .NET API-Referenz](#) zu oder [SetQueueAttributes](#) in der [API-Referenz zu Amazon Simple Queue Service](#).
- Dieses Beispiel gibt eine lange Abfrage für alle Nachrichten in der von Ihnen erstellten Warteschlange an. Dies geschieht mithilfe des `-ReceiveMessageWaitTimeSeconds` Attributs.

Sie können bei einem Aufruf der `ReceiveMessageAsync` Methoden der [AmazonSQSClient](#)-Klasse auch eine lange Abfrage angeben. Weitere Informationen finden Sie unter [Empfangen von Amazon SQS-Nachrichten](#).

Informationen zu Kurzabfragen im Vergleich zu Langabfragen finden Sie unter [Kurz- und Langabfragen](#) im Amazon Simple Queue Service-Entwicklerhandbuch.

- Eine Warteschlange für unzustellbare Nachrichten ist eine, auf die andere (Quell-)Warteschlangen Nachrichten abzielen können, die nicht erfolgreich verarbeitet wurden. Weitere Informationen finden Sie unter [Amazon SQS-Warteschlangen für unzustellbare Nachrichten](#) im Entwicklerhandbuch für Amazon Simple Queue Service.
- Sie können die Liste der Warteschlangen und die Ergebnisse dieses Beispiels auch in der [Amazon SQS-Konsole](#) anzeigen.

Aktualisieren von Amazon SQS-Warteschlangen

Dieses Beispiel zeigt Ihnen, wie Sie mit AWS SDK for .NET eine Amazon SQS-Warteschlange 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 Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Anzeigen von Warteschlangenattributen](#)
- [Attributname validieren](#)
- [Aktualisieren des Warteschlangenattributs](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Anzeigen von Warteschlangenattributen

Der folgende Codeausschnitt zeigt die Attribute der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

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

Attributname validieren

Der folgende Codeausschnitt validiert den Namen des Attributs, das aktualisiert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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;  
}
```

Aktualisieren des Warteschlangenattributs

Der folgende Codeausschnitt aktualisiert ein Attribut der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.SQS](#)

Programmirelemente:

- Namespace [Amazon.SQS](#)
 - Klasse [AmazonSQSClient](#)
 - 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  
{
```

```
// = = = = =
= = =
// Class to update a queue
class Program
{
    private const int MaxArgs = 3;
    private const int InvalidArgCount = 2;

    static async Task Main(string[] args)
    {
        // Parse the command line and show help if necessary
        var parsedArgs = CommandLine.Parse(args);
        if(parsedArgs.Count == 0)
        {
            PrintHelp();
            return;
        }
        if((parsedArgs.Count > MaxArgs) || (parsedArgs.Count == InvalidArgCount))
            CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
                "\nRun the command with no arguments to see help.");

        // Get the application arguments from the parsed list
        var qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
        var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
        var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

        if(string.IsNullOrEmpty(qUrl))
            CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
                "\nRun the command with no arguments to see help.");

        // Create the Amazon SQS client
        var sqsClient = new AmazonSQSClient();

        // In the case of one command-line argument, just show the attributes for the
queue
        if(parsedArgs.Count == 1)
            await ShowAllAttributes(sqsClient, qUrl);

        // Otherwise, attempt to update the given queue attribute with the given value
        else
        {
            // Check to see if the attribute is valid
            if(ValidAttribute(attribute))
            {
```

```
        // Perform the update and then show all the attributes of the queue
        await UpdateAttribute(sqsClient, qUrl, attribute, value);
        await ShowAllAttributes(sqsClient, qUrl);
    }
    else
    {
        Console.WriteLine($"\\nThe given attribute name, {attribute}, isn't valid.");
    }
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
            new List<string>{ QueueAttributeName.All });
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

//
// Method to check the name of the attribute
private static bool ValidAttribute(string attribute)
{
    var attOk = false;
    var qAttNameType = typeof(QueueAttributeName);
    List<string> qAttNamefields = new List<string>();
    foreach(var field in qAttNameType.GetFields())
        qAttNamefields.Add(field.Name);
    foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
    return attOk;
}

//
// Method to update a queue attribute
private static async Task UpdateAttribute(
    IAmazonSQS sqsClient, string qUrl, string attribute, string value)
```

```

    {
        await sqsClient.SetQueueAttributesAsync(qUrl,
            new Dictionary<string, string>{{attribute, value}});
    }

    //
    // Command-line help
    private static void PrintHelp()
    {
        Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
        Console.WriteLine("  -q: The URL of the queue you want to update.");
        Console.WriteLine("  -a: The name of the attribute to update.");
        Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)

```

```
{
    // If the first argument in this iteration starts with a dash it's an option.
    if(args[i].StartsWith("-"))
    {
        var key = args[i++];
        var value = key;

        // Check to see if there's a value that goes with this option?
        if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
        parsedArgs.Add(key, value);
    }

    // If the first argument in this iteration doesn't start with a dash, it's a
value
    else
    {
        parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
        n++;
    }
}

return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
```

```
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
```

Weitere Überlegungen

- Um das `RedrivePolicy` Attribut zu aktualisieren, müssen Sie den gesamten Wert in Anführungszeichen setzen und die Anführungszeichen für die Schlüssel-Wert-Paare mit Escape-Zeichen versehen, wie für Ihr Betriebssystem geeignet.

Unter Windows wird der Wert beispielsweise wie folgt erstellt:

```
"{\"deadLetterTargetArn\": \"DEAD_LETTER-QUEUE-ARN\", \"maxReceiveCount\": \"10\"}"
```

Löschen von Amazon SQS-Warteschlangen

Dieses Beispiel zeigt Ihnen, wie Sie mit AWS SDK for .NET eine Amazon SQS-Warteschlange löschen. Die Anwendung löscht die Warteschlange, wartet bis zu einer bestimmten Zeit, bis die Warteschlange beendet 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 Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Löschen der Warteschlange](#)
- [Warten Sie, bis die Warteschlange beendet ist](#)
- [Anzeigen einer Liste vorhandener Warteschlangen](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Löschen der Warteschlange

Der folgende Codeausschnitt löscht die durch die angegebene Warteschlangen-URL identifizierte Warteschlange.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 beendet ist

Der folgende Codeausschnitt wartet, bis der Löschvorgang abgeschlossen ist, was 60 Sekunden dauern kann.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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;
}
}
```

Anzeigen einer 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 diesen verwendeten Ausschnitt.

```
//
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.SQS](#)

Programmierelemente:

- Namespace [Amazon.SQS](#)

Klasse [AmazonSQSClient](#)

- 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
{
    // = = = = =
    // Class to update a queue
    class Program
    {
        private const int TimeToWait = 60;

        static async Task Main(string[] args)
        {
            // Create the Amazon SQS client
            var sqsClient = new AmazonSQSClient();

            // If no command-line arguments, just show a list of the queues
            if(args.Length == 0)
            {
                Console.WriteLine("\nUsage: SQSCreateQueue queue_url");
                Console.WriteLine("  queue_url - The URL of the queue you want to delete.");
                Console.WriteLine("\nNo arguments specified.");
                Console.Write("Do you want to see a list of the existing queues? ((y) or n):");
                var response = Console.ReadLine();
                if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
                    await ListQueues(sqsClient);
                return;
            }

            // If given a queue URL, delete that queue
            if(args[0].StartsWith("https://sqs."))
            {
```

```
// Delete the queue
await DeleteQueue(sqsClient, args[0]);
// Wait for a little while because it takes a while for the queue to disappear
await Wait(sqsClient, TimeToWait, args[0]);
// Show a list of the remaining queues
await ListQueues(sqsClient);
}
else
{
    Console.WriteLine("The command-line argument isn't a queue URL:");
    Console.WriteLine($"{args[0]}");
}
}

//
// 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 prüft nicht, ob die Warteschlange, die Sie löschen, als Warteschlange für unzustellbare Nachrichten verwendet wird. Ein komplexeres Verfahren könnte dies überprüfen.
- Sie können die Liste der Warteschlangen und die Ergebnisse dieses Beispiels auch in der [Amazon SQS-Konsole](#) anzeigen.

Senden von Amazon SQS-Nachrichten

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden AWS SDK for .NET , um Nachrichten an eine Amazon SQS-Warteschlange zu senden, die Sie [programmgesteuert](#) oder mithilfe der [Amazon SQS-Konsole](#) erstellen können. Die Anwendung sendet eine einzelne Nachricht an die Warteschlange und dann 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, oder um eine Anforderung zum Beenden der Anwendung.

Dieses Beispiel und das [nächste Beispiel für den Empfang von Nachrichten](#) können zusammen verwendet werden, um den Nachrichtenfluss in Amazon SQS anzuzeigen.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Senden einer Nachricht](#)
- [Senden eines Nachrichtenstapels](#)
- [Löschen aller Nachrichten aus der Warteschlange](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Senden einer Nachricht

Der folgende Codeausschnitt sendet eine Nachricht an die Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//  
// 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 eines Nachrichtenstapels

Der folgende Codeausschnitt sendet einen Stapel von Nachrichten an die Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
```

```
// 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.");
}
```

Löschen aller Nachrichten aus der Warteschlange

Der folgende Codeausschnitt löscht alle Nachrichten aus der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert werden. Dies wird auch als Bereinigen der Warteschlange bezeichnet.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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 zeigt relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.SQS](#)

Programmiererelemente:

- Namespace [Amazon.SQS](#)

Klasse [AmazonSQSClient](#)

- 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
{
    // = = = = =
    // Class to send messages to a queue
    class Program
    {
        // Some example messages to send to the queue
        private const string JsonMessage = "{ \"product\": [ { \"name\": \"Product A\", \"price\": \"32\" }, { \"name\": \"Product B\", \"price\": \"27\" } ] }";
        private const string XmlMessage = "<products><product name= \"Product A\" price= \"32\" /><product name= \"Product B\" price= \"27\" /></products>";
        private const string CustomMessage = "||product|Product A|32||product|Product B|27||";
        private const string TextMessage = "Just a plain text message.";

        static async Task Main(string[] args)
        {
            // Do some checks on the command-line
        }
    }
}
```

```
if(args.Length == 0)
{
    Console.WriteLine("\nUsage: SQSSendMessages 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)
// 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 Einschränkungen für Nachrichten, einschließlich der zulässigen Zeichen, finden Sie unter [Kontingente im Zusammenhang mit Nachrichten](#) im [Amazon Simple Queue Service-Entwicklerhandbuch](#).
- Nachrichten verbleiben in Warteschlangen, bis sie gelöscht oder die Warteschlange bereinigt 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 von Amazon SQS-Nachrichten

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK for .NET um Nachrichten aus einer Amazon SQS-Warteschlange zu empfangen, die Sie [programmgesteuert](#) oder mithilfe der [Amazon SQS-Konsole](#) erstellen können. Die Anwendung liest eine einzelne Nachricht aus der Warteschlange, verarbeitet die Nachricht (in diesem Fall zeigt den Nachrichtentext in 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 anzuzeigen.

Die folgenden Abschnitte enthalten Ausschnitte aus diesem Beispiel. Der [vollständige Code für das Beispiel](#) wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- [Empfangen einer Nachricht](#)

- [Löschen einer Nachricht](#)
- [Vollständiger Code](#)
- [Weitere Überlegungen](#)

Empfangen einer Nachricht

Der folgende Codeausschnitt empfängt eine Nachricht von der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// 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.)
    });
}
```

Löschen einer Nachricht

Der folgende Codeausschnitt löscht eine Nachricht aus der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel [am Ende dieses Themas](#) zeigt diesen verwendeten Ausschnitt.

```
//
// Method to delete a message from a queue
private static async Task DeleteMessage(
    IAmazonSQS sqsClient, Message message, string qUrl)
{
    Console.WriteLine($"Deleting message {message.MessageId} from queue...");
    await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
}
```

Vollständiger Code

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

SDK-Referenzen

NuGet -Pakete:

- [AWSSDK.SQS](#)

Programmiererelemente:

- Namespace [Amazon.SQS](#)
Klasse [AmazonSQSClient](#)
- 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 lange Abfragen anzugeben, verwendet dieses Beispiel die `-WaitTimeSeconds`Eigenschaft für jeden Aufruf der `-ReceiveMessageAsync`Methode.

Sie können auch Langabfragen für alle Nachrichten in einer Warteschlange angeben, indem Sie das `-ReceiveMessageWaitTimeSeconds`Attribut beim [Erstellen](#) oder [Aktualisieren](#) der Warteschlange verwenden.

Informationen zu Kurzabfragen im Vergleich zu Langabfragen finden Sie unter [Kurz- und Langabfragen](#) im Amazon Simple Queue Service-Entwicklerhandbuch.

- Während der Nachrichtenverarbeitung können Sie das Empfangs-Handle verwenden, um das Timeout für die Sichtbarkeit der Nachricht zu ändern. Informationen dazu finden Sie in den `ChangeMessageVisibilityAsync` Methoden der [AmazonSQSClient](#)-Klasse.
- Wenn Sie die `DeleteMessageAsync` Methode bedingungslos aufrufen, wird die Nachricht unabhängig von der Zeitüberschreitungseinstellung für die Sichtbarkeit aus der Warteschlange entfernt.

AWS LambdaFür den Rechendienst verwenden

Die AWS SDK for .NET UnterstützungAWS Lambda, mit der Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Weitere Informationen finden Sie auf der [AWS LambdaProduktseite](#) und im [AWS LambdaEntwicklerhandbuch](#), insbesondere im Abschnitt [Arbeiten mit C#](#).

APIs

Das AWS SDK for .NET bietet APIs fürAWS Lambda. Die APIs ermöglichen es Ihnen, mit Lambda-Funktionen wie [Funktionen](#), [Triggern](#) und [Ereignissen](#) zu arbeiten. Den vollständigen Satz von APIs finden Sie unter [Lambda](#) in der [AWS SDK for .NETAPI-Referenz](#).

Die Lambda-APIs werden in [NuGetPaketen](#) bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie [Ihre Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

Themen

- [Anmerkungen zum Schreiben verwendenAWS LambdaFunktionen](#)

Anmerkungen zum Schreiben verwendenAWS LambdaFunktionen

Beim Schreiben von Lambda-Funktionen müssen Sie manchmal eine große Menge an Handler-Code schreiben und aktualisierenAWS CloudFormationVorlagen, unter anderem Aufgaben. Lambda Annotations ist ein Framework, das dazu beitragen soll, diese Belastungen von.NET 6-Lambda-Funktionen zu verringern, wodurch sich das Schreiben von Lambda in C# natürlicher anfühlt.

Als Beispiel für die Vorteile der Verwendung des Lambda Annotations Frameworks betrachten wir die folgenden Codefragmente, die zwei Zahlen addieren.

Ohne Lambda-Annotationen

```
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-Annotationen

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Wie im Beispiel gezeigt, kann Lambda Annotations die Notwendigkeit eines bestimmten Boilerplattencodes überflüssig machen.

Einzelheiten zur Verwendung des Frameworks sowie zusätzliche Informationen finden Sie in den folgenden Ressourcen:

- Der [GitHub LIES MICH](#) für die Dokumentation zu den APIs und Attributen von Lambda Annotations.
- Der [Blogbeitrag](#) für Lambda-Annotationen.
- Der [Amazon.Lambda.Annotations](#) NuGet-Paket.
- Der [Foto-Asset-Management-Projekt](#) auf GitHub. Konkret finden Sie unter [PamApiAnnotations](#) Ordner und Verweise auf Lambda Annotations im Projekt [LIES MICH](#).

Allgemeine Bibliotheken und Frameworks für die AWS SDK for .NET

Die folgenden Abschnitte enthalten Informationen zu High-Level-Bibliotheken und Frameworks, die nicht Teil der Kern-SDK-Funktionalität sind. Diese Bibliotheken und Frameworks verwenden die Kern-SDK-Funktionalität, um Funktionen zu erstellen, die bestimmte Aufgaben vereinfachen.

Wenn Sie noch nicht mit dem vertraut sind [AWS SDK for .NET](#), sollten Sie sich zuerst das [Machen Sie einen kurzen Rundgang](#) Thema ansehen. Es bietet Ihnen eine Einführung in das SDK.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

- [AWS Framework zur Nachrichtenverarbeitung für .NET](#)

AWS Framework zur Nachrichtenverarbeitung für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Das AWS Message Processing Framework für .NET ist ein AWS-natives Framework, das die Entwicklung von .NET-Nachrichtenverarbeitungsanwendungen vereinfacht, die AWS Services wie

Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) und Amazon verwenden EventBridge. Das Framework reduziert die Menge von Boiler-plate-Code, die Entwickler schreiben müssen, sodass Sie sich beim Veröffentlichen und Verbrauchen von Nachrichten auf Ihre Geschäftslogik konzentrieren können.

Allgemeine Informationen zum Message Processing Framework und dessen Verwendung finden Sie in der README Datei im GitHub Repo unter <https://github.com/aws-labs/aws-dotnet-messaging>.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Themen

- [Erste Schritte mit dem AWS Message Processing Framework für .NET](#)
- [Veröffentlichen von Nachrichten mit dem AWS Message Processing Framework für .NET](#)
- [Verwenden von Nachrichten mit dem AWS Message Processing Framework für .NET](#)
- [Erfassen von Telemetriedaten für das AWS Message Processing Framework für .NET](#)
- [Anpassen des AWS Nachrichtenverarbeitungs-Frameworks für .NET](#)
- [Sicherheit für das AWS Message Processing Framework für .NET](#)

Erste Schritte mit dem AWS Message Processing Framework für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre [Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Um das AWS Message Processing Framework für .NET zu verwenden, müssen Sie das [-AWS.Messaging](#) NuGet Paket zu Ihrem Projekt hinzufügen. Beispielsweise:

```
dotnet add package AWS.Messaging --prerelease
```

Das Framework ist in den [DI-Servicecontainer \(Dependency Injection\) von .NET](#) integriert. Sie können das Framework beim Start Ihrer Anwendung konfigurieren, indem Sie `aufufenAddAWSMessageBus`, 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");
});
```

Veröffentlichen von Nachrichten mit dem AWS Message Processing Framework für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Das AWS Message Processing Framework für .NET unterstützt die Veröffentlichung eines oder mehrerer Nachrichtentypen, die Verarbeitung eines oder mehrerer Nachrichtentypen oder die Ausführung beider Nachrichtentypen in derselben Anwendung.

Der folgende Code zeigt eine Konfiguration für eine Anwendung, die verschiedene Nachrichtentypen für verschiedene AWS Services veröffentlicht.

```
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");

    // 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");
```

```
});
```

Sobald Sie das Framework beim Start registriert haben, fügen Sie das generische `IMessagePublisher` in Ihren Code ein. Rufen Sie die `PublishAsync` Methode auf, um einen der oben konfigurierten Nachrichtentypen zu veröffentlichen. Der generische Herausgeber bestimmt basierend auf seinem Typ das Ziel, an das die Nachricht weitergeleitet werden soll.

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.");
        }

        // Publish 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();
}
}
```

Servicespezifische Publisher

Im obigen Beispiel wird die generische verwendet `IMessagePublisher`, die basierend auf dem konfigurierten Nachrichtentyp in jedem unterstützten AWS Service veröffentlichen kann. Das Framework bietet auch servicespezifische Herausgeber für Amazon SQS , Amazon SNS und Amazon EventBridge. Diese spezifischen Publisher stellen Optionen bereit, die nur für diesen Service gelten und mit den Typen `ISQSPublisher`, `ISNSPublisher` und `IEventBridgePublisher` eingefügt werden können.

Wenn Sie beispielsweise Nachrichten in einer SQS FIFO-Warteschlange veröffentlichen, müssen Sie die entsprechende [Nachrichtengruppen-ID](#) festlegen. Der folgende Code zeigt das `ChatMessage` Beispiel erneut, verwendet jetzt jedoch einen `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();
}
}
```

Dies kann für SNS und EventBridge mit `ISNSPublisher` `IEventBridgePublisher` bzw. erfolgen.

```
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>
```

```
});
```

Verwenden von Nachrichten mit dem AWS Message Processing Framework für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Um Nachrichten zu verarbeiten, implementieren Sie einen Nachrichtenhandler über die `IMessageHandler` Schnittstelle für jeden Nachrichtentyp, den Sie verarbeiten möchten. Die Zuordnung zwischen Nachrichtentypen und Nachrichtenhandlern wird im Projektstartup 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());
}
}
```

Die äußere `MessageEnvelope` enthält Metadaten, die vom Framework verwendet werden. Ihre `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-Warteschlange löscht. Bei der Rückgabe von `MessageProcessStatus.Failed()`, in der sie erneut verarbeitet oder in eine [Warteschlange für unzustellbare Nachrichten](#) verschoben werden kann, falls konfiguriert.

Umgang mit Nachrichten in einem Prozess mit langer Laufzeit

Sie können `AddSQSPoller` mit einer SQS-Warteschlangen-URL aufrufen, um eine lange Laufzeit zu starten [BackgroundService](#), die die Warteschlange kontinuierlich 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.MaximumNumberOfConcurrentMessages = 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();
```

Konfigurieren des SQS Message Poller

Der SQS-Nachrichten-Poller kann vom `SQSMessagePollerOptions` beim Aufruf von `AddSQSPoller` konfiguriert werden.

- `MaximumNumberOfConcurrentMessages` – 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, gibt der Aufruf früher als `WaitTimeSeconds` zurück. Der Standardwert ist 20.

Umgang mit Timeout bei der Sichtbarkeit von Nachrichten

SQS-Nachrichten haben eine [Zeitbeschränkung für die Sichtbarkeit](#). Wenn ein Konsument mit der Verarbeitung einer bestimmten Nachricht beginnt, verbleibt sie in der Warteschlange, wird aber von anderen Konsumenten ausgeblendet, um zu vermeiden, dass sie mehr als einmal verarbeitet wird. Wenn die Nachricht nicht verarbeitet und gelöscht wird, bevor sie wieder sichtbar wird, versucht ein anderer Verbraucher möglicherweise, dieselbe Nachricht zu verarbeiten.

Das Framework verfolgt das Sichtbarkeits-Timeout für Nachrichten, die es derzeit verarbeitet, und versucht, es zu verlängern. Sie können dieses Verhalten `SQSMessagePollerOptions` beim Aufrufen von `builder.AddSQSPoller` auf dem konfigurieren.

- `VisibilityTimeout` – Die Dauer in Sekunden, für die empfangene Nachrichten von nachfolgenden Abrufanforderungen ausgeblendet werden. Der Standardwert lautet 30.
- `VisibilityTimeoutExtensionThreshold` – Wenn das Sichtbarkeits-Timeout einer Nachricht innerhalb so vieler Sekunden abläuft, verlängert das Framework das Sichtbarkeits-Timeout (um weitere `VisibilityTimeout` Sekunden). Der Standardwert ist 5.
- `VisibilityTimeoutExtensionHeartbeatInterval` – Wie oft in Sekunden das Framework nach Nachrichten sucht, die innerhalb von `VisibilityTimeoutExtensionThreshold` Sekunden nach Ablauf ablaufen, und dann deren Sichtbarkeits-Timeout verlängert. Der Standardwert lautet 1.

Im folgenden Beispiel prüft das Framework alle 1 Sekunde auf Nachrichten, die noch verarbeitet werden. Für diese Nachrichten verlängert das Framework das Sichtbarkeits-Timeout jeder Nachricht innerhalb von 5 Sekunden, nachdem sie wieder sichtbar wurden, automatisch 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;
        options.VisibilityTimeoutExtensionHeartbeatInterval = 1;
    });
```

Umgang mit Nachrichten in - AWS Lambda Funktionen

Sie können das AWS Message Processing Framework für .NET mit der [Integration von SQS in Lambda](#) verwenden. Dies wird vom `-AWS.Messaging.LambdaPaket` bereitgestellt. Weitere Informationen finden Sie unter [README](#).

Erfassen von Telemetriedaten für das AWS Message Processing Framework für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Das AWS Message Processing Framework für .NET ist für instrumentiert, OpenTelemetry um [Ablaufverfolgungen](#) für jede Nachricht zu protokollieren, die vom Framework veröffentlicht oder verarbeitet wird. Dies wird vom `-AWS.Messaging.Telemetry.OpenTelemetryPaket` bereitgestellt. Weitere Informationen finden Sie unter [README](#).

Anpassen des AWS Nachrichtenverarbeitungs-Frameworks für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

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 Service spezifisch ist. Mit Amazon SQS erstellt es beispielsweise [SendMessage](#) Anforderungen und arbeitet mit den [Message](#) Objekten, die vom Service definiert werden.
2. Innerhalb der SQS-Anforderung und -Antwort legt das Framework das `-MessageBodyElement` (oder `Message` für Amazon SNS oder `Detail` für Amazon EventBridge) auf ein [JSON-formatiertes CloudEvent](#) fest. Dies enthält Metadaten, die vom Framework festgelegt wurden und auf dem `MessageEnvelope` Objekt verfügbar sind, wenn eine Nachricht verarbeitet wird.
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 sie auf eine neue GUID festgelegt, aber dies kann überschrieben werden, indem Sie Ihre eigene implementieren `IMessageIdGenerator` und diese in den DI-Container einfügen.

- "type" steuert, wie die Nachricht an Handler weitergeleitet wird. Standardmäßig wird der vollständige Name des .NET-Typs verwendet, der der Nachricht entspricht. Sie können dies über den `messageTypeIdentifizier` 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 Aufgaben-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 der `überschreibenMessageBusBuilder`.
- "time" wird `DateTime` in UTC auf den aktuellen festgelegt. Dies kann überschrieben werden, indem Sie Ihren eigenen implementieren `IDateTimeHandler` und diesen in den DI-Container einfügen.
- "data" enthält eine JSON-Darstellung des .NET-Objekts, das als Nachricht gesendet oder empfangen wurde:
 - `ConfigureSerializationOptions` Mit in `MessageBusBuilder` können Sie die konfigurieren [System.Text.Json.JsonSerializerOptions](#), die 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 ihn über `AddSerializationCallback` auf implementieren `ISerializationCallback` und registrieren `MessageBusBuilder`.

Sicherheit für das AWS Message Processing Framework für .NET

Dies ist die Vorabdokumentation für ein Feature in der Vorschauversion. Änderungen sind vorbehalten.

Das AWS Message Processing Framework für .NET nutzt AWS SDK for .NET für die Kommunikation mit AWS. Weitere Informationen zur Sicherheit in der finden Sie AWS SDK for .NET unter [Sicherheit für dieses AWS Produkt oder diesen Service](#).

Aus Sicherheitsgründen protokolliert das Framework keine vom Benutzer gesendeten Datennachrichten. Wenn Sie diese Funktionalität für Debugging-Zwecke aktivieren möchten, müssen Sie `EnableDataMessageLogging()` im Message Bus wie folgt aufrufen:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Wenn Sie ein potenzielles Sicherheitsproblem feststellen, finden Sie Informationen zur Meldung in der [Sicherheitsrichtlinie](#).

Programmierung AWS OpsWorks für die Arbeit mit Stacks und Anwendungen

Warning

AWS OpsWorkserreicht das Ende der Nutzungsdauer und akzeptiert keine neuen Kunden. Bestehende Kunden sind bis März oder Mai 2024 nicht betroffen, je nachdem, welche Dienste sie nutzen, und zu welchem Zeitpunkt der Dienst nicht mehr verfügbar sein wird. Um sich auf diesen Übergang vorzubereiten, empfehlen wir Bestandskunden, so bald wie möglich auf andere Lösungen umzusteigen. Weitere Informationen finden Sie auf der [OpsWorks - Produktseite](#).

Das AWS SDK for .NET unterstützt AWS OpsWorks, das eine einfache und flexible Möglichkeit zum Erstellen und Verwalten von Stacks und Anwendungen bietet. Mit AWS OpsWorks können Sie AWS Ressourcen bereitstellen, ihre Konfiguration verwalten, Anwendungen für diese Ressourcen bereitstellen und ihren Zustand überwachen. Weitere Informationen finden Sie auf der [OpsWorks Produktseite](#) und im [AWS OpsWorksBenutzerhandbuch](#).

APIs

Das AWS SDK for .NET bietet APIs fürAWS OpsWorks. Die APIs ermöglichen es Ihnen, mit AWS OpsWorks Funktionen wie [Stacks](#) mit ihren [Ebenen](#), [Instanzen](#) und [Apps](#) zu arbeiten. Den vollständigen Satz von APIs finden Sie in der [AWS SDK for .NETAPI-Referenz](#) (und scrollen Sie zu „Amazon. OpsWorks,,).

Die AWS OpsWorks APIs werden von der bereitgestellt [AWSSDK. OpsWorks](#) NuGet Paket.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie [Ihre Umgebung und Ihr Projekt eingerichtet](#) haben. Lesen Sie auch die Informationen unter [SDK-Funktionen](#).

Support für SonstigeAWS-Services und Konfiguration

DieAWS SDK for .NETunterstütztAWS-Services zusätzlich zu den zuvor beschriebenen. Weitere Information zu den APIs für alle unterstützten Services finden Sie in der [AWS SDK for .NET-API-Referenz](#)aus.

Neben den Namespaces für EinzelpersonenAWS-Services, dasAWS SDK for .NETBietet auch die folgenden APIs:

Flächen	Beschreibung	Ressourcen
AWS-Unterstützung	Programmgesteuerter Zugriff aufAWSSupport-Fälle und Trusted Advisor Advisor-Funktionen	Weitere Informationen finden Sie unter Amazon.AWSSupport und Amazon.AWSSupport.Model .
Allgemeines	Helfer-Klassen und Aufzählungen.	Weitere Informationen finden Sie unter Amazon und Amazon.Util .

AWS SDK for .NET Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie mit verwenden AWS SDK for .NET AWS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Serviceübergreifende Beispiele sind Beispielanwendungen, die über mehrere AWS-Services hinweg arbeiten.

Beispiele

- [Aktionen und Szenarien mit AWS SDK for .NET](#)
- [Serviceübergreifende Beispiele mit AWS SDK for .NET](#)

Aktionen und Szenarien mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS-Services.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Services

- [ACM-Beispiele mit AWS SDK for .NET](#)
- [Aurora-Beispiele mit AWS SDK for .NET](#)
- [Auto Scaling-Beispiele mit AWS SDK for .NET](#)
- [Amazon-Bedrock-Beispiele mit AWS SDK for .NET](#)
- [Amazon-Bedrock-Laufzeitbeispiele mit AWS SDK for .NET](#)
- [AWS CloudFormation -Beispiele mit AWS SDK for .NET](#)

- [CloudWatch -Beispiele mit AWS SDK for .NET](#)
- [CloudWatch Protokolliert Beispiele mit AWS SDK for .NET](#)
- [Beispiele für Amazon Cognito-Identitätsanbieter mit AWS SDK for .NET](#)
- [Amazon Comprehend-Beispiele mit AWS SDK for .NET](#)
- [DynamoDB-Beispiele mit AWS SDK for .NET](#)
- [Amazon EC2-Beispiele mit AWS SDK for .NET](#)
- [Amazon-ECS-Beispiele mit AWS SDK for .NET](#)
- [Beispiele für Elastic Load Balancing mit AWS SDK for .NET](#)
- [EventBridge -Beispiele mit AWS SDK for .NET](#)
- [AWS Glue -Beispiele mit AWS SDK for .NET](#)
- [IAM-Beispiele mit AWS SDK for .NET](#)
- [Beispiele für Amazon Keyspaces mit AWS SDK for .NET](#)
- [Kinesis-Beispiele mit AWS SDK for .NET](#)
- [AWS KMS -Beispiele mit AWS SDK for .NET](#)
- [Lambda-Beispiele mit AWS SDK for .NET](#)
- [MediaConvert -Beispiele mit AWS SDK for .NET](#)
- [Beispiele für Organisationen mit AWS SDK for .NET](#)
- [Amazon Polly-Beispiele mit AWS SDK for .NET](#)
- [Amazon-RDS-Beispiele mit AWS SDK for .NET](#)
- [Amazon Rekognition-Beispiele mit AWS SDK for .NET](#)
- [Beispiele für die Route-53-Domainregistrierung mit AWS SDK for .NET](#)
- [Amazon S3-Beispiele mit AWS SDK for .NET](#)
- [S3-Glacier-Beispiele mit AWS SDK for .NET](#)
- [SageMaker -Beispiele mit AWS SDK for .NET](#)
- [Secrets-Manager-Beispiele mit AWS SDK for .NET](#)
- [Amazon SES-Beispiele mit AWS SDK for .NET](#)
- [Amazon SNS-Beispiele mit AWS SDK for .NET](#)
- [Amazon SQS-Beispiele mit AWS SDK for .NET](#)
- [Step-Functions-Beispiele mit AWS SDK for .NET](#)
- [AWS STS -Beispiele mit AWS SDK for .NET](#)

- [AWS Support -Beispiele mit AWS SDK for .NET](#)
- [Amazon TranscribeBeispiele mit AWS SDK for .NET](#)
- [Amazon Translate-Beispiele mit AWS SDK for .NET](#)

ACM-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit ACM verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Beschreiben eines Zertifikats

Das folgende Codebeispiel zeigt, wie Sie ACM-Zertifikate beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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



```
using Amazon.CertificateManager;
using Amazon.CertificateManager.Model;

namespace DescribeCertificate
{
    class DescribeCertificate
    {
        // The following example retrieves and displays the metadata for a
        // certificate using the AWS Certificate Manager (ACM) service.

        // Specify your AWS Region (an example Region is shown).
        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new
Amazon.CertificateManager.AmazonCertificateManagerClient(ACMRegion);

            var describeCertificateReq = new DescribeCertificateRequest();
            // The ARN used here is just an example. Replace it with the ARN of
            // a certificate that exists on your account.
            describeCertificateReq.CertificateArn =
                "arn:aws:acm:us-
east-1:123456789012:certificate/8cfd7dae-9b6a-2d07-92bc-1c309EXAMPLE";

            var certificateDetailResp =
                DescribeCertificateResponseAsync(client: _client, request:
describeCertificateReq);
            var certificateDetail = certificateDetailResp.Result.Certificate;

            if (certificateDetail is not null)
            {
                DisplayCertificateDetails(certificateDetail);
            }
        }

        /// <summary>
        /// Displays detailed metadata about a certificate retrieved
        /// using the ACM service.
        /// </summary>
        /// <param name="certificateDetail">The object that contains details
        /// returned from the call to DescribeCertificateAsync.</param>
        static void DisplayCertificateDetails(CertificateDetail certificateDetail)
```

```
    {
        Console.WriteLine("\nCertificate Details: ");
        Console.WriteLine($"Certificate Domain:
{certificateDetail.DomainName}");
        Console.WriteLine($"Certificate Arn:
{certificateDetail.CertificateArn}");
        Console.WriteLine($"Certificate Subject: {certificateDetail.Subject}");
        Console.WriteLine($"Certificate Status: {certificateDetail.Status}");
        foreach (var san in certificateDetail.SubjectAlternativeNames)
        {
            Console.WriteLine($"Certificate SubjectAlternativeName: {san}");
        }
    }

    /// <summary>
    /// Retrieves the metadata associated with the ACM service certificate.
    /// </summary>
    /// <param name="client">An AmazonCertificateManagerClient object
    /// used to call DescribeCertificateResponse.</param>
    /// <param name="request">The DescribeCertificateRequest object that
    /// will be passed to the method call.</param>
    /// <returns></returns>
    static async Task<DescribeCertificateResponse>
DescribeCertificateResponseAsync(
    AmazonCertificateManagerClient client, DescribeCertificateRequest
request)
    {
        var response = new DescribeCertificateResponse();

        try
        {
            response = await client.DescribeCertificateAsync(request);
        }
        catch (InvalidArnException)
        {
            Console.WriteLine($"Error: The ARN specified is invalid.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Error: The specified certificate could not be
found.");
        }

        return response;
    }
}
```

```
    }  
  }  
}
```

- Weitere API-Informationen finden Sie unter [DescribeCertificate](#) in der APIAWS SDK for .NET - Referenz für .

Zertifikate auflisten

Das folgende Codebeispiel zeigt, wie Sie ACM-Zertifikate auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Threading.Tasks;  
using Amazon;  
using Amazon.CertificateManager;  
using Amazon.CertificateManager.Model;  
  
namespace ListCertificates  
{  
    // The following example retrieves and displays a list of the  
    // certificates defined for the default account using the AWS  
    // Certificate Manager (ACM) service.  
    class ListCertificates  
    {  
        // Specify your AWS Region (an example Region is shown).  
  
        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;  
        private static AmazonCertificateManagerClient _client;  
  
        static void Main(string[] args)
```

```

    {
        _client = new AmazonCertificateManagerClient(ACMRegion);
        var certificateList = ListCertificatesResponseAsync(client: _client);

        Console.WriteLine("Certificate Summary List\n");

        foreach (var certificate in
certificateList.Result.CertificateSummaryList)
            {
                Console.WriteLine($"Certificate Domain: {certificate.DomainName}");
                Console.WriteLine($"Certificate ARN:
{certificate.CertificateArn}\n");
            }
        }

        /// <summary>
        /// Retrieves a list of the certificates defined in this Region.
        /// </summary>
        /// <param name="client">The ACM client object passed to the
        /// ListCertificateResAsync method call.</param>
        /// <param name="request"></param>
        /// <returns>The ListCertificatesResponse.</returns>
        static async Task<ListCertificatesResponse> ListCertificatesResponseAsync(
            AmazonCertificateManagerClient client)
        {
            var request = new ListCertificatesRequest();

            var response = await client.ListCertificatesAsync(request);
            return response;
        }
    }
}

```

- Weitere API-Informationen finden Sie unter [ListCertificates](#) in der APIAWS SDK for .NET - Referenz für .

Aurora-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Aurora verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Aurora

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Aurora.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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($"Hello Amazon RDS Aurora! Let's list some clusters in
this account:");
    foreach (var cluster in response.DBClusters)
    {
        Console.WriteLine($"\\tCluster: database: {cluster.DatabaseName}
identifier: {cluster.DBClusterIdentifier}.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBClusters](#) in der API-Referenz zu AWS SDK for .NET .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

DB-Cluster erstellen

Das folgende Codebeispiel zeigt, wie Sie einen Aurora-DB-Cluster erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [CreateDBCluster](#) in der API-Referenz zu AWS SDK for .NET .

Erstellen einer DB-Cluster-Parametergruppe

Das folgende Codebeispiel zeigt, wie Sie eine Aurora-DB-Cluster-Parametergruppe erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBClusterParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines DB-Cluster-Snapshots

Das folgende Codebeispiel zeigt, wie Sie einen Aurora-DB-Cluster-Snapshot erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBClusterSnapshot](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen einer DB-Instance in einem DB-Cluster

Das folgende Codebeispiel zeigt, wie Sie eine DB-Instance in einem Aurora-DB-Cluster erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBInstance](#) in der AWS SDK for .NET -API-Referenz.

Löschen eines DB-Clusters

Das folgende Codebeispiel zeigt, wie Sie einen Aurora-DB-Cluster löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteDBCluster](#) in der API-Referenz zu AWS SDK for .NET .

Löschen einer DB-Cluster-Parametergruppe

Das folgende Codebeispiel zeigt, wie Sie eine Aurora-DB-Cluster-Parametergruppe löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteDBClusterParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer DB-Instance

Das folgende Codebeispiel zeigt, wie Sie eine Aurora-DB-Instance löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteDBInstance](#) in der API-Referenz zu AWS SDK for .NET .

Beschreiben von DB-Cluster-Parametergruppen

Das folgende Codebeispiel zeigt, wie Sie Aurora-DB-Cluster-Parametergruppen beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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();
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBClusterParameterGroups](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von DB-Cluster-Snapshots

Das folgende Codebeispiel zeigt, wie Sie Aurora-DB-Cluster-Snapshots beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBClusterSnapshots](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von DB-Clustern

Das folgende Codebeispiel zeigt, wie Sie Aurora-DB-Cluster beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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);
        results.AddRange(response.DBClusters);
        request.Marker = response.Marker;
    }
    while (response.Marker is not null);
    return results;
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBClusters](#) in der API-Referenz zu AWS SDK for .NET .

Beschreiben von DB-Instances

Das folgende Codebeispiel zeigt, wie Sie Aurora-DB-Instances beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBInstances](#) in der API-Referenz zu AWS SDK for .NET .

Beschreiben der Datenbank-Engine-Versionen

Das folgende Codebeispiel zeigt, wie Versionen der Aurora-Datenbank-Engine beschrieben werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBEngineVersions](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben der Optionen für DB-Instances

Das folgende Codebeispiel zeigt, wie Optionen für Aurora-DB-Instances beschrieben werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;  
}
```

- Weitere API-Informationen finden Sie unter [DescribeOrderableDBInstanceOptions](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von Parametern von einer DB-Cluster-Parametergruppe aus

Das folgende Codebeispiel zeigt, wie Parameter aus einer Aurora-DB-Cluster-Parametergruppe beschrieben werden.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [DescribeDBClusterParameters](#) in der APIAWS SDK for .NET -Referenz für .

Aktualisieren von Parametern in einer DB-Cluster-Parametergruppe

Das folgende Codebeispiel zeigt, wie Parameter in einer Aurora-DB-Cluster-Parametergruppe aktualisiert werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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 && 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;
}
```

- Weitere API-Informationen finden Sie unter [ModifyDBClusterParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit DB-Clustern

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.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung 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(
                $"{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}. Engine: {version.Engine} Version
{version.EngineVersion}.");
        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(
                $"{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(
        $"{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($"Clean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"Clean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"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.WriteLine(".");
            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.DeleteClusterParameterGroupNameAsync(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 && 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> DeleteClusterParameterGroupNameAsync(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);
        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 for .NET -API-Referenz.
 - [CreateDBCluster](#)
 - [CreateDBClusterParameterGroup](#)
 - [CreateDBClusterSnapshot](#)
 - [CreateDBInstance](#)
 - [DeleteDBCluster](#)
 - [DeleteDBClusterParameterGroup](#)
 - [DeleteDBInstance](#)
 - [DescribeDBClusterParameterGroups](#)
 - [DescribeDBClusterParameters](#)
 - [DescribeDBClusterSnapshots](#)
 - [DescribeDBClusters](#)
 - [DescribeDBEngineVersions](#)
 - [DescribeDBInstances](#)
 - [DescribeOrderableDBInstanceOptions](#)
 - [ModifyDBClusterParameterGroup](#)

Auto Scaling-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Auto Scaling verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Auto Scaling

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Auto Scaling.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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();

        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {
            Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.AvailabilityZone}");
        });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling groups.");
        }
    }
}
```

```
    }  
  }  
}
```

- Weitere API-Informationen finden Sie unter [DescribeAutoScalingGroups](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Anfügen einer ELB-Zielgruppe an eine Auto Scaling-Gruppe

Das folgende Codebeispiel zeigt, wie Sie eine ELB-Zielgruppe an eine Auto Scaling-Gruppe anfügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
    /// <summary>  
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto  
    Scaling group.  
    /// The  
    /// </summary>  
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</  
param>  
    /// <param name="targetGroupArn">The Arn for the target group.</param>  
    /// <returns>Async task.</returns>  
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string  
targetGroupArn)  
    {  
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(  

```



```

        new AttachLoadBalancerTargetGroupsRequest()
        {
            AutoScalingGroupName = autoScalingGroupName,
            TargetGroupARNs = new List<string>() { targetGroupArn }
        });
    }

```

- Weitere API-Informationen finden Sie unter [AttachLoadBalancerTargetGroups](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen einer Gruppe

Das folgende Codebeispiel zeigt, wie Sie eine Auto Scaling-Gruppe erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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
};

var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
Console.WriteLine($"{groupName} Auto Scaling Group created");
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [CreateAutoScalingGroup](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Gruppe

Das folgende Codebeispiel zeigt, wie Sie eine Auto Scaling-Gruppe löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Aktualisieren Sie die Mindestgröße einer Auto-Scaling-Gruppe auf Null, beenden Sie alle Instances in der Gruppe und löschen Sie die Gruppe.

```
/// <summary>
/// Try to terminate an instance by its Id.
```

```
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                });
            stopping = true;
        }
        catch (ScalingActivityInProgressException)
        {
            Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
```

```
        {
            AutoScalingGroupName = groupName
        });
        stopped = true;
    }
    catch (Exception e)
        when ((e is ScalingActivityInProgressException)
            || (e is Amazon.AutoScaling.Model.ResourceInUseException))
    {
        Console.WriteLine($"Some instances are still running. Waiting...");
        Thread.Sleep(10000);
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
    _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
        new DescribeAutoScalingGroupsRequest()
        {
            AutoScalingGroupNames = new List<string>() { groupName }
        });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
}
```

```
    }
    else
    {
        Console.WriteLine($"No groups found with name {groupName}.");
    }
}
```

```
/// <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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteAutoScalingGroup](#) in der APIAWS SDK for .NET -Referenz für .

Deaktivieren der Erfassung von Metriken für eine Gruppe

Das folgende Codebeispiel zeigt, wie Sie die Erfassung von CloudWatch Metriken für eine Auto Scaling-Gruppe deaktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DisableMetricsCollection](#) in der APIAWS SDK for .NET -Referenz für .

Aktivieren der Erfassung von Metriken für eine Gruppe

Das folgende Codebeispiel zeigt, wie Sie die Erfassung von CloudWatch Metriken für eine Auto Scaling-Gruppe aktivieren.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [EnableMetricsCollection](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Informationen zu Gruppen

Das folgende Codebeispiel zeigt, wie Sie Informationen zu Auto Scaling-Gruppen abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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>();
    groups!.ForEach(group =>
    {
        if (group.AutoScalingGroupName == groupName)
        {
            group.Instances.ForEach(instance =>
            {
                instanceIds.Add(instance.InstanceId);
            });
        }
    });

    var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
    {
        MaxRecords = 10,
        InstanceIds = instanceIds,
    };

    var response = await
_amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
    var instanceDetails = response.AutoScalingInstances;
```



```
        return instanceDetails;
    }
```

- Weitere API-Informationen finden Sie unter [DescribeAutoScalingGroups](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Informationen zu Instances

Das folgende Codebeispiel zeigt, wie Sie Informationen zu Auto Scaling-Instances abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        })
    }
```

```
});

var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
{
    MaxRecords = 10,
    InstanceIds = instanceIds,
};

var response = await
_amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
var instanceDetails = response.AutoScalingInstances;

return instanceDetails;
}
```

- Weitere API-Informationen finden Sie unter [DescribeAutoScalingInstances](#) in der APIAWS SDK for .NET -Referenz für .

Informationen zu Skalierungsaktivitäten abrufen

Das folgende Codebeispiel zeigt, wie Sie Informationen zu Auto Scaling-Aktivitäten abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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<Amazon.AutoScaling.Model.Activity>>
DescribeScalingActivitiesAsync(
```

```
string groupName)
{
    var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
    {
        AutoScalingGroupName = groupName,
        MaxRecords = 10,
    };

    var response = await
    _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
    return response.Activities;
}
```

- Weitere API-Informationen finden Sie unter [DescribeScalingActivities](#) in der APIAWS SDK for .NET -Referenz für .

Festlegen der gewünschten Kapazität einer Gruppe

Das folgende Codebeispiel zeigt, wie Sie die gewünschte Kapazität einer Auto Scaling-Gruppe festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [SetDesiredCapacity](#) in der APIAWS SDK for .NET - Referenz für .

Beenden einer Instance in einer Gruppe

Das folgende Codebeispiel zeigt, wie Sie eine Instance in einer Auto Scaling-Gruppe beenden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [TerminateInstanceInAutoScalingGroup](#) in der APIAWS SDK for .NET -Referenz für .

Aktualisieren einer Gruppe

Das folgende Codebeispiel zeigt, wie Sie die Konfiguration für eine Auto Scaling-Gruppe aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
    }
}
```

- Weitere API-Informationen finden Sie unter [UpdateAutoScalingGroup](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();
}
```

```
// Set up dependency injection for the AWS services.
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<IAmazonIdentityManagementService>()
            .AddAWSService<IAmazonDynamoDB>()
            .AddAWSService<IAmazonElasticLoadBalancingV2>()
            .AddAWSService<IAmazonSimpleSystemsManagement>()
            .AddAWSService<IAmazonAutoScaling>()
            .AddAWSService<IAmazonEC2>()
            .AddTransient<AutoScalerWrapper>()
            .AddTransient<ElasticLoadBalancerWrapper>()
            .AddTransient<SmParameterWrapper>()
            .AddTransient<Recommendations>()
            .AddSingleton<IConfiguration>(_configuration)
        )
    .Build();

ServicesSetup(host);
ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);
```



```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await DestroyResources(true);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _recommendations = host.Services.GetRequiredService<Recommendations>();
    _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
    _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
}

/// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
```

```
/// <returns>True if successful.</returns>
public static async Task<bool> Deploy(bool interactive)
{
    var protocol = "HTTP";
    var port = 80;
    var sshPort = 22;

    Console.WriteLine(
        "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
        "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
        "against various kinds of failures.\n\n" +
        "Some of the resources create by this demo are:\n");

    Console.WriteLine(
        "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
    Console.WriteLine(
        "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
    Console.WriteLine(
        "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
    Console.WriteLine(
        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
    if (interactive)
        Console.ReadLine();

    // Create and populate the DynamoDB table.
    var databaseTableName = _configuration["databaseName"];
    var recommendationsPath = Path.Join(_configuration["resourcePath"],
        "recommendations_objects.json");
    Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
    await _recommendations.CreateDatabaseWithName(databaseTableName);
    await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
    Console.WriteLine(new string('-', 80));
}
```

```
// Create the EC2 Launch Template.

Console.WriteLine(
    $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
    + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
    + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
    + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
    + "run a web server, such as Apache, with least-privileged
credentials.");
Console.WriteLine(
    "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
    + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
    + "that control the flow of the demo.");

var startupScriptPath = Path.Join(_configuration["resourcePath"],
    "server_startup_script.sh");
var instancePolicyPath = Path.Join(_configuration["resourcePath"],
    "instance_policy.json");
await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
    + "Availability Zone.\n");
var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
    + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

Console.WriteLine(new string('-', 80));
```

```
Console.WriteLine("Press Enter when you're ready to continue.");
if (interactive)
    Console.ReadLine();

Console.WriteLine("Creating variables that control the flow of the demo.");
await _smParameterWrapper.Reset();

Console.WriteLine(
    "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
    + "defines how the load balancer connects to instances. The load
balancer provides a\n"
    + "single endpoint where clients connect and dispatches requests to
instances in the group.");

    var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
    var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
    var subnetIds = subnets.Select(s => s.SubnetId).ToList();
    var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

    await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
subnetIds, targetGroup);
    await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
    Console.WriteLine("\nVerifying access to the load balancer endpoint...");
    var endPoint = await
_elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
    var loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

    if (!loadBalancerAccess)
    {
        Console.WriteLine("\nCouldn't connect to the load balancer, verifying
that the port is open...");

        var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
        ipString = ipString.Trim();
```

```
        var defaultSecurityGroup = await
_autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
        var portIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
        var sshPortIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
ipString);

        if (!portIsOpen)
        {
            Console.WriteLine(
                "\nFor this example to work, the default security group for your
default VPC must\n"
                + "allows access from this computer. You can either add it
automatically from this\n"
                + "example or add it yourself using the AWS Management Console.
\n");

            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
            }
        }

        if (!sshPortIsOpen)
        {
            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
            }
        }
        loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
```

```
        Console.WriteLine("Your load balancer is ready. You can access it by  
browsing to:");  
        Console.WriteLine($"\\thttp://{endPoint}\\n");  
    }  
    else  
    {  
        Console.WriteLine(  
            "\\nCouldn't get a successful response from the load balancer  
endpoint. Troubleshoot by\\n"  
            + "manually verifying that your VPC and security group are  
configured correctly and that\\n"  
            + "you can successfully make a GET request to the load balancer  
endpoint:\\n");  
        Console.WriteLine($"\\thttp://{endPoint}\\n");  
    }  
    Console.WriteLine(new string('-', 80));  
    Console.WriteLine("Press Enter when you're ready to continue with the  
demo.");  
    if (interactive)  
        Console.ReadLine();  
    return true;  
}  
  
/// <summary>  
/// Demonstrate the steps of the scenario.  
/// </summary>  
/// <param name="interactive">True to run as an interactive scenario.</param>  
/// <returns>Async task.</returns>  
public static async Task<bool> Demo(bool interactive)  
{  
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],  
        "ssm_only_policy.json");  
  
    Console.WriteLine(new string('-', 80));  
    Console.WriteLine("Resetting parameters to starting values for demo.");  
    await _smParameterWrapper.Reset();  
  
    Console.WriteLine("\\nThis part of the demonstration shows how to toggle  
different parts of the system\\n" +  
        "to create situations where the web service fails, and  
shows how using a resilient\\n" +  
        "architecture can keep the web service running in spite of  
these failures.");  
    Console.WriteLine(new string('-', 88));
```

```
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
    Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
    Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Let's reinstate the recommendation service.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
    Console.WriteLine(
        "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
```

```
        "access the DynamoDB recommendation table.\n"
    );
    await _autoScalerWrapper.CreateInstanceProfileWithName(
        _autoScalerWrapper.BadCredsPolicyName,
        _autoScalerWrapper.BadCredsRoleName,
        _autoScalerWrapper.BadCredsProfileName,
        ssmOnlyPolicy,
        new List<string> { "AmazonSSMManagedInstanceCore" }
    );
    var instances = await
    _autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
    var badInstanceId = instances.First();
    var instanceProfile = await
    _autoScalerWrapper.GetInstanceProfile(badInstanceId);
    Console.WriteLine(
        $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
        "bad credentials...\n"
    );
    await _autoScalerWrapper.ReplaceInstanceProfile(
        badInstanceId,
        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");
```



```
        Console.WriteLine("\nBy implementing deep health checks, the load balancer  
can detect when one of the instances is failing");  
        Console.WriteLine("and take that instance out of rotation.");  
  
        await  
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,  
"deep");  
  
        Console.WriteLine($"Now, checking target health indicates that the  
instance with bad credentials ({badInstanceId})");  
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for  
the load balancer to detect the unhealthy");  
        Console.WriteLine("instance. Sending a GET request to the load balancer  
endpoint always returns a recommendation, because");  
        Console.WriteLine("the load balancer takes unhealthy instances out of its  
rotation.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nBecause the instances in this demo are controlled by an  
auto scaler, the simplest way to fix an unhealthy");  
        Console.WriteLine("instance is to terminate it and let the auto scaler start  
a new instance to replace it.");  
  
        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);  
  
        Console.WriteLine($"Even while the instance is terminating and the new  
instance is starting, sending a GET");  
        Console.WriteLine("request to the web service continues to get a successful  
recommendation response because");  
        Console.WriteLine("starts and reports as healthy, it is included in the load  
balancing rotation.");  
        Console.WriteLine("Note that terminating and replacing an instance typically  
takes several minutes, during which time you");  
        Console.WriteLine("can see the changing health check status until the new  
instance is running and healthy.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nIf the recommendation service fails now, deep health  
checks mean all instances report as unhealthy.");
```

```
        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
```

```

        await
        _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
        _recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Erstellen Sie eine Klasse, die Auto-Scaling- und Amazon-EC2-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
}

```

```
private readonly string _badCredsRoleName = "";
private readonly string _badCredsPolicyName = "";
private readonly string _keyPairName = "";

public string GroupName => _groupName;
public string KeyPairName => _keyPairName;
public string LaunchTemplateName => _launchTemplateName;
public string InstancePolicyName => _instancePolicyName;
public string BadCredsProfileName => _badCredsProfileName;
public string BadCredsRoleName => _badCredsRoleName;
public string BadCredsPolicyName => _badCredsPolicyName;

/// <summary>
/// Constructor for the AutoScalerWrapper.
/// </summary>
/// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
/// <param name="amazonEc2">The injected EC2 client.</param>
/// <param name="amazonIam">The injected IAM client.</param>
/// <param name="amazonSsm">The injected SSM client.</param>
public AutoScalerWrapper(
    IAmazonAutoScaling amazonAutoScaling,
    IAmazonEC2 amazonEc2,
    IAmazonSimpleSystemsManagement amazonSsm,
    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}
```

```

}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,
    string ssmOnlyPolicyFile,
    List<string>? awsManagedPolicies = null)
{
    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": {" +
            "\"Service\": [" +
                "\"ec2.amazonaws.com\"" +
            "]" +
            "}, " +
            "\"Action\": \"sts:AssumeRole\"" +
        "}] " +
    "}";

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {

```

```
var createPolicyResult = await _amazonIam.CreatePolicyAsync(
    new CreatePolicyRequest
    {
        PolicyName = policyName,
        PolicyDocument = policyDocument
    });
policyArn = createPolicyResult.Policy.Arn;
}
catch (EntityAlreadyExistsException)
{
    // The policy already exists, so we look it up to get the Arn.
    var policiesPaginator = _amazonIam.Paginators.ListPolicies(
        new ListPoliciesRequest()
        {
            Scope = PolicyScopeType.Local
        });
    // Get the entire list using the paginator.
    await foreach (var policy in policiesPaginator.Policies)
    {
        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
```

```
        {
            foreach (var awsPolicy in awsManagedPolicies)
            {
                await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                {
                    PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                    RoleName = roleName
                });
            }
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Policy already exists.");
    var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
        new GetInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
}
```

```
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyPairName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyPairName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
        File.Delete($"{deleteKeyPairName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
    }
}
```



```
    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            }
        );
    }
}
```

```
    });
    return launchTemplateResponse.LaunchTemplate;
}

/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
    return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
}

/// <summary>
/// Create an EC2 Auto Scaling group of a specified size and name.
/// </summary>
/// <param name="groupSize">The size for the group.</param>
/// <param name="groupName">The name for the group.</param>
/// <param name="availabilityZones">The availability zones for the group.</
param>
/// <returns>Async task.</returns>
public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
{
    try
    {
        await _amazonAutoScaling.CreateAutoScalingGroupAsync(
            new CreateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                AvailabilityZones = availabilityZones,
                LaunchTemplate =
                    new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                    {
                        LaunchTemplateName = _launchTemplateName,
                        Version = "$Default"
                    },
                MaxSize = groupSize,
                MinSize = groupSize
            });
    }
}
```

```
        Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
    }
}

/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
    return vpcResponse.Vpcs[0];
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    var subnets = new List<Subnet>();
    var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
        new DescribeSubnetsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("vpc-id", new List<string>() { vpcId}),
                new ("availability-zone", availabilityZones),
            }
        });
    while (subnetPaginator.HasNext())
    {
        subnets.AddRange(subnetPaginator.CurrentPage.Items);
    }
}
```

```
        new ("default-for-az", new List<string>() { "true" })
    }
});

// Get the entire list using the paginator.
await foreach (var subnet in subnetPaginator.Subnets)
{
    subnets.Add(subnet);
}

return subnets;
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
```

```
{
    await _amazonIam.RemoveRoleFromInstanceProfileAsync(
        new RemoveRoleFromInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
    await _amazonIam.DeleteInstanceProfileAsync(
        new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
    var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
        new ListAttachedRolePoliciesRequest() { RoleName = roleName });
    foreach (var policy in attachedPolicies.AttachedPolicies)
    {
        await _amazonIam.DetachRolePolicyAsync(
            new DetachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policy.PolicyArn
            });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
```

```
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
    new DescribeAutoScalingGroupsRequest()
    {
        AutoScalingGroupNames = new List<string>() { group }
    });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
        new DescribeIamInstanceProfileAssociationsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("instance-id", new List<string>() { instanceId })
            },
        });
    return response.IamInstanceProfileAssociations[0];
}

/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
```

```
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
        new ReplaceIamInstanceProfileAssociationRequest()
        {
            AssociationId = associationId,
            IamInstanceProfile = new IamInstanceProfileSpecification()
            {
                Name = credsProfileName
            }
        });
    // Allow time before resetting.
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        await _amazonEc2.RebootInstancesAsync(
            new RebootInstancesRequest(new List<string>() { instanceId }));
        Thread.Sleep(10000);

        var instancesPaginator =
        _amazonSsm.Paginators.DescribeInstanceInformation(
            new DescribeInstanceInformationRequest());
        // Get the entire list using the paginator.
        await foreach (var instance in
instancesPaginator.InstanceInformationList)
        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
```

```
        DocumentName = "AWS-RunShellScript",
        Parameters = new Dictionary<string, List<string>>()
        {
            {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
        }
    });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                });
            stopping = true;
        }
        catch (ScalingActivityInProgressException)
        {
            Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
```



```
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                });
            stopped = true;
        }
        catch (Exception e)
            when ((e is ScalingActivityInProgressException)
                || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
```

```
        new UpdateAutoScalingGroupRequest()
        {
            AutoScalingGroupName = groupName,
            MinSize = 0
        });
    var group = describeGroupsResponse.AutoScalingGroups[0];
    foreach (var instance in group.Instances)
    {
        await TryTerminateInstanceById(instance.InstanceId);
    }

    await TryDeleteGroupByName(groupName);
}
else
{
    Console.WriteLine($"No groups found with name {groupName}.");
}
}

/// <summary>
/// Get the default security group for a specified Vpc.
/// </summary>
/// <param name="vpc">The Vpc to search.</param>
/// <returns>The default security group.</returns>
public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
{
    var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
        new DescribeSecurityGroupsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("group-name", new List<string>() { "default" }),
                new ("vpc-id", new List<string>() { vpc.VpcId })
            }
        });
    return groupResponse.SecurityGroups[0];
}

/// <summary>
/// Verify the default security group of a Vpc allows ingress from the calling
computer.
/// This can be done by allowing ingress from this computer's IP address.
```

```
    /// In some situations, such as connecting from a corporate network, you must
    /// instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    /// while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
    ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }

                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
                    to either this computer's IP\n" +
                    "address, to all IP addresses (0.0.0.0/0), or
                    to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }
    }
}
```

```

    }

    return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()
            {
                new IpPermission()
                {
                    FromPort = port,
                    ToPort = port,
                    IpProtocol = "tcp",
                    Ipv4Ranges = new List<IpRange>()
                    {
                        new IpRange() { CidrIp = $"{ipAddress}/32" }
                    }
                }
            }
        });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>

```

```

    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}

```

Erstellen Sie eine Klasse, die Elastic-Load-Balancing-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
    }
}

```

```
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```
public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
{
    List<TargetHealthDescription> result = null!;
    try
    {
        var groupResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                new DescribeTargetGroupsRequest()
                {
                    Names = new List<string>() { groupName }
                });
        var healthResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                new DescribeTargetHealthRequest()
                {
                    TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                });
        ;
        result = healthResponse.TargetHealthDescriptions;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine($"Target group {groupName} not found.");
    }
    return result;
}

/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
```

```
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
        ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
    subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
    }
}
```



```
var loadBalancerReady = false;
while (!loadBalancerReady)
{
    try
    {
        var describeResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });

        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}

///
```

```
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }

    return success;
}

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
```

```
    try
    {
        var describeLoadBalancerResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });
        var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
        await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
            new DeleteLoadBalancerRequest()
            {
                LoadBalancerArn = lbArn
            }
        );
    }
    catch (LoadBalancerNotFoundException)
    {
        Console.WriteLine($"Load balancer {name} not found.");
    }
}

/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
```

```

        new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
        Console.WriteLine($"Deleted load balancing target group
{groupName}.");
        done = true;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine(
            $"Target group {groupName} not found, could not delete.");
        done = true;
    }
    catch (ResourceInUseException)
    {
        Console.WriteLine("Target group not yet released, waiting...");
        Thread.Sleep(10000);
    }
    }
}
}
}

```

Erstellen Sie eine Klasse, die DynamoDB zum Simulieren eines Empfehlungsservices verwendet.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {

```

```
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()
                {
                    AttributeName = "MediaType",
                    KeyType = KeyType.HASH
                },
                new KeySchemaElement()
                {
                    AttributeName = "ItemId",
                    KeyType = KeyType.RANGE
                }
            },
        },
    }
}
```

```
        ProvisionedThroughput = new ProvisionedThroughput()
        {
            ReadCapacityUnits = 5,
            WriteCapacityUnits = 5
        }
    };
    await _amazonDynamoDb.CreateTableAsync(createRequest);

    // Wait until the table is ACTIVE and then report success.
    Console.WriteLine("\nWaiting for table to become active...");

    var request = new DescribeTableRequest
    {
        TableName = tableName
    };

    TableStatus status;
    do
    {
        Thread.Sleep(2000);

        var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
        status = describeTableResponse.Table.TableStatus;

        Console.WriteLine(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
    return false;
}
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
```

```
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}
}
```

Erstellen Sie eine Klasse, die Systems-Manager-Aktionen umschließt.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
parameters
```

```
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }
}
```



```
/// <summary>
/// Set the value of a named Systems Manager parameter.
/// </summary>
/// <param name="name">The name of the parameter.</param>
/// <param name="value">The value to set.</param>
/// <returns>Async task.</returns>
public async Task PutParameterByName(string name, string value)
{
    await _amazonSimpleSystemsManagement.PutParameterAsync(
        new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)

- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Verwalten von Gruppen und Instances

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Amazon EC2 Auto Scaling-Gruppe mit einer Startvorlage und Availability Zones und erhalten Sie Informationen über laufende Instances.
- Aktivieren Sie die Erfassung von Amazon- CloudWatch Metriken.
- Aktualisieren Sie die gewünschte Kapazität der Gruppe und warten Sie, bis eine Instance gestartet wird.
- Beenden Sie eine Instance in der Gruppe.
- Auflisten von Skalierungsaktivitäten, die als Reaktion auf Benutzeranfragen und Kapazitätsänderungen auftreten.
- Rufen Sie Statistiken für CloudWatch Metriken ab und bereinigen Sie dann Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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.First().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 is not null)
{
    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 is not null)
{
    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!);
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!);
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.WriteLine("\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 umschließen die CloudWatch Aktionen Auto Scaling , Amazon EC2 und .

```
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
    };

    var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
    Console.WriteLine($"{groupName} Auto Scaling Group created");
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <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<Amazon.AutoScaling.Model.Activity>>
DescribeScalingActivitiesAsync(
    string groupName)
{
    var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
    {
        AutoScalingGroupName = groupName,
        MaxRecords = 10,
    };

    var response = await
_amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
    return response.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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
        _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var 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 groupList = new List<string>
        {
            groupName,
        };
    }
}
```

```
        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        var 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<Amazon.CloudWatch.Model.Metric>>
GetCloudWatchMetricsAsync(string groupName)
{
    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);

    return response.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 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);

    return response.Datapoints;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAutoScalingInstances](#)
 - [DescribeScalingActivities](#)
 - [DisableMetricsCollection](#)
 - [EnableMetricsCollection](#)
 - [SetDesiredCapacity](#)
 - [TerminateInstanceInAutoScalingGroup](#)

- [UpdateAutoScalingGroup](#)

Amazon-Bedrock-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Bedrock verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon Bedrock.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace ListFoundationModelsExample
{
    /// <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
        {
            ListFoundationModelsResponse 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)}}");
}
}
}
```

- Weitere API-Informationen finden Sie unter [ListFoundationModels](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)

Aktionen

Auflisten verfügbarer Amazon-Bedrock-Grundlagenmodelle

Das folgende Codebeispiel zeigt, wie Sie verfügbare Amazon-Bedrock-Grundlagenmodelle auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die verfügbaren Bedrock-Grundlagenmodelle 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
    {
        ListFoundationModelsResponse 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);
    }
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [ListFoundationModels](#) in der APIAWS SDK for .NET -Referenz für .

Amazon-Bedrock-Laufzeitbeispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Bedrock Runtime verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Bildgenerierung mit Amazon Titan Image Generator G1

Das folgende Codebeispiel zeigt, wie das G1-Modell des Amazon Titan Image Generators auf Amazon Bedrock zur Bildgenerierung aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Amazon Titan Image Generator G1-Grundlagenmodell asynchron auf, um Bilder zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Amazon Titan Image Generator G1 model to run
    an inference based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that describes the image Amazon Titan
    Image Generator G1 has to generate.</param>
    /// <returns>A base-64 encoded image generated by model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for Amazon Titan Image
    Generator G1, refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
    parameters-titan-image.html
    /// </remarks>
    public static async Task<string?> InvokeTitanImageGeneratorG1Async(string
    prompt, int seed)
    {
        string titanImageGeneratorG1ModelId = "amazon.titan-image-generator-v1";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()
        {
            { "taskType", "TEXT_IMAGE" },
            { "textToImageParams", new JsonObject()
                {
                    { "text", prompt }
                }
            },
            { "imageGenerationConfig", new JsonObject()
                {
                    { "numberOfImages", 1 },
                    { "quality", "standard" },
                    { "cfgScale", 8.0f },
                    { "height", 512 },
                    { "width", 512 },
                    { "seed", seed }
                }
            }
        }
    }
}
```

```
        }
    }.ToJsonString();

    try
    {
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
        {
            ModelId = titanImageGeneratorG1ModelId,
            Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
            ContentType = "application/json",
            Accept = "application/json"
        });

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var results = JsonNode.ParseAsync(response.Body).Result?
["images"]?.AsArray();

            return results?[0]?.GetValue<string>();
        }
        else
        {
            Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
        }
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine(e.Message);
    }
    return null;
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Bildgenerierung mit Stability.ai Stable XL

Das folgende Codebeispiel zeigt, wie das Modell Stability.ai Stable XL auf Amazon Bedrock zur Bildgenerierung aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie asynchron das stabile XL-Grundlagenmodell Stability.ai auf, um Bilder zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Stability.ai Stable Diffusion XLmodel to run
    an inference based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that describes the image Stability.ai
    Stable Diffusion XL has to generate.</param>
    /// <returns>A base-64 encoded image generated by model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for Stability.ai Stable
    Diffusion XL, refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
    parameters-stability-diffusion.html
    /// </remarks>
    public static async Task<string?> InvokeStableDiffusionXLG1Async(string
    prompt, int seed, string? stylePreset = null)
    {
        string stableDiffusionXLModelId = "stability.stable-diffusion-xl";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        var jsonPayload = new JsonObject()
        {
            { "text_prompts", new JsonArray() {
                new JsonObject()
                {
                    { "text", prompt }
                }
            }
        },
            { "seed", seed }
        }
```

```
};

if (!string.IsNullOrEmpty(stylePreset))
{
    jsonPayload.Add("style_preset", stylePreset);
}

string payload = jsonPayload.ToString();

try
{
    InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
    {
        ModelId = stableDiffusionXLModelId,
        Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
        ContentType = "application/json",
        Accept = "application/json"
    });

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var results = JsonNode.ParseAsync(response.Body).Result?
["artifacts"]?.AsArray();

        return results?[0]?["base64"]?.GetValue<string>();
    }
    else
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return null;
}
```


- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit AI21 Labs Jurassic-2

Das folgende Codebeispiel zeigt, wie das AI21 Labs Jurassic-2-Modell auf Amazon Bedrock zur Textgenerierung aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das AI21 Labs Jurassic-2-Grundlagenmodell asynchron auf.

```
    /// <summary>
    /// Asynchronously invokes the AI21 Labs Jurassic-2 model to run an
    inference based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Claude to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for AI21 Labs Jurassic-2,
    refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
    parameters-jurassic2.html
    /// </remarks>
    public static async Task<string> InvokeJurassic2Async(string prompt)
    {
        string jurassic2ModelId = "ai21.j2-mid-v1";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()
```

```
        {
            { "prompt", prompt },
            { "maxTokens", 200 },
            { "temperature", 0.5 }
        }.ToJsonString();

        string generatedText = "";
        try
        {
            InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
            {
                ModelId = jurassic2ModelId,
                Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
                ContentType = "application/json",
                Accept = "application/json"
            });

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                return JsonNode.ParseAsync(response.Body)
                    .Result?["completions"]?
                    .ToArray()[0]?["data"]?
                    .AsObject()["text"]?.GetValue<string>() ?? "";
            }
            else
            {
                Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
            }
        }
        catch (AmazonBedrockRuntimeException e)
        {
            Console.WriteLine(e.Message);
        }
        return generatedText;
    }
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit Amazon Titan Text G1

Das folgende Codebeispiel zeigt, wie das Amazon Titan Text G1-Modell in Amazon Bedrock zur Textgenerierung aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Amazon Titan Text G1-Grundlagenmodell asynchron auf, um Text zu generieren.

```
/// <summary>
/// Asynchronously invokes the Amazon Titan Text G1 Express model to run an
inference based on the provided input.
/// </summary>
/// <param name="prompt">The prompt that you want Amazon Titan Text G1
Express to complete.</param>
/// <returns>The inference response from the model</returns>
/// <remarks>
/// The different model providers have individual request and response
formats.
/// For the format, ranges, and default values for Amazon Titan Text G1
Express, refer to:
///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-titan-text.html
/// </remarks>
public static async Task<string> InvokeTitanTextG1Async(string prompt)
{
    string titanTextG1ModelId = "amazon.titan-text-express-v1";

    AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

    string payload = new JsonObject()
    {
        { "inputText", prompt },
        { "textGenerationConfig", new JsonObject()
        {
            { "maxTokenCount", 512 },
```

```

        { "temperature", 0f },
        { "topP", 1f }
    }
}
}.ToJsonString();

string generatedText = "";
try
{
    InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
    {
        ModelId = titanTextG1ModelId,
        Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
        ContentType = "application/json",
        Accept = "application/json"
    });

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var results = JsonNode.ParseAsync(response.Body).Result?
["results"]?.ToArray();

        return results is null ? "" : string.Join(" ", results.Select(x
=> x?["outputText"]?.GetValue<string?>()));
    }
    else
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return generatedText;
}

```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit Anthropic Claude 2

Das folgende Codebeispiel zeigt, wie das Modell Anthropic Claude 2 in Amazon Bedrock zur Textgenerierung aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Grundlagenmodell Anthropic Claude 2 asynchron auf, um Text zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Anthropic Claude 2 model to run an inference
    based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Claude to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for Anthropic Claude, refer
    to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
    parameters-claude.html
    /// </remarks>
    public static async Task<string> InvokeClaudeAsync(string prompt)
    {
        string claudeModelId = "anthropic.claude-v2";

        // Claude requires you to enclose the prompt as follows:
        string enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()
        {
```

```
        { "prompt", enclosedPrompt },
        { "max_tokens_to_sample", 200 },
        { "temperature", 0.5 },
        { "stop_sequences", new JSONArray("\n\nHuman:") }
    }.ToJsonString();

    string generatedText = "";
    try
    {
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
        {
            ModelId = claudeModelId,
            Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
            ContentType = "application/json",
            Accept = "application/json"
        });

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            return JsonNode.ParseAsync(response.Body).Result?
["completion"]?.GetValue<string>() ?? "";
        }
        else
        {
            Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
        }
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine(e.Message);
    }
    return generatedText;
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit Anthropic Claude 2 mit einem Antwort-Stream

Das folgende Codebeispiel zeigt, wie das Modell Anthropic Claude 2 in Amazon Bedrock zur Textgenerierung mit einem Antwortstream aufgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Modell Anthropic Claude 2 asynchron auf und verarbeiten Sie den Antwort-Stream.

```

    /// <summary>
    /// Asynchronously invokes the Anthropic Claude 2 model to run an inference
    based on the provided input and process the response stream.
    /// </summary>
    /// <param name="prompt">The prompt that you want Claude to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for Anthropic Claude, refer
    to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
    parameters-claude.html
    /// </remarks>
    public static async IEnumerable<string>
    InvokeClaudeWithResponseStreamAsync(string prompt, [EnumeratorCancellation]
    CancellationToken cancellationToken = default)
    {
        string claudeModelId = "anthropic.claude-v2";

        // Claude requires you to enclose the prompt as follows:
        string enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()

```

```
        {
            { "prompt", enclosedPrompt },
            { "max_tokens_to_sample", 200 },
            { "temperature", 0.5 },
            { "stop_sequences", new JSONArray("\n\nHuman:") }
        }.ToJsonString();

        InvokeModelWithResponseStreamResponse? response = null;

        try
        {
            response = await client.InvokeModelWithResponseStreamAsync(new
InvokeModelWithResponseStreamRequest()
            {
                ModelId = claudeModelId,
                Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
                ContentType = "application/json",
                Accept = "application/json"
            });
        }
        catch (AmazonBedrockRuntimeException e)
        {
            Console.WriteLine(e.Message);
        }

        if (response is not null && response.HttpStatusCode ==
System.Net.HttpStatusCode.OK)
        {
            // create a buffer to write the event in to move from a push mode to
a pull mode
            Channel<string> buffer = Channel.CreateUnbounded<string>();
            bool isStreaming = true;

            response.Body.ChunkReceived += BodyOnChunkReceived;
            response.Body.StartProcessing();

            while ((!cancellationToken.IsCancellationRequested && isStreaming)
|| (!cancellationToken.IsCancellationRequested && buffer.Reader.Count > 0))
            {
                // pull the completion from the buffer and add it to the
IAsyncEnumerable collection
                yield return await buffer.Reader.ReadAsync(cancellationToken);
            }
            response.Body.ChunkReceived -= BodyOnChunkReceived;
```



```
        yield break;

        // handle the ChunkReceived events
        async void BodyOnChunkReceived(object? sender,
EventStreamEventReceivedArgs<PayloadPart> e)
        {
            var streamResponse =
JsonSerializer.Deserialize<JsonObject>(e.EventStreamEvent.Bytes) ??
throw new NullReferenceException($"Unable to deserialize
{nameof(e.EventStreamEvent.Bytes)}");

            if (streamResponse["stop_reason"]?.GetValue<string?>() != null)
            {
                isStreaming = false;
            }

            // write the received completion chunk into the buffer
            await
buffer.Writer.WriteAsync(streamResponse["completion"]?.GetValue<string>(),
cancellationTokens);
        }
    }
    else if (response is not null)
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }

    yield break;
}
}
```

- Weitere API-Informationen finden Sie unter [InvokeModelWithResponseStream](#) in der APIAWS SDK for .NET -Referenz zu .

Textgenerierung mit Meta Llama 2 Chat

Das folgende Codebeispiel zeigt, wie das Meta-Llama-2-Chat-Modell auf Amazon Bedrock zur Textgenerierung aufgerufen wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Meta-Llama-2-Grundlagenmodell asynchron auf, um Text zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Meta Llama 2 Chat model to run an inference
based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Llama 2 to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
formats.
    /// For the format, ranges, and default values for Meta Llama 2 Chat, refer
to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-meta.html
    /// </remarks>
    public static async Task<string> InvokeLlama2Async(string prompt)
    {
        string llama2ModelId = "meta.llama2-13b-chat-v1";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()
        {
            { "prompt", prompt },
            { "max_gen_len", 512 },
            { "temperature", 0.5 },
            { "top_p", 0.9 }
        }.ToJsonString();

        string generatedText = "";
        try
        {
```

```
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
    {
        ModelId = llama2ModelId,
        Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
        ContentType = "application/json",
        Accept = "application/json"
    });

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        return JsonNode.ParseAsync(response.Body)
            .Result?["generation"]?.GetValue<string>() ?? "";
    }
    else
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return generatedText;
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit Mistral 7B

Das folgende Codebeispiel zeigt, wie Sie das Modell Mistral 7B in Amazon Bedrock zur Textgenerierung aufrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Grundlagenmodell Mistral 7B asynchron auf, um Text zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Mistral 7B model to run an inference based on
the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Mistral 7B to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
formats.
    /// For the format, ranges, and default values for Mistral 7B, refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-mistral.html
    /// </remarks>
    public static async Task<List<string?>> InvokeMistral7BAsync(string prompt)
    {
        string mistralModelId = "mistral.mistral-7b-instruct-v0:2";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USWest2);

        string payload = new JsonObject()
        {
            { "prompt", prompt },
            { "max_tokens", 200 },
            { "temperature", 0.5 }
        }.ToJsonString();

        List<string?>? generatedText = null;
        try
        {
            InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
```

```
        {
            ModelId = mistralModelId,
            Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
            ContentType = "application/json",
            Accept = "application/json"
        });

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var results = JsonNode.ParseAsync(response.Body).Result?
["outputs"]?.ToArray();

            generatedText = results?.Select(x => x?
["text"]?.GetValue<string?>())?.ToList();
        }
        else
        {
            Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
        }
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine(e.Message);
    }
    return generatedText ?? [];
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Textgenerierung mit Mixtral 8x7B

Das folgende Codebeispiel zeigt, wie das Modell Mixtral 8x7B in Amazon Bedrock zur Textgenerierung aufgerufen wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie das Grundlagenmodell Mixtral 8x7B asynchron auf, um Text zu generieren.

```
    /// <summary>
    /// Asynchronously invokes the Mixtral 8x7B model to run an inference based
on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Mixtral 8x7B to
complete.</param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
formats.
    /// For the format, ranges, and default values for Mixtral 8x7B, refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-mistral.html
    /// </remarks>
    public static async Task<List<string?>> InvokeMixtral8x7BAsync(string
prompt)
    {
        string mixtralModelId = "mistral.mixtral-8x7b-instruct-v0:1";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USWest2);

        string payload = new JsonObject()
        {
            { "prompt", prompt },
            { "max_tokens", 200 },
            { "temperature", 0.5 }
        }.ToJsonString();

        List<string?>? generatedText = null;
        try
        {
```

```
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
    {
        ModelId = mixtralModelId,
        Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
        ContentType = "application/json",
        Accept = "application/json"
    });

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var results = JsonNode.ParseAsync(response.Body).Result?
["outputs"]?.ToArray();

        generatedText = results?.Select(x => x?
["text"]?.GetValue<string?>())?.ToList();
    }
    else
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return generatedText ?? [];
}
```

- Weitere API-Informationen finden Sie unter [InvokeModel](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erstellen einer Playground-Anwendung für die Interaktion mit Amazon-Bedrock-Grundlagenmodellen

Das folgende Codebeispiel zeigt, wie Sie Playgrounds erstellen, um über verschiedene Modalitäten mit Amazon-Bedrock-Grundlagenmodellen zu interagieren.

AWS SDK for .NET

.NET Foundation Model (FM) Playground ist eine .NET MAUI Blazor-Beispielanwendung, die zeigt, wie Amazon Bedrock aus C#-Code verwendet wird. Dieses Beispiel zeigt, wie .NET- und C#-Entwickler Amazon Bedrock verwenden können, um generative KI-fähige Anwendungen zu erstellen. Sie können die folgenden vier Playgrounds verwenden, um Amazon-Bedrock-Grundlagenmodelle zu testen und mit ihnen zu interagieren:

- Ein Text-Playground.
- Ein Chat-Playground.
- Ein Sprach-Chat-Playground.
- Ein Image-Playground.

Das Beispiel listet auch die Grundlagenmodelle auf, auf die Sie Zugriff haben, und ihre Merkmale. Quellcode und Bereitstellungsanweisungen finden Sie im Projekt unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon-Bedrock-Laufzeit

AWS CloudFormation -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS CloudFormation.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS CloudFormation

Das folgende Codebeispiel veranschaulicht die ersten Schritte mit AWS CloudFormation.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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($"In 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($"Stack: {stack.StackName}");
        Console.WriteLine($"  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.SSO0IDC 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;
}
}
```

- Weitere API-Informationen finden Sie unter [DescribeStackResources](#) in der APIAWS SDK for .NET -Referenz für .

CloudWatch -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden CloudWatch.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo 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.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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($"Hello Amazon CloudWatch! Following are some metrics
available in the {metricNamespace} namespace:");
    Console.WriteLine();
    foreach (var metric in response.Metrics.Take(5))
    {
        Console.WriteLine($"Metric: {metric.MetricName}");
        Console.WriteLine($"Namespace: {metric.Namespace}");
        Console.WriteLine($"Dimensions: {string.Join(", ",
metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
        Console.WriteLine();
    }
}
}
```

- Weitere API-Informationen finden Sie unter [ListMetrics](#) in der APIAWS SDK for .NET -Referenz zu .

Themen


- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen eines Dashboards

Das folgende Codebeispiel zeigt, wie Sie ein Amazon- CloudWatch Dashboard erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [PutDashboard](#) in der APIAWS SDK for .NET - Referenz für .

Metrik-Alarm erstellen

Das folgende Codebeispiel zeigt, wie Sie einen Amazon- CloudWatch Alarm erstellen oder aktualisieren und ihn der angegebenen Metrik, dem mathematischen Metrikausdruck, dem Anomalieerkennungmodell oder der Metrics-Insights-Abfrage zuordnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [PutMetricAlarm](#) in der APIAWS SDK for .NET - Referenz für .

Anomaliedetektor erstellen

Das folgende Codebeispiel zeigt, wie Sie einen Amazon CloudWatch -Anomaliedetektor erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [PutAnomalyDetector](#) in der APIAWS SDK for .NET - Referenz für .

Löschen von Alarmen

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Alarme löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteAlarms](#) in der APIAWS SDK for .NET - Referenz für .

Anomaliedetektor löschen

Das folgende Codebeispiel zeigt, wie Sie einen Amazon CloudWatch -Anomaliedetektor löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteAnomalyDetector](#) in der APIAWS SDK for .NET -Referenz für .

Dashboards löschen

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Dashboards löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DeleteDashboards](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben eines Alarmverlaufs

Das folgende Codebeispiel zeigt, wie Sie einen Amazon- CloudWatch Alarmverlauf beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [DescribeAlarmHistory](#) in der APIAWS SDK for .NET -Referenz für .

Alarmer beschreiben

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Alarmer beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeAlarms](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben von Alarmen für eine Metrik

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Alarme für eine Metrik beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

```
}

```

- Weitere API-Informationen finden Sie unter [DescribeAlarmsForMetric](#) in der APIAWS SDK for .NET -Referenz für .

Anomaliedetektoren beschreiben

Das folgende Codebeispiel zeigt, wie Sie Amazon CloudWatch -Anomaliedetektoren beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
    }
```

- Weitere API-Informationen finden Sie unter [DescribeAnomalyDetectors](#) in der APIAWS SDK for .NET -Referenz für .

Deaktivieren von Alarmaktionen

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Alarmaktionen deaktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [DisableAlarmActions](#) in der APIAWS SDK for .NET -Referenz für .

Aktivieren von Alarmaktionen

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Alarmaktionen aktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [EnableAlarmActions](#) in der APIAWS SDK for .NET - Referenz für .

Metrikdaten-Image abrufen

Das folgende Codebeispiel zeigt, wie Sie ein Amazon- CloudWatch Metrikdaten-Image abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [GetMetricWidgetImage](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen der Dashboard-Details

Das folgende Codebeispiel zeigt, wie Sie Amazon CloudWatch -Dashboard-Details abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [GetDashboard](#) in der APIAWS SDK for .NET - Referenz für .

Metrikdaten abrufen

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Metrikdaten abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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}";
        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,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
}
```

- Weitere API-Informationen finden Sie unter [GetMetricData](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Metrikstatistiken

Das folgende Codebeispiel zeigt, wie Sie Amazon- CloudWatch Metrikstatistiken abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```

- Weitere API-Informationen finden Sie unter [GetMetricStatistics](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Dashboards

Das folgende Codebeispiel zeigt, wie Sie Amazon CloudWatch -Dashboards auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}

```


- Weitere API-Informationen finden Sie unter [ListDashboards](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Metriken

Das folgende Codebeispiel zeigt, wie Sie die Metadaten für Amazon- CloudWatch Metriken auflisten. Um Daten für eine Metrik abzurufen, verwenden Sie die GetMetricStatistics Aktionen GetMetricData oder .

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;  
}
```

- Weitere API-Informationen finden Sie unter [ListMetrics](#) in der APIAWS SDK for .NET -Referenz für .

Einfügen von Daten in eine Metrik

Das folgende Codebeispiel zeigt, wie Metrikdatenpunkte in Amazon veröffentlicht werden CloudWatch.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
    }
}
```

- Weitere API-Informationen finden Sie unter [PutMetricData](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit CloudWatch-Metriken, -Dashboards und -Alarmen

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.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung 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($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
    var metrics = await _cloudWatchWrapper.ListMetrics();
    // Get a distinct list of namespaces.
    var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
    for (int i = 0; i < namespaces.Count; i++)
    {
        Console.WriteLine($"  {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($"{i + 1}. {alarm.AlarmName}");
        Console.WriteLine($"{i}\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++)
{
    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($"{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($"{i + 1}. Getting 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($"{i + 1}. File saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"{i + 1}. Press 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($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"Deleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"Cleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
            anomalyDetector != null)
        {
            Console.WriteLine($"Cleaning 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;
}

/// <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;
}

/// <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}";
        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,
            });

        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;
    }

    /// <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 for .NET -API-Referenz.
 - [DeleteAlarms](#)
 - [DeleteAnomalyDetector](#)
 - [DeleteDashboards](#)
 - [DescribeAlarmHistory](#)
 - [DescribeAlarms](#)
 - [DescribeAlarmsForMetric](#)
 - [DescribeAnomalyDetectors](#)
 - [GetMetricData](#)
 - [GetMetricStatistics](#)
 - [GetMetricWidgetImage](#)
 - [ListMetrics](#)
 - [PutAnomalyDetector](#)
 - [PutDashboard](#)
 - [PutMetricAlarm](#)
 - [PutMetricData](#)

CloudWatch Protokolliert Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit - CloudWatch Protokollen verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Verknüpfen eines Schlüssels mit einer Protokollgruppe

Das folgende Codebeispiel zeigt, wie Sie einen - AWS KMS Schlüssel einer vorhandenen CloudWatch Logs-Protokollgruppe zuordnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to associate an AWS Key Management Service (AWS KMS) key with
/// an Amazon CloudWatch Logs log group.
/// </summary>
public class AssociateKmsKey
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
    }
}
```

```
// constructor.
var client = new AmazonCloudWatchLogsClient();

string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
string groupName = "cloudwatchlogs-example-loggroup";

var request = new AssociateKmsKeyRequest
{
    KmsKeyId = kmsKeyId,
    LogGroupName = groupName,
};

var response = await client.AssociateKmsKeyAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Successfully associated KMS key ID: {kmsKeyId}
with log group: {groupName}.");
}
else
{
    Console.WriteLine("Could not make the association between:
{kmsKeyId} and {groupName}.");
}
}
```

- Weitere API-Informationen finden Sie unter [AssociateKmsKey](#) in der APIAWS SDK for .NET - Referenz für .

Stornieren einer Exportaufgabe

Das folgende Codebeispiel zeigt, wie Sie eine vorhandene CloudWatch Protokollexportaufgabe abbrechen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

```
}  
}
```

- Weitere API-Informationen finden Sie unter [CancelExportTask](#) in der APIAWS SDK for .NET - Referenz zu .

Eine Protokollgruppe erstellen

Das folgende Codebeispiel zeigt, wie Sie eine neue CloudWatch Logs-Protokollgruppe erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;  
using Amazon.CloudWatchLogs.Model;  
  
/// <summary>  
/// Shows how to create an Amazon CloudWatch Logs log group.  
/// </summary>  
public class CreateLogGroup  
{  
    public static async Task Main()  
    {  
        // This client object will be associated with the same AWS Region  
        // as the default user on this system. If you need to use a  
        // different AWS Region, pass it as a parameter to the client  
        // constructor.  
        var client = new AmazonCloudWatchLogsClient();  
  
        string logGroupName = "cloudwatchlogs-example-loggroup";  
  
        var request = new CreateLogGroupRequest
```

```
        {
            LogGroupName = logGroupName,
        };

        var response = await client.CreateLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
        }
        else
        {
            Console.WriteLine("Could not create log group.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateLogGroup](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines neuen Protokoll-Streams

Das folgende Codebeispiel zeigt, wie Sie einen neuen CloudWatch Logs-Protokollstream erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
```



```
/// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch
/// log group.
/// </summary>
public class CreateLogStream
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string logStreamName = "cloudwatchlogs-example-logstream";

        var request = new CreateLogStreamRequest
        {
            LogGroupName = logGroupName,
            LogStreamName = logStreamName,
        };

        var response = await client.CreateLogStreamAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{logStreamName} successfully created for
{logGroupName}.");
        }
        else
        {
            Console.WriteLine("Could not create stream.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateLogStream](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Exportaufgabe

Das folgende Codebeispiel zeigt, wie Sie eine neue CloudWatch Protokolleexportaufgabe erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskName = "export-task-example";
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string destination = "doc-example-bucket";
        var fromTime = 1437584472382;
        var toTime = 1437584472833;

        var request = new CreateExportTaskRequest
        {
            From = fromTime,
            To = toTime,
            TaskName = taskName,
            LogGroupName = logGroupName,
            Destination = destination,
        };
    }
}
```

```
var response = await client.CreateExportTaskAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"The task, {taskName} with ID: " +
        $"{response.TaskId} has been created
successfully.");
}
}
```

- Weitere API-Informationen finden Sie unter [CreateExportTask](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Protokollgruppe

Das folgende Codebeispiel zeigt, wie Sie eine vorhandene CloudWatch Logs-Protokollgruppe löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon CloudWatch Logs Service to delete an existing
/// CloudWatch Logs log group.
/// </summary>
public class DeleteLogGroup
{
    public static async Task Main()
```

```
{
    var client = new AmazonCloudWatchLogsClient();
    string logGroupName = "cloudwatchlogs-example-loggroup";

    var request = new DeleteLogGroupRequest
    {
        LogGroupName = logGroupName,
    };

    var response = await client.DeleteLogGroupAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [DeleteLogGroup](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben von Exportaufgaben

Das folgende Codebeispiel zeigt, wie CloudWatch Sie Protokollexportaufgaben beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
/// <summary>
/// Shows how to retrieve a list of information about Amazon CloudWatch
/// Logs export tasks.
/// </summary>
public class DescribeExportTasks
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        var request = new DescribeExportTasksRequest
        {
            Limit = 5,
        };

        var response = new DescribeExportTasksResponse();

        do
        {
            response = await client.DescribeExportTasksAsync(request);
            response.ExportTasks.ForEach(t =>
            {
                Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has status:
{t.Status}");
            });
        }
        while (response.NextToken is not null);
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeExportTasks](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von Protokollgruppen

Das folgende Codebeispiel zeigt, wie CloudWatch Protokollgruppen beschrieben werden.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Retrieves information about existing Amazon CloudWatch Logs log groups
/// and displays the information on the console.
/// </summary>
public class DescribeLogGroups
{
    public static async Task Main()
    {
        // Creates a CloudWatch Logs client using the default
        // user. If you need to work with resources in another
        // AWS Region than the one defined for the default user,
        // pass the AWS Region as a parameter to the client constructor.
        var client = new AmazonCloudWatchLogsClient();

        bool done = false;
        string newToken = null;

        var request = new DescribeLogGroupsRequest
        {
            Limit = 5,
        };

        DescribeLogGroupsResponse response;

        do
        {
            if (newToken is not null)
            {
                request.NextToken = newToken;
            }
        }
    }
}
```

```
    }

    response = await client.DescribeLogGroupsAsync(request);

    response.LogGroups.ForEach(lg =>
    {
        Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
        Console.WriteLine($"Created on: {lg.CreationTime.Date.Date}");
        Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
    });

    if (response.NextToken is null)
    {
        done = true;
    }
    else
    {
        newToken = response.NextToken;
    }
}
while (!done);
}
}
```

- Weitere API-Informationen finden Sie unter [DescribeLogGroups](#) in der APIAWS SDK for .NET - Referenz für .

Starten einer Live-Tail-Sitzung

Das folgende Codebeispiel zeigt, wie Sie eine Live-Tail-Sitzung für eine vorhandene Protokollgruppe/ einen vorhandenen Protokollstream starten.

AWS SDK for .NET

Binden Sie die erforderlichen Dateien ein.

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Starten Sie die Live-Tail-Sitzung.

```
var client = new AmazonCloudWatchLogsClient();
var request = new StartLiveTailRequest
{
    LogGroupIdentifiers = logGroupIdentifiers,
    LogStreamNames = logStreamNames,
    LogEventFilterPattern = filterPattern,
};

var response = await client.StartLiveTailAsync(request);

// Catch if request fails
if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
{
    Console.WriteLine("Failed to start live tail session");
    return;
}
```

Sie können die Ereignisse aus der Live-Tail-Sitzung auf zwei Arten behandeln:

```
/* Method 1
 * 1). Asynchronously loop through the event stream
 * 2). Set a timer to dispose the stream and stop the Live Tail session
at the end.
*/
var eventStream = response.ResponseStream;
var task = Task.Run(() =>
{
    foreach (var item in eventStream)
    {
        if (item is LiveTailSessionUpdate liveTailSessionUpdate)
        {
            foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
            {
                Console.WriteLine("Message : {0}",
sessionResult.Message);
            }
        }
    }
}
```



```

        if (item is LiveTailSessionStart)
        {
            Console.WriteLine("Live Tail session started");
        }
        // On-stream exceptions are processed here
        if (item is CloudWatchLogsEventStreamException)
        {
            Console.WriteLine($"ERROR: {item}");
        }
    }
});
// Close the stream to stop the session after a timeout
if (!task.Wait(TimeSpan.FromSeconds(10))){
    eventStream.Dispose();
    Console.WriteLine("End of line");
}

```

```

/* Method 2
 * 1). Add event handlers to each event variable
 * 2). Start processing the stream and wait for a timeout using
AutoResetEvent
*/
AutoResetEvent endEvent = new AutoResetEvent(false);
var eventStream = response.ResponseStream;
using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
{
    eventStream.SessionStartReceived += (sender, e) =>
    {
        Console.WriteLine("LiveTail session started");
    };
    eventStream.SessionUpdateReceived += (sender, e) =>
    {
        foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
            Console.WriteLine("Message: {0}", logEvent.Message);
        }
    };
    // On-stream exceptions are captured here
    eventStream.ExceptionReceived += (sender, e) =>
    {
        Console.WriteLine($"ERROR: {e.EventStreamException.Message}");
    };
}

```

```
};

    eventStream.StartProcessing();
    // Stream events for this amount of time.
    endEvent.WaitOne(TimeSpan.FromSeconds(10));
    Console.WriteLine("End of line");
}
```

- Weitere API-Informationen finden Sie unter [StartLiveTail](#) in der APIAWS SDK for .NET - Referenz für .

Beispiele für Amazon Cognito-Identitätsanbieter mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET 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 Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Bestätigen eines Benutzers

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Cognito-Benutzer bestätigen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Confirm that the user has signed up.
/// </summary>
/// <param name="clientId">The Id of this application.</param>
/// <param name="code">The confirmation code sent to the user.</param>
/// <param name="userName">The username.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
{
    var signUpRequest = new ConfirmSignUpRequest
    {
        ClientId = clientId,
        ConfirmationCode = code,
        Username = userName,
    };

    var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
    if (response.HttpStatusCode == HttpStatusCode.OK)
    {
        Console.WriteLine($"{userName} was confirmed");
        return true;
    }
    return false;
}
```

- Weitere API-Informationen finden Sie unter [ConfirmSignUp](#) in der APIAWS SDK for .NET - Referenz für .

Bestätigen eines MFA-Geräts für die Nachverfolgung

Das folgende Codebeispiel zeigt, wie Sie ein MFA-Gerät für die Nachverfolgung durch Amazon Cognito bestätigen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Initiates and confirms tracking of the device.
/// </summary>
/// <param name="accessToken">The user's access token.</param>
/// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
/// <param name="deviceName">The device name.</param>
/// <returns></returns>
public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
{
    var request = new ConfirmDeviceRequest
    {
        AccessToken = accessToken,
        DeviceKey = deviceKey,
        DeviceName = deviceName
    };

    var response = await _cognitoService.ConfirmDeviceAsync(request);
    return response.UserConfirmationNecessary;
}
```

- Weitere API-Informationen finden Sie unter [ConfirmDevice](#) in der APIAWS SDK for .NET - Referenz für .

Abruf eines Tokens, um eine MFA-Anwendung einem Benutzer zuzuordnen

Das folgende Codebeispiel zeigt, wie Sie ein Token abrufen, um eine MFA-Anwendung einem Amazon Cognito-Benutzer zuzuordnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get an MFA token to authenticate the user with the authenticator.
/// </summary>
/// <param name="session">The session name.</param>
/// <returns>The session name.</returns>
public async Task<string> AssociateSoftwareTokenAsync(string session)
{
    var softwareTokenRequest = new AssociateSoftwareTokenRequest
    {
        Session = session,
    };

    var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
    var secretCode = tokenResponse.SecretCode;

    Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

    return tokenResponse.Session;
}
```

- Weitere API-Informationen finden Sie unter [AssociateSoftwareToken](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Informationen über einen Benutzer

Das folgende Codebeispiel zeigt, wie Sie Informationen zu einem Amazon Cognito-Benutzer abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the specified user from an Amazon Cognito user pool with administrator
access.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
/// <returns>Async task.</returns>
public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
{
    AdminGetUserRequest userRequest = new AdminGetUserRequest
    {
        Username = userName,
        UserPoolId = poolId,
    };

    var response = await _cognitoService.AdminGetUserAsync(userRequest);

    Console.WriteLine($"User status {response.UserStatus}");
    return response.UserStatus;
}
```

- Weitere API-Informationen finden Sie unter [AdminGetUser](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten der Benutzerpools

Das folgende Code-Beispiel veranschaulicht, wie Amazon Cognito-Benutzerpools aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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;
}
```

- Weitere API-Informationen finden Sie unter [ListUserPools](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Benutzern

Das folgende Codebeispiel zeigt, wie Sie Amazon Cognito-Benutzer auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of users for the Amazon Cognito user pool.
/// </summary>
/// <param name="userPoolId">The user pool ID.</param>
/// <returns>A list of users.</returns>
public async Task<List<UserType>> ListUsersAsync(string userPoolId)
{
    var request = new ListUsersRequest
    {
        UserPoolId = userPoolId
    };

    var users = new List<UserType>();

    var usersPaginator = _cognitoService.Paginators.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- Weitere API-Informationen finden Sie unter [ListUsers](#) in der APIAWS SDK for .NET -Referenz für .

Erneutes Senden eines Bestätigungscode

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Cognito-Bestätigungscode erneut senden.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Send a new confirmation code to a user.
/// </summary>
/// <param name="clientId">The Id of the client application.</param>
/// <param name="userName">The username of user who will receive the code.</
param>
/// <returns>The delivery details.</returns>
public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
clientId, string userName)
{
    var codeRequest = new ResendConfirmationCodeRequest
    {
        ClientId = clientId,
        Username = userName,
    };

    var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);

    Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

    return response.CodeDeliveryDetails;
}
```

- Weitere API-Informationen finden Sie unter [ResendConfirmationCode](#) in der APIAWS SDK for .NET -Referenz für .

Reagieren auf eine Amazon Cognito SRP-Authentifizierungsherausforderung

Das folgende Codebeispiel zeigt, wie Sie auf eine Amazon Cognito-Authentifizierungsaufforderung reagieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
    challengeResponses.Add("USERNAME", userName);
    challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
    {
        ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
        ClientId = clientId,
        ChallengeResponses = challengeResponses,
        Session = session,
        UserPoolId = userPoolId,
    };
}
```

```

    };

    var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
    Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
    return response.AuthenticationResult;
}

```

- Weitere API-Informationen finden Sie unter [AdminRespondToAuthChallenge](#) in der APIAWS SDK for .NET -Referenz für .

Registrieren eines Benutzers

Das folgende Codebeispiel zeigt, wie Sie einen Benutzer mit Amazon Cognito registrieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Sign up a new user.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The username to use.</param>
/// <param name="password">The user's password.</param>
/// <param name="email">The email address of the user.</param>
/// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
public async Task<bool> SignUpAsync(string clientId, string userName, string
password, string email)
{
    var userAttrs = new AttributeType
    {
        Name = "email",
        Value = email,

```

```
};

var userAttrsList = new List<AttributeType>();

userAttrsList.Add(userAttrs);

var signUpRequest = new SignUpRequest
{
    UserAttributes = userAttrsList,
    Username = userName,
    ClientId = clientId,
    Password = password
};

var response = await _cognitoService.SignUpAsync(signUpRequest);
return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [SignUp](#) in der APIAWS SDK for .NET -Referenz für

Starten der Authentifizierung

Das folgende Codebeispiel zeigt, wie Sie die Authentifizierung mit Amazon Cognito starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Initiate authorization.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The name of the user who is authenticating.</param>
/// <param name="password">The password for the user who is authenticating.</
param>
```

```
/// <returns>The response from the initiate auth request.</returns>
public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var authRequest = new InitiateAuthRequest
    {
        ClientId = clientId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.InitiateAuthAsync(authRequest);
    Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

    return response;
}
```

- Weitere API-Informationen finden Sie unter [InitiateAuth](#) in der APIAWS SDK for .NET -Referenz für .

Starten der Authentifizierung mit Administratoranmeldeinformationen

Das folgende Codebeispiel zeigt, wie Sie die Authentifizierung mit Amazon Cognito und Administratoranmeldeinformationen starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Initiate an admin auth request.
```

```
/// </summary>
/// <param name="clientId">The client ID to use.</param>
/// <param name="userPoolId">The ID of the user pool.</param>
/// <param name="userName">The username to authenticate.</param>
/// <param name="password">The user's password.</param>
/// <returns>The session to use in challenge-response.</returns>
public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var request = new AdminInitiateAuthRequest
    {
        ClientId = clientId,
        UserPoolId = userPoolId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.AdminInitiateAuthAsync(request);
    return response.Session;
}
```

- Weitere API-Informationen finden Sie unter [AdminInitiateAuth](#) in der APIAWS SDK for .NET - Referenz für .

Verifizieren einer MFA-Anwendung mit einem Benutzer

Das folgende Codebeispiel zeigt, wie Sie eine MFA-Anwendung mit einem Amazon Cognito-Benutzer verifizieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Verify the TOTP and register for MFA.
/// </summary>
/// <param name="session">The name of the session.</param>
/// <param name="code">The MFA code.</param>
/// <returns>The status of the software token.</returns>
public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
{
    var tokenRequest = new VerifySoftwareTokenRequest
    {
        UserCode = code,
        Session = session,
    };

    var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

    return verifyResponse.Status;
}
```

- Weitere API-Informationen finden Sie unter [VerifySoftwareToken](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Registrieren eines Benutzers bei einem Benutzerpool, der MFA erfordert

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Registrieren Sie einen Benutzer mit einem Benutzernamen, einem Passwort und einer E-Mail-Adresse und bestätigen Sie ihn.
- Einrichten der Multi-Faktor-Authentifizierung durch Zuordnung einer MFA-Anwendung zu dem Benutzer.
- Anmelden unter Verwendung eines Passworts und eines MFA-Codes.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Cognito.
        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<IAmazonCognitoIdentityProvider>()
                    .AddTransient<CognitoWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CognitoBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

        Console.WriteLine(new string('-', 80));
    }
}
```



```
UiMethods.DisplayOverview();
Console.WriteLine(new string('-', 80));

// clientId - The app client Id value that you get from the AWS CDK script.
var clientId = configuration["ClientId"]; // **** REPLACE WITH CLIENT ID
VALUE FROM CDK SCRIPT";

// poolId - The pool Id that you get from the AWS CDK script.
var poolId = configuration["PoolId"]!; // **** REPLACE WITH POOL ID VALUE
FROM CDK SCRIPT";
var userName = configuration["UserName"];
var password = configuration["Password"];
var email = configuration["Email"];

// If the username wasn't set in the configuration file,
// get it from the user now.
if (userName is null)
{
    do
    {
        Console.Write("Username: ");
        userName = Console.ReadLine();
    }
    while (string.IsNullOrEmpty(userName));
}
Console.WriteLine($"\\nUsername: {userName}");

// If the password wasn't set in the configuration file,
// get it from the user now.
if (password is null)
{
    do
    {
        Console.Write("Password: ");
        password = Console.ReadLine();
    }
    while (string.IsNullOrEmpty(password));
}

// If the email address wasn't set in the configuration file,
// get it from the user now.
if (email is null)
{
    do
```

```
    {
        Console.WriteLine("Email: ");
        email = Console.ReadLine();
    } while (string.IsNullOrEmpty(email));
}

// Now sign up the user.
Console.WriteLine($"\\nSigning up {userName} with email address: {email}");
await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

// Add the user to the user pool.
Console.WriteLine($"Adding {userName} to the user pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

UiMethods.DisplayTitle("Get confirmation code");
Console.WriteLine($"Conformation code sent to {userName}.");
Console.WriteLine("Would you like to send a new code? (Y/N) ");
var answer = Console.ReadLine();

if (answer!.ToLower() == "y")
{
    await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
    Console.WriteLine("Sending a new confirmation code");
}

Console.WriteLine("Enter confirmation code (from Email): ");
var code = Console.ReadLine();

await cognitoWrapper.ConfirmSignUpAsync(clientId, code, userName);

UiMethods.DisplayTitle("Checking status");
Console.WriteLine($"Rechecking the status of {userName} in the user pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

Console.WriteLine($"Setting up authenticator for {userName} in the user
pool");
var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
userName, password);

var setupSession = await
cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
Console.WriteLine("Enter the 6-digit code displayed in Google Authenticator: ");
var setupCode = Console.ReadLine();
```

```
        var setupResult = await
cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
        Console.WriteLine($"Setup status: {setupResult}");

        Console.WriteLine($"Now logging in {userName} in the user pool");
        var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
poolId, userName, password);

        Console.Write("Enter a new 6-digit code displayed in Google Authenticator:
");
        var authCode = Console.ReadLine();

        var authResult = await
cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
authSession, poolId);
        Console.WriteLine($"Authenticated and received access token:
{authResult.AccessToken}");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cognito scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
}

using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
    /// Constructor for the wrapper class containing Amazon Cognito actions.
    /// </summary>
    /// <param name="cognitoService">The Amazon Cognito client object.</param>
    public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)
    {
        _cognitoService = cognitoService;
    }
}
```

```
/// <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;
}

/// <summary>
/// Get a list of users for the Amazon Cognito user pool.
/// </summary>
/// <param name="userPoolId">The user pool ID.</param>
/// <returns>A list of users.</returns>
public async Task<List<UserType>> ListUsersAsync(string userPoolId)
{
    var request = new ListUsersRequest
    {
        UserPoolId = userPoolId
    };

    var users = new List<UserType>();

    var usersPaginator = _cognitoService.Paginators.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

```
/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
    challengeResponses.Add("USERNAME", userName);
    challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
    {
        ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
        ClientId = clientId,
        ChallengeResponses = challengeResponses,
        Session = session,
        UserPoolId = userPoolId,
    };

    var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
    Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
    return response.AuthenticationResult;
}

/// <summary>
/// Verify the TOTP and register for MFA.
/// </summary>
/// <param name="session">The name of the session.</param>
```

```
/// <param name="code">The MFA code.</param>
/// <returns>The status of the software token.</returns>
public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
{
    var tokenRequest = new VerifySoftwareTokenRequest
    {
        UserCode = code,
        Session = session,
    };

    var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

    return verifyResponse.Status;
}

/// <summary>
/// Get an MFA token to authenticate the user with the authenticator.
/// </summary>
/// <param name="session">The session name.</param>
/// <returns>The session name.</returns>
public async Task<string> AssociateSoftwareTokenAsync(string session)
{
    var softwareTokenRequest = new AssociateSoftwareTokenRequest
    {
        Session = session,
    };

    var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
    var secretCode = tokenResponse.SecretCode;

    Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

    return tokenResponse.Session;
}

/// <summary>
/// Initiate an admin auth request.
/// </summary>
```

```
/// <param name="clientId">The client ID to use.</param>
/// <param name="userPoolId">The ID of the user pool.</param>
/// <param name="userName">The username to authenticate.</param>
/// <param name="password">The user's password.</param>
/// <returns>The session to use in challenge-response.</returns>
public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var request = new AdminInitiateAuthRequest
    {
        ClientId = clientId,
        UserPoolId = userPoolId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.AdminInitiateAuthAsync(request);
    return response.Session;
}

/// <summary>
/// Initiate authorization.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The name of the user who is authenticating.</param>
/// <param name="password">The password for the user who is authenticating.</
param>
/// <returns>The response from the initiate auth request.</returns>
public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var authRequest = new InitiateAuthRequest
    {
        ClientId = clientId,
        AuthParameters = authParameters,
```

```
        AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.InitiateAuthAsync(authRequest);
    Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

    return response;
}

/// <summary>
/// Confirm that the user has signed up.
/// </summary>
/// <param name="clientId">The Id of this application.</param>
/// <param name="code">The confirmation code sent to the user.</param>
/// <param name="userName">The username.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
{
    var signUpRequest = new ConfirmSignUpRequest
    {
        ClientId = clientId,
        ConfirmationCode = code,
        Username = userName,
    };

    var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
    if (response.HttpStatusCode == HttpStatusCode.OK)
    {
        Console.WriteLine($"{userName} was confirmed");
        return true;
    }
    return false;
}

/// <summary>
/// Initiates and confirms tracking of the device.
/// </summary>
/// <param name="accessToken">The user's access token.</param>
/// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
/// <param name="deviceName">The device name.</param>
/// <returns></returns>
```



```
public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
{
    var request = new ConfirmDeviceRequest
    {
        AccessToken = accessToken,
        DeviceKey = deviceKey,
        DeviceName = deviceName
    };

    var response = await _cognitoService.ConfirmDeviceAsync(request);
    return response.UserConfirmationNecessary;
}

/// <summary>
/// Send a new confirmation code to a user.
/// </summary>
/// <param name="clientId">The Id of the client application.</param>
/// <param name="userName">The username of user who will receive the code.</
param>
/// <returns>The delivery details.</returns>
public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
clientId, string userName)
{
    var codeRequest = new ResendConfirmationCodeRequest
    {
        ClientId = clientId,
        Username = userName,
    };

    var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);

    Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

    return response.CodeDeliveryDetails;
}

/// <summary>
/// Get the specified user from an Amazon Cognito user pool with administrator
access.
```

```
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
/// <returns>Async task.</returns>
public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
{
    AdminGetUserRequest userRequest = new AdminGetUserRequest
    {
        Username = userName,
        UserPoolId = poolId,
    };

    var response = await _cognitoService.AdminGetUserAsync(userRequest);

    Console.WriteLine($"User status {response.UserStatus}");
    return response.UserStatus;
}

/// <summary>
/// Sign up a new user.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The username to use.</param>
/// <param name="password">The user's password.</param>
/// <param name="email">The email address of the user.</param>
/// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
public async Task<bool> SignUpAsync(string clientId, string userName, string
password, string email)
{
    var userAttrs = new AttributeType
    {
        Name = "email",
        Value = email,
    };

    var userAttrsList = new List<AttributeType>();

    userAttrsList.Add(userAttrs);

    var signUpRequest = new SignUpRequest
    {
```

```
        UserAttributes = userAttrsList,
        Username = userName,
        ClientId = clientId,
        Password = password
    };

    var response = await _cognitoService.SignUpAsync(signUpRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Amazon Comprehend-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Comprehend verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Erkennen von Entitäten in einem Dokument

Das folgende Codebeispiel zeigt, wie Entitäten in einem Dokument mit Amazon Comprehend erkannt werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to use the AmazonComprehend service detect any
/// entities in submitted text.
/// </summary>
public static class DetectEntities
{
    /// <summary>
    /// The main method calls the DetectEntitiesAsync method to find any
    /// entities in the sample code.
    /// </summary>
    public static async Task Main()
```

```
{
    string text = "It is raining today in Seattle";

    var comprehendClient = new AmazonComprehendClient();

    Console.WriteLine("Calling DetectEntities\n");
    var detectEntitiesRequest = new DetectEntitiesRequest()
    {
        Text = text,
        LanguageCode = "en",
    };
    var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

    foreach (var e in detectEntitiesResponse.Entities)
    {
        Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
    }

    Console.WriteLine("Done");
}
}
```

- Weitere API-Informationen finden Sie unter [DetectEntities](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen von Schlüsselphrasen in einem Dokument

Das folgende Codebeispiel zeigt, wie Sie Schlüsselphrasen in einem Dokument mit Amazon Comprehend erkennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to use the Amazon Comprehend service to
/// search text for key phrases.
/// </summary>
public static class DetectKeyPhrase
{
    /// <summary>
    /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync
    /// to detect any key phrases in the sample text.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

        // Call DetectKeyPhrases API
        Console.WriteLine("Calling DetectKeyPhrases");
        var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
        {
            Text = text,
            LanguageCode = "en",
        };
        var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
        foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
        {
            Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score}, BeginOffset:
{kp.BeginOffset}, EndOffset: {kp.EndOffset}");
        }

        Console.WriteLine("Done");
    }
}
```

- Weitere API-Informationen finden Sie unter [DetectKeyPhrases](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen von persönlich identifizierbaren Informationen in einem Dokument

Das folgende Codebeispiel zeigt, wie Sie persönlich identifizierbare Informationen (PII) in einem Dokument mit Amazon Comprehend erkennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally identifiable information within the supplied text.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();
        var text = @"Hello Paul Santos. The latest statement for your
                    credit card account 1111-0000-1111-0000 was
                    mailed to 123 Any Street, Seattle, WA 98109.";

        var request = new DetectPiiEntitiesRequest
        {
```

```
        Text = text,
        LanguageCode = "EN",
    };

    var response = await comprehendClient.DetectPiiEntitiesAsync(request);

    if (response.Entities.Count > 0)
    {
        foreach (var entity in response.Entities)
        {
            var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
            Console.WriteLine($"{entity.Type}: {entityValue}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [DetectPiiEntities](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen syntaktischer Elemente eines Dokuments

Das folgende Codebeispiel zeigt, wie Sie syntaktische Elemente eines Dokuments mit Amazon Comprehend erkennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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



```
/// <summary>
/// This example shows how to use Amazon Comprehend to detect syntax
/// elements by calling the DetectSyntaxAsync method.
/// </summary>
public class DetectingSyntax
{
    /// <summary>
    /// This method calls DetectSynaxAsync to identify the syntax elements
    /// in the sample text.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new AmazonComprehendClient();

        // Call DetectSyntax API
        Console.WriteLine("Calling DetectSyntaxAsync\n");
        var detectSyntaxRequest = new DetectSyntaxRequest()
        {
            Text = text,
            LanguageCode = "en",
        };
        DetectSyntaxResponse detectSyntaxResponse = await
comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
        foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
        {
            Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
{s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
        }

        Console.WriteLine("Done");
    }
}
```

- Weitere API-Informationen finden Sie unter [DetectSyntax](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen der dominanten Sprache in einem Dokument

Das folgende Codebeispiel zeigt, wie Sie die vorherrschende Sprache in einem Dokument mit Amazon Comprehend erkennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example calls the Amazon Comprehend service to determine the
/// dominant language.
/// </summary>
public static class DetectDominantLanguage
{
    /// <summary>
    /// Calls Amazon Comprehend to determine the dominant language used in
    /// the sample text.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle.";

        var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

        Console.WriteLine("Calling DetectDominantLanguage\n");
        var detectDominantLanguageRequest = new DetectDominantLanguageRequest()
        {
            Text = text,
        };
    }
}
```

```
        var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
        foreach (var dl in detectDominantLanguageResponse.Languages)
        {
            Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
        }

        Console.WriteLine("Done");
    }
}
```

- Weitere API-Informationen finden Sie unter [DetectDominantLanguage](#) in der APIAWS SDK for .NET -Referenz für .

Erkennen der Stimmung eines Dokuments

Das folgende Codebeispiel zeigt, wie Sie die Stimmung eines Dokuments mit Amazon Comprehend erkennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to detect the overall sentiment of the supplied
/// text using the Amazon Comprehend service.
/// </summary>
public static class DetectSentiment
{
```

```
/// <summary>
/// This method calls the DetectSentimentAsync method to analyze the
/// supplied text and determine the overall sentiment.
/// </summary>
public static async Task Main()
{
    string text = "It is raining today in Seattle";

    var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

    // Call DetectKeyPhrases API
    Console.WriteLine("Calling DetectSentiment");
    var detectSentimentRequest = new DetectSentimentRequest()
    {
        Text = text,
        LanguageCode = "en",
    };
    var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
    Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
    Console.WriteLine("Done");
}
}
```

- Weitere API-Informationen finden Sie unter [DetectSentiment](#) in der APIAWS SDK for .NET - Referenz für .

Starten eines Themenmodellierungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Comprehend-Themenmodellierungsauftrag starten.

AWS SDK for .NET

Note

Auf [GitHub](#) gibt es mehr Beispiele. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example scans the documents in an Amazon Simple Storage Service
/// (Amazon S3) bucket and analyzes it for topics. The results are stored
/// in another bucket and then the resulting job properties are displayed
/// on the screen. This example was created using the AWS SDK for .NET
/// version 3.7 and .NET Core version 5.0.
/// </summary>
public static class TopicModeling
{
    /// <summary>
    /// This methos calls a topic detection job by calling the Amazon
    /// Comprehend StartTopicsDetectionJobRequest.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();

        string inputS3Uri = "s3://input bucket/input path";
        InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
        string outputS3Uri = "s3://output bucket/output path";
        string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";

        int numberOfTopics = 10;

        var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
        {
            InputDataConfig = new InputDataConfig()
            {
                S3Uri = inputS3Uri,
                InputFormat = inputDocFormat,
            },
            OutputDataConfig = new OutputDataConfig()
            {
                S3Uri = outputS3Uri,
            },
            DataAccessRoleArn = dataAccessRoleArn,
            NumberOfTopics = numberOfTopics,
        }
    }
}
```

```
};

    var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

    var jobId = startTopicsDetectionJobResponse.JobId;
    Console.WriteLine("JobId: " + jobId);

    var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
    {
        JobId = jobId,
    };

    var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);
PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

    var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
    foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
    {
        PrintJobProperties(props);
    }
}

/// <summary>
/// This method is a helper method that displays the job properties
/// from the call to StartTopicsDetectionJobRequest.
/// </summary>
/// <param name="props">A list of properties from the call to
/// StartTopicsDetectionJobRequest.</param>
private static void PrintJobProperties(TopicsDetectionJobProperties props)
{
    Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
    Console.WriteLine($"NumberOfTopics: {props.NumberOfTopics}\nInputS3Uri:
{props.InputDataConfig.S3Uri}");
    Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [StartTopicsDetectionJob](#) in der APIAWS SDK for .NET -Referenz für .

DynamoDB-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit DynamoDB verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo DynamoDB

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit DynamoDB.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.Model;  
  
namespace DynamoDB_Actions;
```

```
public static class HelloDynamoDB
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();

        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListTables](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie eine DynamoDB-Tabelle erstellt wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
    {
        var response = await client.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,
    },
},
ProvisionedThroughput = new ProvisionedThroughput
{
    ReadCapacityUnits = 5,
    WriteCapacityUnits = 5,
},
});

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("Waiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = response.TableDescription.TableName,
};

TableStatus status;

int sleepDuration = 2000;

do
{
    System.Threading.Thread.Sleep(sleepDuration);

    var describeTableResponse = await
client.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
```

- Weitere API-Informationen finden Sie unter [CreateTable](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie eine DynamoDB-Tabelle gelöscht wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
{
    var request = new DeleteTableRequest
    {
        TableName = tableName,
    };

    var response = await client.DeleteTableAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
        return true;
    }
    else
    {
        Console.WriteLine("Could not delete table.");
        return false;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteTable](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Elements aus einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie ein Element aus einer DynamoDB-Tabelle gelöscht wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
    Movie movieToDelete)
{
    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,
    };

    var response = await client.DeleteItemAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [Deleteltem](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen eines Element-Batchs

Im folgenden Codebeispiel wird gezeigt, wie ein DynamoDB-Element-Batch abgerufen wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient
client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
                {
                    { _table1Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
                    }
                }
            }
        }
    }
}
```

```

        new Dictionary<string, AttributeValue>()
        {
            { "Name", new AttributeValue {
                S = "Amazon DynamoDB"
            } }
        },
        new Dictionary<string, AttributeValue>()
        {
            { "Name", new AttributeValue {
                S = "Amazon S3"
            } }
        }
    }
}],
{
    _table2Name,
    new KeysAndAttributes
    {
        Keys = new List<Dictionary<string, AttributeValue> >()
        {
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon DynamoDB"
                } },
                { "Subject", new AttributeValue {
                    S = "DynamoDB Thread 1"
                } }
            },
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon DynamoDB"
                } },
                { "Subject", new AttributeValue {
                    S = "DynamoDB Thread 2"
                } }
            },
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon S3"
                } },
                { "Subject", new AttributeValue {

```

```
                S = "S3 Thread 1"
            } }
        }
    }
}
};

BatchGetItemResponse response;
do
{
    Console.WriteLine("Making request");
    response = await client.BatchGetItemAsync(request);

    // Check the response.
    var responses = response.Responses; // Attribute list in the
response.

    foreach (var tableResponse in responses)
    {
        var tableResults = tableResponse.Value;
        Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
        foreach (var item1 in tableResults)
        {
            PrintItem(item1);
        }
    }

    // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
    Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
    foreach (var unprocessedTableKeys in unprocessedKeys)
    {
        // Print table name.
        Console.WriteLine(unprocessedTableKeys.Key);
        // Print unprocessed primary keys.
        foreach (var key in unprocessedTableKeys.Value.Keys)
        {
            PrintItem(key);
        }
    }
}
```

```

        request.RequestItems = unprocessedKeys;
    } while (response.UnprocessedKeys.Count > 0);
}

private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
{
    foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
    {
        string attributeName = kvp.Key;
        AttributeValue value = kvp.Value;

        Console.WriteLine(
            attributeName + " " +
            (value.S == null ? "" : "S=[" + value.S + "]") +
            (value.N == null ? "" : "N=[" + value.N + "]") +
            (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
            (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]")
            );
    }
    Console.WriteLine("*****");
}

static void Main()
{
    var client = new AmazonDynamoDBClient();

    RetrieveMultipleItemsBatchGet(client);
}
}
}

```

- Weitere API-Informationen finden Sie unter [BatchGetItem](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen eines Elements aus einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie ein Element aus einer DynamoDB-Tabelle abgerufen wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<Dictionary<string, AttributeValue>>
GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
    {
        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 client.GetItemAsync(request);
        return response.Item;
    }
```

- Weitere API-Informationen finden Sie unter [GetItem](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Informationen zu einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie Informationen zu einer DynamoDB-Tabelle abgerufen werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });

    var table = response.Table;
    Console.WriteLine($"Name: {table.TableName}");
    Console.WriteLine($"# of items: {table.ItemCount}");
    Console.WriteLine($"Provision Throughput (reads/sec): " +
        $"{table.ProvisionedThroughput.ReadCapacityUnits}");
    Console.WriteLine($"Provision Throughput (writes/sec): " +
        $"{table.ProvisionedThroughput.WriteCapacityUnits}");
}
```

- Weitere API-Informationen finden Sie unter [DescribeTable](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Tabellen

Im folgenden Codebeispiel wird gezeigt, wie DynamoDB-Tabellen aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });

        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- Weitere API-Informationen finden Sie unter [ListTables](#) in der APIAWS SDK for .NET -Referenz für .

Einfügen eines Elements in eine Tabelle

Im folgenden Codebeispiel wird gezeigt, wie Sie ein Element in eine DynamoDB-Tabelle einfügen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="newMovie">A Movie object containing information 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
Movie newMovie, string tableName)
    {
        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,
        };

        var response = await client.PutItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Weitere API-Informationen finden Sie unter [PutItem](#) in der APIAWS SDK for .NET -Referenz für

Abfragen einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie DynamoDB-Tabellen abgefragt werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
{
    var movieTable = Table.LoadTable(client, tableName);
    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;

    Search 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;
}
```

- Weitere API-Informationen finden Sie unter [Query](#) in der AWS SDK for .NET -API-Referenz.

Ausführen einer PartiQL-Anweisung

Das folgende Codebeispiel zeigt, wie Sie eine PartiQL-Anweisung für eine DynamoDB-Tabelle ausführen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie eine INSERT-Anweisung, um ein Element hinzuzufügen.

```
/// <summary>
/// Inserts a single movie into the movies table.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="movieTitle">The title of the movie to insert.</param>
/// <param name="year">The year that the movie was released.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the INSERT operation.</returns>
public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
{
    string insertBatch = $"INSERT INTO {tableName} VALUE {'title': ?,
'year': ?}";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = insertBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

Verwenden Sie eine SELECT-Anweisung, um ein Element abzurufen.

```
/// <summary>
/// Uses a PartiQL SELECT statement to retrieve a single movie from the
/// movie database.
/// </summary>
/// <param name="tableName">The name of the movie table.</param>
/// <param name="movieTitle">The title of the movie to retrieve.</param>
/// <returns>A list of movie data. If no movie matches the supplied
/// title, the list is empty.</returns>
```

```

    public static async Task<List<Dictionary<string, AttributeValue>>>
    GetSingleMovie(string tableName, string movieTitle)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
        };

        var response = await Client.ExecuteStatementAsync(new
    ExecuteStatementRequest
        {
            Statement = selectSingle,
            Parameters = parameters,
        });

        return response.Items;
    }

```

Verwenden einer SELECT-Anweisung, um ein Liste an Elementen abzurufen.

```

    /// <summary>
    /// Retrieve multiple movies by year using a SELECT statement.
    /// </summary>
    /// <param name="tableName">The name of the movie table.</param>
    /// <param name="year">The year the movies were released.</param>
    /// <returns></returns>
    public static async Task<List<Dictionary<string, AttributeValue>>>
    GetMovies(string tableName, int year)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { N = year.ToString() },
        };

        var response = await Client.ExecuteStatementAsync(new
    ExecuteStatementRequest
        {
            Statement = selectSingle,

```



```

        Parameters = parameters,
    });

    return response.Items;
}

```

Verwenden Sie eine UPDATE-Anweisung, um ein Element zu aktualisieren.

```

/// <summary>
/// Updates a single movie in the table, adding information for the
/// producer.
/// </summary>
/// <param name="tableName">the name of the table.</param>
/// <param name="producer">The name of the producer.</param>
/// <param name="movieTitle">The movie title.</param>
/// <param name="year">The year the movie was released.</param>
/// <returns>A Boolean value that indicates the success of the
/// UPDATE operation.</returns>
public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
{
    string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = insertSingle,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = producer },
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Verwenden einer DELETE-Anweisung, um einen einzelnen Film zu löschen.

```
/// <summary>
/// Deletes a single movie from the table.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="movieTitle">The title of the movie to delete.</param>
/// <param name="year">The year that the movie was released.</param>
/// <returns>A Boolean value that indicates the success of the
/// DELETE operation.</returns>
public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
{
    var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = deleteSingle,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [ExecuteStatement](#) in der APIAWS SDK for .NET - Referenz für .

Ausführen von PartiQL-Anweisungstapel

Das folgende Codebeispiel zeigt, wie Stapel von PartiQL-Anweisungen in einer DynamoDB-Tabelle ausgeführt werden.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie Stapel von INSERT-Anweisungen, um Elemente hinzuzufügen.

```
/// <summary>
/// Inserts movies imported from a JSON file into the movie table by
/// using an Amazon DynamoDB PartiQL INSERT statement.
/// </summary>
/// <param name="tableName">The name of the table into which the movie
/// information will be inserted.</param>
/// <param name="movieFileName">The name of the JSON file that contains
/// movie information.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the insert operation.</returns>
public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
{
    // Get the list of movies from the JSON file.
    var movies = ImportMovies(movieFileName);

    var success = false;

    if (movies is not null)
    {
        // Insert the movies in a batch using PartiQL. Because the
        // batch can contain a maximum of 25 items, insert 25 movies
        // at a time.
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
        var statements = new List<BatchStatementRequest>();

        try
        {
            for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
            {
                for (var i = indexOffset; i < indexOffset + 25; i++)
                {
```

```
        statements.Add(new BatchStatementRequest
        {
            Statement = insertBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movies[i].Title },
                new AttributeValue { N =
movies[i].Year.ToString() },
            },
        });
    }

    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
    {
        Statements = statements,
    });

    // Wait between batches for movies to be successfully added.
    System.Threading.Thread.Sleep(3000);

    success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

    // Clear the list of statements for the next batch.
    statements.Clear();
}
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine(ex.Message);
}
}

return success;
}

/// <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 full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
```

```

    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        if (allMovies is not null)
        {
            // Return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }
        else
        {
            return null!;
        }
    }
}

```

Verwenden Sie Stapel von SELECT-Anweisungen, um Elemente abzurufen.

```

/// <summary>
/// Gets movies from the movie table by
/// using an Amazon DynamoDB PartiQL SELECT statement.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year1">The year of the first movie.</param>
/// <param name="year2">The year of the second movie.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> GetBatch(
    string tableName,
    string title1,
    string title2,
    int year1,
    int year2)
{
    var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
    var statements = new List<BatchStatementRequest>

```

```
    {
        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title1 },
                new AttributeValue { N = year1.ToString() },
            },
        },

        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title2 },
                new AttributeValue { N = year2.ToString() },
            },
        }
    };

    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
    {
        Statements = statements,
    });

    if (response.Responses.Count > 0)
    {
        response.Responses.ForEach(r =>
        {
            Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
        });
        return true;
    }
    else
    {
        Console.WriteLine($"Couldn't find either {title1} or {title2}.");
        return false;
    }
}
```

Verwenden Sie Stapel von UPDATE-Anweisungen, um Elemente zu aktualisieren.

```

    /// <summary>
    /// Updates information for multiple movies.
    /// </summary>
    /// <param name="tableName">The name of the table containing the
    /// movies to be updated.</param>
    /// <param name="producer1">The producer name for the first movie
    /// to update.</param>
    /// <param name="title1">The title of the first movie.</param>
    /// <param name="year1">The year that the first movie was released.</param>
    /// <param name="producer2">The producer name for the second
    /// movie to update.</param>
    /// <param name="title2">The title of the second movie.</param>
    /// <param name="year2">The year that the second movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the update.</
returns>
    public static async Task<bool> UpdateBatch(
        string tableName,
        string producer1,
        string title1,
        int year1,
        string producer2,
        string title2,
        int year2)
    {
        string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
        var statements = new List<BatchStatementRequest>
        {
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer1 },
                    new AttributeValue { S = title1 },
                    new AttributeValue { N = year1.ToString() },
                },
            },
        },
    },

```

```

        new BatchStatementRequest
        {
            Statement = updateBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer2 },
                new AttributeValue { S = title2 },
                new AttributeValue { N = year2.ToString() },
            },
        },
    };

    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
    {
        Statements = statements,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Verwenden Sie Stapel von DELETE-Anweisungen, um Elemente zu löschen.

```

/// <summary>
/// Deletes multiple movies using a PartiQL BatchExecuteAsync
/// statement.
/// </summary>
/// <param name="tableName">The name of the table containing the
/// moves that will be deleted.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="year1">The year the first movie was released.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year2">The year the second movie was released.</param>
/// <returns>A Boolean value indicating the success of the operation.</
returns>
public static async Task<bool> DeleteBatch(
    string tableName,
    string title1,
    int year1,
    string title2,
    int year2)

```



```

    {
        string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";
        var statements = new List<BatchStatementRequest>
        {
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = title1 },
                    new AttributeValue { N = year1.ToString() },
                },
            },
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = title2 },
                    new AttributeValue { N = year2.ToString() },
                },
            }
        };

        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
        {
            Statements = statements,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

- Weitere API-Informationen finden Sie unter [BatchExecuteStatement](#) in der APIAWS SDK for .NET -Referenz für .

Scannen einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie eine DynamoDB-Tabelle gescannt wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
public static async Task<int> ScanTableAsync(
    AmazonDynamoDBClient client,
    string tableName,
    int startYear,
    int endYear)
{
    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 client.ScanAsync(request);
        foundCount += response.Items.Count;
        response.Items.ForEach(i => DisplayItem(i));
        request.ExclusiveStartKey = response.LastEvaluatedKey;
    }
```

```

    }
    while (response.LastEvaluatedKey.Count > 0);
    return foundCount;
}

```

- Weitere API-Informationen finden Sie unter [Scan](#) in der AWS SDK for .NET -API-Referenz.

Aktualisieren eines Elements in einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie Sie ein Element in einer DynamoDB-Tabelle aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> UpdateItemAsync(
        AmazonDynamoDBClient client,
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)

```

```
{
    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,
    };

    var response = await client.UpdateItemAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [UpdateItem](#) in der APIAWS SDK for .NET -Referenz für .

Schreiben eines Element-Batchs

Im folgenden Codebeispiel wird gezeigt, wie ein DynamoDB-Element-Batch geschrieben wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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 full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null;
    }

    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.
    return allMovies.GetRange(0, 250);
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
```

```
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public static async Task<long> BatchWriteItemsAsync(
    AmazonDynamoDBClient client,
    string movieFileName)
{
    var movies = ImportMovies(movieFileName);
    if (movies is null)
    {
        Console.WriteLine("Couldn't find the JSON file with movie data.");
        return 0;
    }

    var context = new DynamoDBContext(client);

    var movieBatch = context.CreateBatchWrite<Movie>();
    movieBatch.AddPutItems(movies);

    Console.WriteLine("Adding imported movies to the table.");
    await movieBatch.ExecuteAsync();

    return movies.Count;
}
```

- Weitere API-Informationen finden Sie unter [BatchWriteItem](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit Tabellen, Elementen und Abfragen

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.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// This example application performs the following basic Amazon DynamoDB
// functions:
//
//     CreateTableAsync
//     PutItemAsync
//     UpdateItemAsync
//     BatchWriteItemAsync
//     GetItemAsync
//     DeleteItemAsync
//     Query
//     Scan
//     DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();
    }
}
```

```
// Create a new table and wait for it to be active.
Console.WriteLine($"Creating the new table: {tableName}");

var success = await DynamoDbMethods.CreateMovieTableAsync(client,
tableName);

if (success)
{
    Console.WriteLine($"\\nTable: {tableName} successfully created.");
}
else
{
    Console.WriteLine($"\\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 DynamoDbMethods.PutItemAsync(client, 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 DynamoDbMethods.UpdateItemAsync(client, 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 DynamoDbMethods.BatchWriteItemsAsync(client,
movieFileName);
    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 DynamoDbMethods.GetItemAsync(client, lookupMovie,
tableName);
    if (item.Count > 0)
    {
        DynamoDbMethods.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 DynamoDbMethods.DeleteItemAsync(client, 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($"Movies released in {findYear}");
var queryCount = await DynamoDbMethods.QueryMoviesAsync(client, tableName,
findYear);
Console.WriteLine($"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 DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

WaitForEnter();

// Delete the table.
success = await DynamoDbMethods.DeleteTableAsync(client, tableName);
```

```
        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is done.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        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()
{
    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="client">An initialized Amazon DynamoDB client object.</
param>
/// <param name="tableName">The name of the table to create.</param>
/// <returns>A Boolean value indicating the success of the operation.</
returns>
public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
{
    var response = await client.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,
        },
    },
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5,
    },
});

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("Waiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = response.TableDescription.TableName,
};

TableStatus status;

int sleepDuration = 2000;

do
{
    System.Threading.Thread.Sleep(sleepDuration);

    var describeTableResponse = await
client.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
}
```

```
        while (status != "ACTIVE");

        return status == TableStatus.ACTIVE;
    }
}
```

Fügt der Tabelle einen einzelnen Film hinzu.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <param name="newMovie">A Movie object containing information 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
    Movie newMovie, string tableName)
    {
        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,
        };

        var response = await client.PutItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

Aktualisiert ein einzelnes Element in einer Tabelle.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<bool> UpdateItemAsync(
        AmazonDynamoDBClient client,
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        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,
    };

    var response = await client.UpdateItemAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Ruft ein einzelnes Element aus der Filmtabelle ab.

```

    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
    /// <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 static async Task<Dictionary<string, AttributeValue>>
    GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
    {
        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 client.GetItemAsync(request);
        return response.Item;
    }

```


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 full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null;
    }

    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.
    return allMovies.GetRange(0, 250);
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public static async Task<long> BatchWriteItemsAsync(
    AmazonDynamoDBClient client,
    string movieFileName)
```

```

{
    var movies = ImportMovies(movieFileName);
    if (movies is null)
    {
        Console.WriteLine("Couldn't find the JSON file with movie data.");
        return 0;
    }

    var context = new DynamoDBContext(client);

    var movieBatch = context.CreateBatchWrite<Movie>();
    movieBatch.AddPutItems(movies);

    Console.WriteLine("Adding imported movies to the table.");
    await movieBatch.ExecuteAsync();

    return movies.Count;
}

```

Löscht ein einzelnes Element aus der Tabelle.

```

/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
    Movie movieToDelete)
{
    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,
};

var response = await client.DeleteItemAsync(request);
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

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="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
{
    var movieTable = Table.LoadTable(client, tableName);
    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;

    Search 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;
}

```

Scannt die Tabelle nach Filmen, die in einem bestimmten Zeitraum veröffentlicht wurden.

```

public static async Task<int> ScanTableAsync(
    AmazonDynamoDBClient client,
    string tableName,
    int startYear,
    int endYear)
{
    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 client.ScanAsync(request);
    foundCount += response.Items.Count;
    response.Items.ForEach(i => DisplayItem(i));
    request.ExclusiveStartKey = response.LastEvaluatedKey;
}
while (response.LastEvaluatedKey.Count > 0);
return foundCount;
}

```

Löscht die Filmtabelle.

```

public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
{
    var request = new DeleteTableRequest
    {
        TableName = tableName,
    };

    var response = await client.DeleteTableAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
        return true;
    }
}

```

```
        else
        {
            Console.WriteLine("Could not delete table.");
            return false;
        }
    }
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Abfrage](#)
 - [Scan](#)
 - [UpdateItem](#)

Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Stapels von Elementen mithilfe mehrerer SELECT-Anweisungen.
- Hinzufügen eines Stapels von Elementen hinzu, indem mehrere INSERT-Anweisungen ausgeführt werden.
- Aktualisieren eines Stapels von Elementen mithilfe mehrerer UPDATE-Anweisungen.
- Löschen eines Stapels von Elementen mithilfe mehrerer DELETE-Anweisungen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
// GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = "moviedata.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
```

```
        Console.WriteLine("Movies could not be added to the table.");
    }

    WaitForEnter();

    // Update multiple movies by using the BatchExecute statement.
    var title1 = "Star Wars";
    var year1 = 1977;
    var title2 = "Wizard of Oz";
    var year2 = 1939;

    Console.WriteLine($"Updating two movies with producer information: {title1} and
        {title2}.");
    success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
        year2);
    if (success)
    {
        Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
    }
    else
    {
        Console.WriteLine("Select statement failed.");
    }

    WaitForEnter();

    // Update multiple movies by using the BatchExecute statement.
    var producer1 = "LucasFilm";
    var producer2 = "MGM";

    Console.WriteLine($"Updating two movies with producer information: {title1} and
        {title2}.");
    success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1, year1,
        producer2, title2, year2);
    if (success)
    {
        Console.WriteLine($"Successfully updated {title1} and {title2}.");
    }
    else
    {
        Console.WriteLine("Update failed.");
    }

    WaitForEnter();
```



```
// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
    year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
}
```

```

    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting the
table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}

    /// <summary>
    /// Gets movies from the movie table by
    /// using an Amazon DynamoDB PartiQL SELECT statement.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="title1">The title of the first movie.</param>
    /// <param name="title2">The title of the second movie.</param>
    /// <param name="year1">The year of the first movie.</param>
    /// <param name="year2">The year of the second movie.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> GetBatch(
        string tableName,
        string title1,
        string title2,
        int year1,
        int year2)
    {
        var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
        var statements = new List<BatchStatementRequest>
        {
            new BatchStatementRequest
            {

```

```
        Statement = getBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = title1 },
            new AttributeValue { N = year1.ToString() },
        },
    },

    new BatchStatementRequest
    {
        Statement = getBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = title2 },
            new AttributeValue { N = year2.ToString() },
        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

if (response.Responses.Count > 0)
{
    response.Responses.ForEach(r =>
    {
        Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
    });
    return true;
}
else
{
    Console.WriteLine($"Couldn't find either {title1} or {title2}.");
    return false;
}
}

/// <summary>
/// Inserts movies imported from a JSON file into the movie table by
/// using an Amazon DynamoDB PartiQL INSERT statement.
```

```
/// </summary>
/// <param name="tableName">The name of the table into which the movie
/// information will be inserted.</param>
/// <param name="movieFileName">The name of the JSON file that contains
/// movie information.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the insert operation.</returns>
public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
{
    // Get the list of movies from the JSON file.
    var movies = ImportMovies(movieFileName);

    var success = false;

    if (movies is not null)
    {
        // Insert the movies in a batch using PartiQL. Because the
        // batch can contain a maximum of 25 items, insert 25 movies
        // at a time.
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
        var statements = new List<BatchStatementRequest>();

        try
        {
            for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
            {
                for (var i = indexOffset; i < indexOffset + 25; i++)
                {
                    statements.Add(new BatchStatementRequest
                    {
                        Statement = insertBatch,
                        Parameters = new List<AttributeValue>
                        {
                            new AttributeValue { S = movies[i].Title },
                            new AttributeValue { N =
movies[i].Year.ToString() },
                        },
                    });
                }

                var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
```

```
        {
            Statements = statements,
        });

        // Wait between batches for movies to be successfully added.
        System.Threading.Thread.Sleep(3000);

        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

        // Clear the list of statements for the next batch.
        statements.Clear();
    }
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine(ex.Message);
}
}

return success;
}

/// <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 full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    if (allMovies is not null)
    {
        // Return the first 250 entries.
        return allMovies.GetRange(0, 250);
    }
}
```

```
    }
    else
    {
        return null!;
    }
}

/// <summary>
/// Updates information for multiple movies.
/// </summary>
/// <param name="tableName">The name of the table containing the
/// movies to be updated.</param>
/// <param name="producer1">The producer name for the first movie
/// to update.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="year1">The year that the first movie was released.</param>
/// <param name="producer2">The producer name for the second
/// movie to update.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year2">The year that the second movie was released.</param>
/// <returns>A Boolean value that indicates the success of the update.</
returns>
public static async Task<bool> UpdateBatch(
    string tableName,
    string producer1,
    string title1,
    int year1,
    string producer2,
    string title2,
    int year2)
{
    string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = updateBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer1 },
                new AttributeValue { S = title1 },
                new AttributeValue { N = year1.ToString() },
            }
        }
    }
}
```

```
        },
    },

    new BatchStatementRequest
    {
        Statement = updateBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = producer2 },
            new AttributeValue { S = title2 },
            new AttributeValue { N = year2.ToString() },
        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Deletes multiple movies using a PartiQL BatchExecuteAsync
/// statement.
/// </summary>
/// <param name="tableName">The name of the table containing the
/// moves that will be deleted.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="year1">The year the first movie was released.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year2">The year the second movie was released.</param>
/// <returns>A Boolean value indicating the success of the operation.</
returns>
public static async Task<bool> DeleteBatch(
    string tableName,
    string title1,
    int year1,
    string title2,
    int year2)
{
```

```

    string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = updateBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title1 },
                new AttributeValue { N = year1.ToString() },
            },
        },

        new BatchStatementRequest
        {
            Statement = updateBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title2 },
                new AttributeValue { N = year2.ToString() },
            },
        }
    };

    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
    {
        Statements = statements,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Weitere API-Informationen finden Sie unter [BatchExecuteStatement](#) in der APIAWS SDK for .NET -Referenz für .

Abfragen einer Tabelle mit PartiQL

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.

- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
AmazonDynamoDBClient();

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
```

```
// batch can contain a maximum of 25 items, insert 25 movies
// at a time.
string insertBatch = $"INSERT INTO {tableName} VALUE {'title': ?,
'year': ?}";
var statements = new List<BatchStatementRequest>();

try
{
    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
    {
        for (var i = indexOffset; i < indexOffset + 25; i++)
        {
            statements.Add(new BatchStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movies[i].Title },
                    new AttributeValue { N =
movies[i].Year.ToString() },
                },
            });
        }

        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
        {
            Statements = statements,
        });

        // Wait between batches for movies to be successfully added.
        System.Threading.Thread.Sleep(3000);

        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

        // Clear the list of statements for the next batch.
        statements.Clear();
    }
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine(ex.Message);
}
```

```
    }

    return success;
}

/// <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 full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    if (allMovies is not null)
    {
        // Return the first 250 entries.
        return allMovies.GetRange(0, 250);
    }
    else
    {
        return null!;
    }
}

/// <summary>
/// Uses a PartiQL SELECT statement to retrieve a single movie from the
/// movie database.
/// </summary>
/// <param name="tableName">The name of the movie table.</param>
/// <param name="movieTitle">The title of the movie to retrieve.</param>
/// <returns>A list of movie data. If no movie matches the supplied
/// title, the list is empty.</returns>
```

```
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
        };

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = selectSingle,
            Parameters = parameters,
        });

        return response.Items;
    }

    /// <summary>
    /// Retrieve multiple movies by year using a SELECT statement.
    /// </summary>
    /// <param name="tableName">The name of the movie table.</param>
    /// <param name="year">The year the movies were released.</param>
    /// <returns></returns>
    public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { N = year.ToString() },
        };

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = selectSingle,
            Parameters = parameters,
        });

        return response.Items;
    }
}
```

```
    }

    /// <summary>
    /// Inserts a single movie into the movies table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to insert.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the INSERT operation.</returns>
    public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
    {
        string insertBatch = $"INSERT INTO {tableName} VALUE {'title': ?,
'year': ?}";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Updates a single movie in the table, adding information for the
    /// producer.
    /// </summary>
    /// <param name="tableName">the name of the table.</param>
    /// <param name="producer">The name of the producer.</param>
    /// <param name="movieTitle">The movie title.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// UPDATE operation.</returns>
```

```
public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
{
    string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = insertSingle,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = producer },
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

```
/// <summary>
/// Deletes a single movie from the table.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="movieTitle">The title of the movie to delete.</param>
/// <param name="year">The year that the movie was released.</param>
/// <returns>A Boolean value that indicates the success of the
/// DELETE operation.</returns>
public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
{
    var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = deleteSingle,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },

```

```
        new AttributeValue { N = year.ToString() },
    },
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Displays the list of movies returned from a database query.
/// </summary>
/// <param name="items">The list of movie information to display.</param>
private static void DisplayMovies(List<Dictionary<string, AttributeValue>>
items)
{
    if (items.Count > 0)
    {
        Console.WriteLine($"Found {items.Count} movies.");
        items.ForEach(item =>
Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
    }
    else
    {
        Console.WriteLine($"Didn't find a movie that matched the supplied
criteria.");
    }
}

}

}

/// <summary>
/// Uses a PartiQL SELECT statement to retrieve a single movie from the
/// movie database.
/// </summary>
/// <param name="tableName">The name of the movie table.</param>
/// <param name="movieTitle">The title of the movie to retrieve.</param>
/// <returns>A list of movie data. If no movie matches the supplied
/// title, the list is empty.</returns>
public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
```

```
{
    string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
    var parameters = new List<AttributeValue>
    {
        new AttributeValue { S = movieTitle },
    };

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = selectSingle,
        Parameters = parameters,
    });

    return response.Items;
}

/// <summary>
/// Inserts a single movie into the movies table.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="movieTitle">The title of the movie to insert.</param>
/// <param name="year">The year that the movie was released.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the INSERT operation.</returns>
public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
{
    string insertBatch = $"INSERT INTO {tableName} VALUE {'title': ?,
'year': ?}";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = insertBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });
}
```



```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Updates a single movie in the table, adding information for the
    /// producer.
    /// </summary>
    /// <param name="tableName">the name of the table.</param>
    /// <param name="producer">The name of the producer.</param>
    /// <param name="movieTitle">The movie title.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// UPDATE operation.</returns>
    public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
    {
        string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertSingle,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer },
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Deletes a single movie from the table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to delete.</param>
    /// <param name="year">The year that the movie was released.</param>
```

```
/// <returns>A Boolean value that indicates the success of the
/// DELETE operation.</returns>
public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
{
    var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = deleteSingle,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [ExecuteStatement](#) in der APIAWS SDK for .NET - Referenz für .

Verwenden eines Dokumentmodells

Das folgende Codebeispiel zeigt, wie CRUD-Operationen (Create, Read, Update, Delete) und Batch-Operationen unter Verwendung eines Dokumentmodells für DynamoDB und eines AWS SDK ausgeführt werden.

Weitere Informationen finden Sie unter [Dokumentmodell](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie CRUD-Operationen unter Verwendung eines Dokumentmodells durch.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
    /// <param name="tableName">The name of the table to load.</param>
    /// <returns>A DynamoDB table object.</returns>
    public static Table LoadTable(IAmazonDynamoDB client, string tableName)
    {
        Table productCatalog = Table.LoadTable(client, tableName);
        return productCatalog;
    }

    /// <summary>
    /// Creates an example book item and adds it to the DynamoDB table
    /// ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
```

```
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
    {
        Console.WriteLine("\n*** Executing CreateBookItem() ***");
        var book = new Document
        {
            ["Id"] = sampleBookId,
            ["Title"] = "Book " + sampleBookId,
            ["Price"] = 19.99,
            ["ISBN"] = "111-1111111111",
            ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
            ["PageCount"] = 500,
            ["Dimensions"] = "8.5x11x.5",
            ["InPublication"] = new DynamoDBBool(true),
            ["InStock"] = new DynamoDBBool(false),
            ["QuantityOnHand"] = 0,
        };

        // Adds the book to the ProductCatalog table.
        await productCatalog.PutItemAsync(book);
    }

    /// <summary>
    /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void RetrieveBook(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing RetrieveBook() ***");

        // Optional configuration.
        var config = new GetItemOperationConfig
        {
            AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
            ConsistentRead = true,
        };
    }
}
```

```
        Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
        Console.WriteLine("RetrieveBook: Printing book retrieved...");
        PrintDocument(document);
    }

    /// <summary>
    /// Updates multiple attributes for a book and writes the changes to the
    /// DynamoDB table ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void UpdateMultipleAttributes(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\nUpdating multiple attributes....");
        int partitionKey = sampleBookId;

        var book = new Document
        {
            ["Id"] = partitionKey,

            // List of attribute updates.
            // The following replaces the existing authors list.
            ["Authors"] = new List<string> { "Author x", "Author y" },
            ["newAttribute"] = "New Value",
            ["ISBN"] = null, // Remove it.
        };

        // Optional parameters.
        var config = new UpdateItemOperationConfig
        {
            // Gets updated item in response.
            ReturnValues = ReturnValues.AllNewAttributes,
        };

        Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
        Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
        PrintDocument(updatedBook);
    }
}
```

```
    }

    /// <summary>
    /// Updates a book item if it meets the specified criteria.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void UpdateBookPriceConditionally(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

        int partitionKey = sampleBookId;

        var book = new Document
        {
            ["Id"] = partitionKey,
            ["Price"] = 29.99,
        };

        // For conditional price update, creating a condition expression.
        var expr = new Expression
        {
            ExpressionStatement = "Price = :val",
        };
        expr.ExpressionAttributeValueValues[":val"] = 19.00;

        // Optional parameters.
        var config = new UpdateItemOperationConfig
        {
            ConditionalExpression = expr,
            ReturnValues = ReturnValues.AllNewAttributes,
        };

        Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
        Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
        PrintDocument(updatedBook);
    }

    /// <summary>
```

```
    /// Deletes the book with the supplied Id value from the DynamoDB table
    /// ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async Task DeleteBook(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing DeleteBook() ***");

        // Optional configuration.
        var config = new DeleteItemOperationConfig
        {
            // Returns the deleted item.
            ReturnValues = ReturnValues.AllOldAttributes,
        };
        Document document = await productCatalog.DeleteItemAsync(sampleBookId,
config);
        Console.WriteLine("DeleteBook: Printing deleted just deleted...");

        PrintDocument(document);
    }

    /// <summary>
    /// Prints the information for the supplied DynamoDB document.
    /// </summary>
    /// <param name="updatedDocument">A DynamoDB document object.</param>
    public static void PrintDocument(Document updatedDocument)
    {
        if (updatedDocument is null)
        {
            return;
        }

        foreach (var attribute in updatedDocument.GetAttributeNames())
        {
            string stringValue = null;
            var value = updatedDocument[attribute];

            if (value is null)
            {
                continue;
            }
        }
    }
}
```

```

    }

    if (value is Primitive)
    {
        stringValue = value.AsPrimitive().Value.ToString();
    }
    else if (value is PrimitiveList)
    {
        stringValue = string.Join(",", (from primitive
                                        in value.AsPrimitiveList().Entries
                                        select
primitive.Value).ToArray());
    }

    Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
    }
}
}

```

Führen Sie Batch-Schreiboperationen unter Verwendung eines Dokumentmodells durch.

```

/// <summary>
/// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
/// operations.
/// </summary>
public class MidLevelBatchWriteItem
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();

        await SingleTableBatchWrite(client);
        await MultiTableBatchWrite(client);
    }

    /// <summary>
    /// Perform a batch operation on a single DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB object.</param>

```



```
public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
{
    Table productCatalog = Table.LoadTable(client, "ProductCatalog");
    var batchWrite = productCatalog.CreateBatchWrite();

    var book1 = new Document
    {
        ["Id"] = 902,
        ["Title"] = "My book1 in batch write using .NET helper classes",
        ["ISBN"] = "902-11-11-1111",
        ["Price"] = 10,
        ["ProductCategory"] = "Book",
        ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
        ["Dimensions"] = "8.5x11x.5",
        ["InStock"] = new DynamoDBBool(true),
        ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown at
this time.
    };

    batchWrite.AddDocumentToPut(book1);

    // Specify delete item using overload that takes PK.
    batchWrite.AddKeyToDelete(12345);
    Console.WriteLine("Performing batch write in SingleTableBatchWrite()");
    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Perform a batch operation involving multiple DynamoDB tables.
/// </summary>
/// <param name="client">An initialized DynamoDB client object.</param>
public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
{
    // Specify item to add in the Forum table.
    Table forum = Table.LoadTable(client, "Forum");
    var forumBatchWrite = forum.CreateBatchWrite();

    var forum1 = new Document
    {
        ["Name"] = "Test BatchWrite Forum",
        ["Threads"] = 0,
    };
    forumBatchWrite.AddDocumentToPut(forum1);
}
```

```
// Specify item to add in the Thread table.
Table thread = Table.LoadTable(client, "Thread");
var threadBatchWrite = thread.CreateBatchWrite();

var thread1 = new Document
{
    ["ForumName"] = "S3 forum",
    ["Subject"] = "My sample question",
    ["Message"] = "Message text",
    ["KeywordTags"] = new List<string> { "S3", "Bucket" },
};
threadBatchWrite.AddDocumentToPut(thread1);

// Specify item to delete from the Thread table.
threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

// Create multi-table batch.
var superBatch = new MultiTableDocumentBatchWrite();
superBatch.AddBatch(forumBatchWrite);
superBatch.AddBatch(threadBatchWrite);
Console.WriteLine("Performing batch write in MultiTableBatchWrite()");

// Execute the batch.
await superBatch.ExecuteAsync();
}
}
```

Scannen Sie eine Tabelle unter Verwendung eines Dokumentmodells.

```
/// <summary>
/// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
/// table for values.
/// </summary>
public class MidLevelScanOnly
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();
```

```
        Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");

        await FindProductsWithNegativePrice(productCatalogTable);
        await FindProductsWithNegativePriceWithConfig(productCatalogTable);
    }

    /// <summary>
    /// Retrieves any products that have a negative price in a DynamoDB table.
    /// </summary>
    /// <param name="productCatalogTable">A DynamoDB table object.</param>
    public static async Task FindProductsWithNegativePrice(
        Table productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        var scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

        Search search = productCatalogTable.Scan(scanFilter);

        do
        {
            var documentList = await search.GetNextSetAsync();
            Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");

            foreach (var document in documentList)
            {
                PrintDocument(document);
            }
        }
        while (!search.IsDone);
    }

    /// <summary>
    /// Finds any items in the ProductCatalog table using a DynamoDB
    /// configuration object.
    /// </summary>
    /// <param name="productCatalogTable">A DynamoDB table object.</param>
    public static async Task FindProductsWithNegativePriceWithConfig(
        Table productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        var scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);
```

```
var config = new ScanOperationConfig()
{
    Filter = scanFilter,
    Select = SelectValues.SpecificAttributes,
    AttributesToGet = new List<string> { "Title", "Id" },
};

Search search = productCatalogTable.Scan(config);

do
{
    var documentList = await search.GetNextSetAsync();
    Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

    foreach (var document in documentList)
    {
        PrintDocument(document);
    }
} while (!search.IsDone);
}

/// <summary>
/// Displays the details of the passed DynamoDB document object on the
/// console.
/// </summary>
/// <param name="document">A DynamoDB document object.</param>
public static void PrintDocument(Document document)
{
    Console.WriteLine();
    foreach (var attribute in document.GetAttributeNames())
    {
        string stringValue = null;
        var value = document[attribute];
        if (value is Primitive)
        {
            stringValue = value.AsPrimitive().Value.ToString();
        }
        else if (value is PrimitiveList)
        {
            stringValue = string.Join(",", (from primitive
            in value.AsPrimitiveList().Entries
```

```
select
primitive.Value).ToArray());
    }

    Console.WriteLine($"{attribute} - {stringValue}");
}
}
}
```

Verwenden Sie ein Dokumentmodell, um eine Tabelle abzufragen und zu scannen.

```
/// <summary>
/// Shows how to perform mid-level query procedures on an Amazon DynamoDB
/// table.
/// </summary>
public class MidLevelQueryAndScan
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();

        // Query examples.
        Table replyTable = Table.LoadTable(client, "Reply");
        string forumName = "Amazon DynamoDB";
        string threadSubject = "DynamoDB Thread 2";

        await FindRepliesInLast15Days(replyTable);
        await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
        await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

        // Get Example.
        Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
        int productId = 101;

        await GetProduct(productCatalogTable, productId);
    }

    /// <summary>
```

```
/// Retrieves information about a product from the DynamoDB table
/// ProductCatalog based on the product ID and displays the information
/// on the console.
/// </summary>
/// <param name="tableName">The name of the table from which to retrieve
/// product information.</param>
/// <param name="productId">The ID of the product to retrieve.</param>
public static async Task GetProduct(Table tableName, int productId)
{
    Console.WriteLine("*** Executing GetProduct() ***");
    Document productDocument = await tableName.GetItemAsync(productId);
    if (productDocument != null)
    {
        PrintDocument(productDocument);
    }
    else
    {
        Console.WriteLine("Error: product " + productId + " does not
exist");
    }
}

/// <summary>
/// Retrieves replies from the passed DynamoDB table object.
/// </summary>
/// <param name="table">The table we want to query.</param>
public static async Task FindRepliesInLast15Days(
    Table table)
{
    DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
    var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
    filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

    // Use Query overloads that take the minimum required query parameters.
    Search search = table.Query(filter);

    do
    {
        var documentSet = await search.GetNextSetAsync();
        Console.WriteLine("\nFindRepliesInLast15Days:
printing .....");

        foreach (var document in documentSet)
```

```
        {
            PrintDocument(document);
        }
    }
    while (!search.IsDone);
}

/// <summary>
/// Retrieve replies made during a specific time period.
/// </summary>
/// <param name="table">The table we want to query.</param>
/// <param name="forumName">The name of the forum that we're interested
in.</param>
/// <param name="threadSubject">The subject of the thread, which we are
/// searching for replies.</param>
public static async Task FindRepliesPostedWithinTimePeriod(
    Table table,
    string forumName,
    string threadSubject)
{
    DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
    DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0, 0));

    var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadSubject);
    filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

    var config = new QueryOperationConfig()
    {
        Limit = 2, // 2 items/page.
        Select = SelectValues.SpecificAttributes,
        AttributesToGet = new List<string>
    {
        "Message",
        "ReplyDateTime",
        "PostedBy",
    },
        ConsistentRead = true,
        Filter = filter,
    };

    Search search = table.Query(config);
```

```
        do
        {
            var documentList = await search.GetNextSetAsync();
            Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

            foreach (var document in documentList)
            {
                PrintDocument(document);
            }
        }
        while (!search.IsDone);
    }

    /// <summary>
    /// Perform a query for replies made in the last 15 days using a DynamoDB
    /// QueryOperationConfig object.
    /// </summary>
    /// <param name="table">The table we want to query.</param>
    /// <param name="forumName">The name of the forum that we're interested
in.</param>
    /// <param name="threadName">The bane of the thread that we are searching
    /// for replies.</param>
    public static async Task FindRepliesInLast15DaysWithConfig(
        Table table,
        string forumName,
        string threadName)
    {
        DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
        var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadName);
        filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

        var config = new QueryOperationConfig()
        {
            Filter = filter,

            // Optional parameters.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "Message",
```



```
        "ReplyDateTime",
        "PostedBy",
    },
    ConsistentRead = true,
};

Search search = table.Query(config);

do
{
    var documentSet = await search.GetNextSetAsync();
    Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

    foreach (var document in documentSet)
    {
        PrintDocument(document);
    }
}
while (!search.IsDone);
}

/// <summary>
/// Displays the contents of the passed DynamoDB document on the console.
/// </summary>
/// <param name="document">A DynamoDB document to display.</param>
public static void PrintDocument(Document document)
{
    Console.WriteLine();
    foreach (var attribute in document.GetAttributeNames())
    {
        string stringValue = null;
        var value = document[attribute];

        if (value is Primitive)
        {
            stringValue = value.AsPrimitive().Value.ToString();
        }
        else if (value is PrimitiveList)
        {
            stringValue = string.Join(",", (from primitive
                in value.AsPrimitiveList().Entries
                select
primitive.Value).ToArray());
        }
    }
}
```

```
        }  
        Console.WriteLine($"{attribute} - {stringValue}");  
    }  
}
```

Verwenden eines übergeordneten Object-Persistence-Modells

Das folgende Codebeispiel zeigt, wie CRUD-Operationen (Create, Read, Update, Delete) und Batch-Operationen unter Verwendung eines Object-Persistence-Modells für DynamoDB und eines - AWS SDK ausgeführt werden.

Weitere Informationen finden Sie unter [Object-Persistence-Modell](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie CRUD-Operationen unter Verwendung eines übergeordneten Object-Persistence-Modells durch.

```
/// <summary>  
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB  
/// table.  
/// </summary>  
public class HighLevelItemCrud  
{  
    public static async Task Main()  
    {  
        var client = new AmazonDynamoDBClient();  
        DynamoDBContext context = new DynamoDBContext(client);  
        await PerformCRUDOperations(context);  
    }  
}
```

```
public static async Task PerformCRUDOperations(IDynamoDBContext context)
{
    int bookId = 1001; // Some unique value.
    Book myBook = new Book
    {
        Id = bookId,
        Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
        Isbn = "111-1111111001",
        BookAuthors = new List<string> { "Author 1", "Author 2" },
    };

    // Save the book to the ProductCatalog table.
    await context.SaveAsync(myBook);

    // Retrieve the book from the ProductCatalog table.
    Book bookRetrieved = await context.LoadAsync<Book>(bookId);

    // Update some properties.
    bookRetrieved.Isbn = "222-2222221001";

    // Update existing authors list with the following values.
    bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };

    await context.SaveAsync(bookRetrieved);

    // Retrieve the updated book. This time, add the optional
    // ConsistentRead parameter using DynamoDBContextConfig object.
    await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
    {
        ConsistentRead = true,
    });

    // Delete the book.
    await context.DeleteAsync<Book>(bookId);

    // Try to retrieve deleted book. It should return null.
    Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
    {
        ConsistentRead = true,
    });

    if (deletedBook == null)
```

```
        {  
            Console.WriteLine("Book is deleted");  
        }  
    }  
}
```

Führen Sie Batch-Schreiboperationen unter Verwendung eines übergeordneten Object-Persistence-Modells durch.

```
/// <summary>  
/// Performs high-level batch write operations to an Amazon DynamoDB table.  
/// This example was written using the AWS SDK for .NET version 3.7 and .NET  
/// Core 5.0.  
/// </summary>  
public class HighLevelBatchWriteItem  
{  
    public static async Task SingleTableBatchWrite(IDynamoDBContext context)  
    {  
        Book book1 = new Book  
        {  
            Id = 902,  
            InPublication = true,  
            Isbn = "902-11-11-1111",  
            PageCount = "100",  
            Price = 10,  
            ProductCategory = "Book",  
            Title = "My book3 in batch write",  
        };  
  
        Book book2 = new Book  
        {  
            Id = 903,  
            InPublication = true,  
            Isbn = "903-11-11-1111",  
            PageCount = "200",  
            Price = 10,  
            ProductCategory = "Book",  
            Title = "My book4 in batch write",  
        };  
    }  
}
```

```
var bookBatch = context.CreateBatchWrite<Book>();
bookBatch.AddPutItems(new List<Book> { book1, book2 });

Console.WriteLine("Adding two books to ProductCatalog table.");
await bookBatch.ExecuteAsync();
}

public static async Task MultiTableBatchWrite(IDynamoDBContext context)
{
    // New Forum item.
    Forum newForum = new Forum
    {
        Name = "Test BatchWrite Forum",
        Threads = 0,
    };
    var forumBatch = context.CreateBatchWrite<Forum>();
    forumBatch.AddPutItem(newForum);

    // New Thread item.
    Thread newThread = new Thread
    {
        ForumName = "S3 forum",
        Subject = "My sample question",
        KeywordTags = new List<string> { "S3", "Bucket" },
        Message = "Message text",
    };

    DynamoDBOperationConfig config = new DynamoDBOperationConfig();
    config.SkipVersionCheck = true;
    var threadBatch = context.CreateBatchWrite<Thread>(config);
    threadBatch.AddPutItem(newThread);
    threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

    var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

    Console.WriteLine("Performing batch write in MultiTableBatchWrite().");
    await superBatch.ExecuteAsync();
}

public static async Task Main()
{
    AmazonDynamoDBClient client = new AmazonDynamoDBClient();
    DynamoDBContext context = new DynamoDBContext(client);
```

```

        await SingleTableBatchWrite(context);
        await MultiTableBatchWrite(context);
    }
}

```

Weisen Sie einer Tabelle unter Verwendung eines übergeordneten Object-Persistence-Modells beliebige Daten zu.

```

/// <summary>
/// Shows how to map arbitrary data to an Amazon DynamoDB table.
/// </summary>
public class HighLevelMappingArbitraryData
{
    /// <summary>
    /// Creates a book, adds it to the DynamoDB ProductCatalog table, retrieves
    /// the new book from the table, updates the dimensions and writes the
    /// changed item back to the table.
    /// </summary>
    /// <param name="context">The DynamoDB context object used to write and
    /// read data from the table.</param>
    public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
    {
        // Create a book.
        DimensionType myBookDimensions = new DimensionType()
        {
            Length = 8M,
            Height = 11M,
            Thickness = 0.5M,
        };

        Book myBook = new Book
        {
            Id = 501,
            Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
            Isbn = "999-9999999999",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
            Dimensions = myBookDimensions,
        };
    }
}

```

```
// Add the book to the DynamoDB table ProductCatalog.
await context.SaveAsync(myBook);

// Retrieve the book.
Book bookRetrieved = await context.LoadAsync<Book>(501);

// Update the book dimensions property.
bookRetrieved.Dimensions.Height += 1;
bookRetrieved.Dimensions.Length += 1;
bookRetrieved.Dimensions.Thickness += 0.2M;

// Write the changed item to the table.
await context.SaveAsync(bookRetrieved);
}

public static async Task Main()
{
    var client = new AmazonDynamoDBClient();
    DynamoDBContext context = new DynamoDBContext(client);
    await AddRetrieveUpdateBook(context);
}
}
```

Verwenden Sie ein übergeordnetes Object-Persistence-Modell, um eine Tabelle abzufragen und zu scannen.

```
/// <summary>
/// Shows how to perform high-level query and scan operations to Amazon
/// DynamoDB tables.
/// </summary>
public class HighLevelQueryAndScan
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        DynamoDBContext context = new DynamoDBContext(client);

        // Get an item.
```

```

        await GetBook(context, 101);

        // Sample forum and thread to test queries.
        string forumName = "Amazon DynamoDB";
        string threadSubject = "DynamoDB Thread 1";

        // Sample queries.
        await FindRepliesInLast15Days(context, forumName, threadSubject);
        await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

        // Scan table.
        await FindProductsPricedLessThanZero(context);
    }

    public static async Task GetBook(IDynamoDBContext context, int productId)
    {
        Book bookItem = await context.LoadAsync<Book>(productId);

        Console.WriteLine("\nGetBook: Printing result.....");
        Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn} \n
No. of pages: {bookItem.PageCount}");
    }

    /// <summary>
    /// Queries a DynamoDB table to find replies posted within the last 15 days.
    /// </summary>
    /// <param name="context">The DynamoDB context used to perform the query.</
param>
    /// <param name="forumName">The name of the forum that we're interested
in.</param>
    /// <param name="threadSubject">The thread object containing the query
parameters.</param>
    public static async Task FindRepliesInLast15Days(
        IDynamoDBContext context,
        string forumName,
        string threadSubject)
    {
        string replyId = $"{forumName} #{threadSubject}";
        DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

        List<object> times = new List<object>();
        times.Add(twoWeeksAgoDate);
    }

```



```
List<ScanCondition> scs = new List<ScanCondition>();
var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
scs.Add(sc);

var cfg = new DynamoDBOperationConfig
{
    QueryFilter = scs,
};

AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId, cfg);
IEnumerable<Reply> latestReplies = await response.GetRemainingAsync();

Console.WriteLine("\nReplies in last 15 days:");

foreach (Reply r in latestReplies)
{
    Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
}

/// <summary>
/// Queries for replies posted within a specific time period.
/// </summary>
/// <param name="context">The DynamoDB context used to perform the query.</
param>
/// <param name="forumName">The name of the forum that we're interested
in.</param>
/// <param name="threadSubject">Information about the subject that we're
/// interested in.</param>
public static async Task FindRepliesPostedWithinTimePeriod(
    IDynamoDBContext context,
    string forumName,
    string threadSubject)
{
    string forumId = forumName + "#" + threadSubject;
    Console.WriteLine("\nReplies posted within time period:");

    DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
    DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

    List<object> times = new List<object>();
    times.Add(startDate);
```

```
        times.Add(endDate);

        List<ScanCondition> scs = new List<ScanCondition>();
        var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
        scs.Add(sc);

        var cfg = new DynamoDBOperationConfig
        {
            QueryFilter = scs,
        };

        AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId, cfg);
        IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

        foreach (Reply r in repliesInAPeriod)
        {
            Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
        }
    }

    /// <summary>
    /// Queries the DynamoDB ProductCatalog table for products costing less
    /// than zero.
    /// </summary>
    /// <param name="context">The DynamoDB context object used to perform the
    /// query.</param>
    public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
    {
        int price = 0;

        List<ScanCondition> scs = new List<ScanCondition>();
        var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
        var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");

        scs.Add(sc1);
        scs.Add(sc2);

        AsyncSearch<Book> response = context.ScanAsync<Book>(scs);
```

```
        IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

        Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

        foreach (Book r in itemsWithWrongPrice)
        {
            Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
        }
    }
}
```

Amazon EC2-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon EC2 verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Amazon EC2

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon EC2.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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>A Task object.</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>();

        var request = new DescribeSecurityGroupsRequest
        {
            MaxResults = 10,
        };

        // Retrieve information about up to 10 Amazon EC2 security groups.
        var response = await ec2Client.DescribeSecurityGroupsAsync(request);
    }
}
```

```
// Now print the security groups returned by the call to
// DescribeSecurityGroupsAsync.
Console.WriteLine("Security Groups:");
response.SecurityGroups.ForEach(group =>
{
    Console.WriteLine($"Security group: {group.GroupName} ID:
{group.GroupId}");
});
}
```

- Weitere API-Informationen finden Sie unter [DescribeSecurityGroups](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Zuweisen einer Elastic-IP-Adresse

Das folgende Codebeispiel zeigt, wie Sie eine Elastic IP-Adresse für Amazon EC2 zuweisen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Allocate an Elastic IP address.
/// </summary>
/// <returns>The allocation Id of the allocated address.</returns>
public async Task<string> AllocateAddress()
{
    var request = new AllocateAddressRequest();
```

```
    var response = await _amazonEC2.AllocateAddressAsync(request);
    return response.AllocationId;
}
```

- Weitere API-Informationen finden Sie unter [AllocateAddress](#) in der APIAWS SDK for .NET - Referenz für .

Zuordnen einer Elastic-IP-Adresse zu einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Elastic IP-Adresse einer Amazon EC2-Instance zuordnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Associate an Elastic IP address to an EC2 instance.
/// </summary>
/// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
/// <param name="instanceId">The instance Id of the EC2 instance to
/// associate the address with.</param>
/// <returns>The association Id that represents
/// the association of the Elastic IP address with an instance.</returns>
public async Task<string> AssociateAddress(string allocationId, string
instanceId)
{
    var request = new AssociateAddressRequest
    {
        AllocationId = allocationId,
        InstanceId = instanceId
    };

    var response = await _amazonEC2.AssociateAddressAsync(request);
```

```
        return response.AssociationId;
    }
```

- Weitere API-Informationen finden Sie unter [AssociateAddress](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Startvorlage

Das folgende Codebeispiel zeigt, wie eine Amazon-EC2-Startvorlage erstellt wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
/// The launch template specifies a Bash script in its user data field that runs
after
/// the instance is started. This script installs the Python packages and starts
a Python
/// web server on the instance.
/// </summary>
/// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
/// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
/// <returns>The template object.</returns>
public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
{
    await CreateKeyPair(_keyPairName);
    await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

    var startServerText = await File.ReadAllTextAsync(startupScriptPath);
    var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);
```

```
var amiLatest = await _amazonSsm.GetParameterAsync(
    new GetParameterRequest() { Name = _amiParam });
var amiId = amiLatest.Parameter.Value;
var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
    new CreateLaunchTemplateRequest()
    {
        LaunchTemplateName = _launchTemplateName,
        LaunchTemplateData = new RequestLaunchTemplateData()
        {
            InstanceType = _instanceType,
            ImageId = amiId,
            IamInstanceProfile =
                new
                    LaunchTemplateIamInstanceProfileSpecificationRequest()
                {
                    Name = _instanceProfileName
                },
            KeyName = _keyPairName,
            UserData = System.Convert.ToBase64String(plainTextBytes)
        }
    });
return launchTemplateResponse.LaunchTemplate;
}
```

- Weitere API-Informationen finden Sie unter [CreateLaunchTemplate](#) in der APIAWS SDK for .NET -Referenz für .

Eine Sicherheitsgruppe erstellen

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2Sicherheitsgruppe erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
///  
/// <summary>
```



```

    /// Create an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>
    public async Task<string> CreateSecurityGroup(string groupName, string
groupDescription)
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        return response.GroupId;
    }

```

- Weitere API-Informationen finden Sie unter [CreateSecurityGroup](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Sicherheitsschlüsselpaars

Das folgende Codebeispiel zeigt, wie Sie ein Sicherheitsschlüsselpaar für Amazon EC2 erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Create an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name for the new key pair.</param>
    /// <returns>The Amazon EC2 key pair created.</returns>
    public async Task<KeyPair?> CreateKeyPair(string keyPairName)
    {
        var request = new CreateKeyPairRequest
        {
            KeyName = keyPairName,

```

```
};

var response = await _amazonEC2.CreateKeyPairAsync(request);

if (response.HttpStatusCode == HttpStatusCode.OK)
{
    var kp = response.KeyPair;
    return kp;
}
else
{
    Console.WriteLine("Could not create key pair.");
    return null;
}
}

/// <summary>
/// Save KeyPair information to a temporary file.
/// </summary>
/// <param name="keyPair">The name of the key pair.</param>
/// <returns>The full path to the temporary file.</returns>
public string SaveKeyPair(KeyPair keyPair)
{
    var tempPath = Path.GetTempPath();
    var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
    var pemFileName = Path.ChangeExtension(tempFileName, "pem");

    // Save the key pair to a file in a temporary folder.
    using var stream = new FileStream(pemFileName, FileMode.Create);
    using var writer = new StreamWriter(stream);
    writer.WriteLine(keyPair.KeyMaterial);

    return pemFileName;
}
```

- Weitere API-Informationen finden Sie unter [CreateKeyPair](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen und Ausführen einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2 erstellen und ausführen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create and run an EC2 instance.
/// </summary>
/// <param name="ImageId">The image Id of the image used as a basis for the
/// EC2 instance.</param>
/// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
/// <param name="keyName">The name of the key pair to associate with the
/// instance.</param>
/// <param name="groupId">The Id of the Amazon EC2 security group that will be
/// allowed to interact with the new EC2 instance.</param>
/// <returns>The instance Id of the new EC2 instance.</returns>
public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
{
    var request = new RunInstancesRequest
    {
        ImageId = imageId,
        InstanceType = instanceType,
        KeyName = keyName,
        MinCount = 1,
        MaxCount = 1,
        SecurityGroupIds = new List<string> { groupId }
    };
    var response = await _amazonEC2.RunInstancesAsync(request);
    return response.Reservation.Instances[0].InstanceId;
}
```

- Weitere API-Informationen finden Sie unter [RunInstances](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Startvorlage

Das folgende Code-Beispiel zeigt, wie Sie eine Amazon-SEC-Startvorlage löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteLaunchTemplate](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Sicherheitsgruppe

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2Sicherheitsgruppe löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an Amazon EC2 security group.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteSecurityGroup(string groupId)
{
    var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteSecurityGroup](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Sicherheitsschlüsselpaars

Das folgende Codebeispiel zeigt, wie Sie ein Amazon EC2-Sicherheitsschlüsselpaar löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an Amazon EC2 key pair.
/// </summary>
```

```
/// <param name="keyPairName">The name of the key pair to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyPair(string keyPairName)
{
    try
    {
        await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Delete the temporary file where the key pair information was saved.
/// </summary>
/// <param name="tempFileName">The path to the temporary file.</param>
public void DeleteTempFile(string tempFileName)
{
    if (File.Exists(tempFileName))
    {
        File.Delete(tempFileName);
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteKeyPair](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben von Availability Zones

Das folgende Codebeispiel zeigt, wie Sie Amazon EC2 Availability Zones beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
    return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
}
```

- Weitere API-Informationen finden Sie unter [DescribeAvailabilityZones](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von Instances

Das folgende Codebeispiel zeigt, wie Amazon-EC2 Amazon EC2 beschrieben werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about existing EC2 images.
/// </summary>
/// <returns>Async task.</returns>
public async Task DescribeInstances()
```

```
{
    // List all EC2 instances.
    await GetInstanceDescriptions();

    string tagName = "IncludeInList";
    string tagValue = "Yes";
    await GetInstanceDescriptionsFiltered(tagName, tagValue);
}

/// <summary>
/// Get information for all existing Amazon EC2 instances.
/// </summary>
/// <returns>Async task.</returns>
public async Task GetInstanceDescriptions()
{
    Console.WriteLine("Showing all instances:");
    var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

    await foreach (var response in paginator.Responses)
    {
        foreach (var reservation in response.Reservations)
        {
            foreach (var instance in reservation.Instances)
            {
                Console.Write($"Instance ID: {instance.InstanceId}");
                Console.WriteLine($"\\tCurrent State: {instance.State.Name}");
            }
        }
    }
}

/// <summary>
/// Get information about EC2 instances filtered by a tag name and value.
/// </summary>
/// <param name="tagName">The name of the tag to filter on.</param>
/// <param name="tagValue">The value of the tag to look for.</param>
/// <returns>Async task.</returns>
public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
{
    // This tag filters the results of the instance list.
    var filters = new List<Filter>
    {
```



```

        new Filter
        {
            Name = $"tag:{tagName}",
            Values = new List<string>
            {
                tagValue,
            },
        },
    };
    var request = new DescribeInstancesRequest
    {
        Filters = filters,
    };

    Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to \"Yes\".");
    var paginator = _amazonEC2.Paginators.DescribeInstances(request);

    await foreach (var response in paginator.Responses)
    {
        foreach (var reservation in response.Reservations)
        {
            foreach (var instance in reservation.Instances)
            {
                Console.Write($"Instance ID: {instance.InstanceId} ");
                Console.WriteLine($"\\tCurrent State: {instance.State.Name}");
            }
        }
    }
}

```

- Weitere API-Informationen finden Sie unter [DescribeInstances](#) in der APIAWS SDK for .NET - Referenz für .

Aufhebung der Zuordnung einer Elastic IP-Adresse zu einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Elastic IP-Adresse von einer Amazon EC2-Instance trennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    var response = await _amazonEC2.DisassociateAddressAsync(
        new DisassociateAddressRequest { AssociationId = associationId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DisassociateAddress](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Daten zu einer Sicherheitsgruppe

Das folgende Codebeispiel zeigt, wie Sie Daten zu einer Amazon EC2Sicherheitsgruppe abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve information for an Amazon EC2 security group.
/// </summary>
/// <param name="groupId">The Id of the Amazon EC2 security group.</param>
```

```
/// <returns>A list of security group information.</returns>
public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
{
    var request = new DescribeSecurityGroupsRequest();
    var groupIds = new List<string> { groupId };
    request.GroupIds = groupIds;

    var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
    return response.SecurityGroups;
}

/// <summary>
/// Display the information returned by the call to
/// DescribeSecurityGroupsAsync.
/// </summary>
/// <param name="securityGroup">A list of security group information.</param>
public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
{
    Console.WriteLine($"{securityGroup.GroupName}");
    Console.WriteLine("Ingress permissions:");
    securityGroup.IpPermissions.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.Write($"{range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

        Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");
    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");
    });
}
```

```

        Console.WriteLine($"{\tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"{range.CidrIp}
"); });

        Console.WriteLine($"{\n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.WriteLine($"{range.CidrIpv6} "); });

        Console.WriteLine($"{\n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{id.Id} "));

        Console.WriteLine($"{\n\tTo Port: {permission.ToPort}");
    });
}

```

- Weitere API-Informationen finden Sie unter [DescribeSecurityGroups](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Daten über Instance-Typen

Das folgende Codebeispiel zeigt, wie Sie Daten zu Amazon EC2-Typen abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Describe the instance types available.
/// </summary>
/// <returns>A list of instance type information.</returns>
public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
{
    var request = new DescribeInstanceTypesRequest();

```

```
var filters = new List<Filter>
    { new Filter("processor-info.supported-architecture", new List<string>
    { architecture.ToString() }) };
filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

request.Filters = filters;
var instanceTypes = new List<InstanceTypeInfo>();

var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
await foreach (var instanceType in paginator.InstanceTypes)
{
    instanceTypes.Add(instanceType);
}
return instanceTypes;
}
```

- Weitere API-Informationen finden Sie unter [DescribeInstanceTypes](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Daten über das mit einer Instance verknüpfte Instance-Profil

Das folgende Codebeispiel zeigt, wie Sie Daten über das Instance-Profil abrufen, das einer Amazon-SEC-Instance zugeordnet ist.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
```

```
var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
    new DescribeIamInstanceProfileAssociationsRequest()
    {
        Filters = new List<Amazon.EC2.Model.Filter>()
        {
            new ("instance-id", new List<string>() { instanceId })
        },
    });
return response.IamInstanceProfileAssociations[0];
}
```

- Weitere API-Informationen finden Sie unter [DescribeIamInstanceProfileAssociations](#) in der APIAWS SDK for .NET -Referenz für .

Standard-VPC abrufen

Das folgende Codebeispiel zeigt, wie Sie die Standard-VPC des Girokontos abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
}
```

```
        return vpcResponse.Vpcs[0];
    }
```

- Weitere API-Informationen finden Sie unter [DescribeVpcs](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen der Standardsubnetze für eine VPC

Das folgende Codebeispiel zeigt, wie Sie die Standard-Subnetze für eine VPC abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    var subnets = new List<Subnet>();
    var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
        new DescribeSubnetsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("vpc-id", new List<string>() { vpcId}),
                new ("availability-zone", availabilityZones),
                new ("default-for-az", new List<string>() { "true" })
            }
        });

    // Get the entire list using the paginator.
```

```

        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

```

- Weitere API-Informationen finden Sie unter [DescribeSubnets](#) in der APIAWS SDK for .NET - Referenz für .

Sicherheitsschlüsselpaare auflisten

Das folgende Codebeispiel zeigt, wie Sie Amazon EC2-Sicherheitsschlüsselpaare auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Get information about an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair.</param>
/// <returns>A list of key pair information.</returns>
public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
{
    var request = new DescribeKeyPairsRequest();
    if (!string.IsNullOrEmpty(keyPairName))
    {
        request = new DescribeKeyPairsRequest
        {
            KeyNames = new List<string> { keyPairName }
        };
    }
    var response = await _amazonEC2.DescribeKeyPairsAsync(request);
    return response.KeyPairs.ToList();
}

```



```
}
```

- Weitere API-Informationen finden Sie unter [DescribeKeyPairs](#) in der APIAWS SDK for .NET - Referenz für .

Neustarten einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2 neu starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Reboot EC2 instances.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the instances that will be
rebooted.</param>
/// <returns>Async task.</returns>
public async Task RebootInstances(string ec2InstanceId)
{
    var request = new RebootInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.RebootInstancesAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine("Instances successfully rebooted.");
    }
    else
    {
        Console.WriteLine("Could not reboot one or more instances.");
    }
}
```

```
}
```

Ersetzen Sie das Profil für eine Instance, starten Sie einen Webserver neu und starten Sie ihn neu.

```
/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
        new ReplaceIamInstanceProfileAssociationRequest()
        {
            AssociationId = associationId,
            IamInstanceProfile = new IamInstanceProfileSpecification()
            {
                Name = credsProfileName
            }
        }
    );
    // Allow time before resetting.
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        await _amazonEc2.RebootInstancesAsync(
            new RebootInstancesRequest(new List<string>() { instanceId }));
        Thread.Sleep(10000);

        var instancesPaginator =
            _amazonSsm.Paginators.DescribeInstanceInformation(
```

```

        new DescribeInstanceInformationRequest());
    // Get the entire list using the paginator.
    await foreach (var instance in
instancesPaginator.InstanceInformationList)
    {
        instanceReady = instance.InstanceId == instanceId;
        if (instanceReady)
        {
            break;
        }
    }
}
Console.WriteLine($"Sending restart command to instance {instanceId}");
await _amazonSsm.SendCommandAsync(
    new SendCommandRequest()
    {
        InstanceIds = new List<string>() { instanceId },
        DocumentName = "AWS-RunShellScript",
        Parameters = new Dictionary<string, List<string>>()
        {
            {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
        }
    });
Console.WriteLine($"Restarted the web server on instance {instanceId}");
}

```

- Weitere API-Informationen finden Sie unter [RebootInstances](#) in der APIAWS SDK for .NET - Referenz für .

Freigeben einer Elastic-IP-Adresse

Das folgende Codebeispiel zeigt, wie Sie eine Elastic IP-Adresse freigeben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

```

- Weitere API-Informationen finden Sie unter [ReleaseAddress](#) in der APIAWS SDK for .NET - Referenz zu .

Ersetzen Sie das mit einer Instance verknüpfte Instance-Profil

Im folgenden Codebeispiel wird gezeigt, wie das Instance-Profil, das einer Amazon-EC2-Instance zugeordnet ist, ersetzt wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>

```

```
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
        new ReplaceIamInstanceProfileAssociationRequest()
        {
            AssociationId = associationId,
            IamInstanceProfile = new IamInstanceProfileSpecification()
            {
                Name = credsProfileName
            }
        });
    // Allow time before resetting.
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        await _amazonEc2.RebootInstancesAsync(
            new RebootInstancesRequest(new List<string>() { instanceId }));
        Thread.Sleep(10000);

        var instancesPaginator =
        _amazonSsm.Paginators.DescribeInstanceInformation(
            new DescribeInstanceInformationRequest());
        // Get the entire list using the paginator.
        await foreach (var instance in
instancesPaginator.InstanceInformationList)
        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
```

```

        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
            DocumentName = "AWS-RunShellScript",
            Parameters = new Dictionary<string, List<string>>()
            {
                {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
            }
        });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

```

- Weitere API-Informationen finden Sie unter [ReplacelamInstanceProfileAssociation](#) in der APIAWS SDK for .NET -Referenz für .

Eingehende Regeln für eine Sicherheitsgruppe festlegen

Das folgende Codebeispiel zeigt, wie eingehende Regeln für eine Amazon EC2Sicherheitsgruppe festgelegt werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Authorize the local computer ingress to EC2 instances associated
/// with the virtual private cloud (VPC) security group.
/// </summary>
/// <param name="groupName">The name of the security group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
{
    // Get the IP address for the local computer.
    var ipAddress = await GetIpAddress();
    Console.WriteLine($"Your IP address is: {ipAddress}");
}

```

```

        var ipRanges = new List<IpRange> { new IpRange { CidrIp =
    $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Authorize the local computer for ingress to
    /// the Amazon EC2 SecurityGroup.
    /// </summary>
    /// <returns>The IPv4 address of the computer running the scenario.</returns>
    private static async Task<string> GetIpAddress()
    {
        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
    checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }
}

```

- Weitere API-Informationen finden Sie unter [AuthorizeSecurityGroupIngress](#) in der APIAWS SDK for .NET -Referenz für .

Starten einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2 starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Start an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
/// to start.</param>
/// <returns>Async task.</returns>
public async Task StartInstances(string ec2InstanceId)
{
    var request = new StartInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.StartInstancesAsync(request);

    if (response.StartingInstances.Count > 0)
    {
        var instances = response.StartingInstances;
        instances.ForEach(i =>
        {
            Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [StartInstances](#) in der APIAWS SDK for .NET - Referenz für .

Anhalten einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2 anhalten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Stop an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance to
/// stop.</param>
/// <returns>Async task.</returns>
public async Task StopInstances(string ec2InstanceId)
{
    // In addition to the list of instance Ids, the
    // request can also include the following properties:
    //     Force      When true, forces the instances to
    //                 stop but you must check the integrity
    //                 of the file system. Not recommended on
    //                 Windows instances.
    //     Hibernate  When true, hibernates the instance if the
    //                 instance was enabled for hibernation when
    //                 it was launched.
    var request = new StopInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.StopInstancesAsync(request);

    if (response.StoppingInstances.Count > 0)
    {
        var instances = response.StoppingInstances;
        instances.ForEach(i =>
        {
            Console.WriteLine($"Successfully stopped the EC2 Instance " +
                $"with InstanceID: {i.InstanceId}.");
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [StopInstances](#) in der APIAWS SDK for .NET - Referenz für .

Beenden einer Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon EC2 beenden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Terminate an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance
/// to terminate.</param>
/// <returns>Async task.</returns>
public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
{
    var request = new TerminateInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId }
    };

    var response = await _amazonEC2.TerminateInstancesAsync(request);
    return response.TerminatingInstances;
}
```

- Weitere API-Informationen finden Sie unter [TerminateInstances](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();
}
```

```
// Set up dependency injection for the AWS services.
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<IAmazonIdentityManagementService>()
            .AddAWSService<IAmazonDynamoDB>()
            .AddAWSService<IAmazonElasticLoadBalancingV2>()
            .AddAWSService<IAmazonSimpleSystemsManagement>()
            .AddAWSService<IAmazonAutoScaling>()
            .AddAWSService<IAmazonEC2>()
            .AddTransient<AutoScalerWrapper>()
            .AddTransient<ElasticLoadBalancerWrapper>()
            .AddTransient<SmParameterWrapper>()
            .AddTransient<Recommendations>()
            .AddSingleton<IConfiguration>(_configuration)
        )
    .Build();

ServicesSetup(host);
ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await DestroyResources(true);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _recommendations = host.Services.GetRequiredService<Recommendations>();
    _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
    _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
}

/// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
```

```
/// <returns>True if successful.</returns>
public static async Task<bool> Deploy(bool interactive)
{
    var protocol = "HTTP";
    var port = 80;
    var sshPort = 22;

    Console.WriteLine(
        "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
        "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
        "against various kinds of failures.\n\n" +
        "Some of the resources create by this demo are:\n");

    Console.WriteLine(
        "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
    Console.WriteLine(
        "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
    Console.WriteLine(
        "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
    Console.WriteLine(
        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
    if (interactive)
        Console.ReadLine();

    // Create and populate the DynamoDB table.
    var databaseTableName = _configuration["databaseName"];
    var recommendationsPath = Path.Join(_configuration["resourcePath"],
        "recommendations_objects.json");
    Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
    await _recommendations.CreateDatabaseWithName(databaseTableName);
    await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
    Console.WriteLine(new string('-', 80));
}
```

```
// Create the EC2 Launch Template.

Console.WriteLine(
    $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
    + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
    + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
    + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
    + "run a web server, such as Apache, with least-privileged
credentials.");
Console.WriteLine(
    "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
    + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
    + "that control the flow of the demo.");

var startupScriptPath = Path.Join(_configuration["resourcePath"],
    "server_startup_script.sh");
var instancePolicyPath = Path.Join(_configuration["resourcePath"],
    "instance_policy.json");
await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
    + "Availability Zone.\n");
var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
    + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

Console.WriteLine(new string('-', 80));
```

```
Console.WriteLine("Press Enter when you're ready to continue.");
if (interactive)
    Console.ReadLine();

Console.WriteLine("Creating variables that control the flow of the demo.");
await _smParameterWrapper.Reset();

Console.WriteLine(
    "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
    + "defines how the load balancer connects to instances. The load
balancer provides a\n"
    + "single endpoint where clients connect and dispatches requests to
instances in the group.");

var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
var subnetIds = subnets.Select(s => s.SubnetId).ToList();
var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
subnetIds, targetGroup);
await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
Console.WriteLine("\nVerifying access to the load balancer endpoint...");
var endPoint = await
_elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
var loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

if (!loadBalancerAccess)
{
    Console.WriteLine("\nCouldn't connect to the load balancer, verifying
that the port is open...");

    var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
    ipString = ipString.Trim();
```



```
        var defaultSecurityGroup = await
_autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
        var portIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
        var sshPortIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
ipString);

        if (!portIsOpen)
        {
            Console.WriteLine(
                "\nFor this example to work, the default security group for your
default VPC must\n"
                + "allows access from this computer. You can either add it
automatically from this\n"
                + "example or add it yourself using the AWS Management Console.
\n");

            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
            }
        }

        if (!sshPortIsOpen)
        {
            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
            }
        }
        loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
```

```

        Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
        Console.WriteLine($"\\thttp://{endPoint}\\n");
    }
    else
    {
        Console.WriteLine(
            "\\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\\n"
            + "manually verifying that your VPC and security group are
configured correctly and that\\n"
            + "you can successfully make a GET request to the load balancer
endpoint:\\n");
        Console.WriteLine($"\\thttp://{endPoint}\\n");
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
    if (interactive)
        Console.ReadLine();
    return true;
}

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\\nThis part of the demonstration shows how to toggle
different parts of the system\\n" +
        "to create situations where the web service fails, and
shows how using a resilient\\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
}

```

```
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
    Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
    Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Let's reinstate the recommendation service.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
    Console.WriteLine(
        "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
```

```
        "access the DynamoDB recommendation table.\n"
    );
    await _autoScalerWrapper.CreateInstanceProfileWithName(
        _autoScalerWrapper.BadCredsPolicyName,
        _autoScalerWrapper.BadCredsRoleName,
        _autoScalerWrapper.BadCredsProfileName,
        ssmOnlyPolicy,
        new List<string> { "AmazonSSMManagedInstanceCore" }
    );
    var instances = await
    _autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
    var badInstanceId = instances.First();
    var instanceProfile = await
    _autoScalerWrapper.GetInstanceProfile(badInstanceId);
    Console.WriteLine(
        $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
        "bad credentials...\n"
    );
    await _autoScalerWrapper.ReplaceInstanceProfile(
        badInstanceId,
        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");
```

```
        Console.WriteLine("\nBy implementing deep health checks, the load balancer  
can detect when one of the instances is failing");  
        Console.WriteLine("and take that instance out of rotation.");  
  
        await  
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,  
"deep");  
  
        Console.WriteLine($" \nNow, checking target health indicates that the  
instance with bad credentials ({badInstanceId})");  
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for  
the load balancer to detect the unhealthy");  
        Console.WriteLine("instance. Sending a GET request to the load balancer  
endpoint always returns a recommendation, because");  
        Console.WriteLine("the load balancer takes unhealthy instances out of its  
rotation.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nBecause the instances in this demo are controlled by an  
auto scaler, the simplest way to fix an unhealthy");  
        Console.WriteLine("instance is to terminate it and let the auto scaler start  
a new instance to replace it.");  
  
        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);  
  
        Console.WriteLine($" \nEven while the instance is terminating and the new  
instance is starting, sending a GET");  
        Console.WriteLine("request to the web service continues to get a successful  
recommendation response because");  
        Console.WriteLine("starts and reports as healthy, it is included in the load  
balancing rotation.");  
        Console.WriteLine("Note that terminating and replacing an instance typically  
takes several minutes, during which time you");  
        Console.WriteLine("can see the changing health check status until the new  
instance is running and healthy.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nIf the recommendation service fails now, deep health  
checks mean all instances report as unhealthy.");
```

```
        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
        }
    }
}
```

```

        await
        _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
        _recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Erstellen Sie eine Klasse, die Auto-Scaling- und Amazon-EC2-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
}

```

```
private readonly string _badCredsRoleName = "";
private readonly string _badCredsPolicyName = "";
private readonly string _keyPairName = "";

public string GroupName => _groupName;
public string KeyPairName => _keyPairName;
public string LaunchTemplateName => _launchTemplateName;
public string InstancePolicyName => _instancePolicyName;
public string BadCredsProfileName => _badCredsProfileName;
public string BadCredsRoleName => _badCredsRoleName;
public string BadCredsPolicyName => _badCredsPolicyName;

/// <summary>
/// Constructor for the AutoScalerWrapper.
/// </summary>
/// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
/// <param name="amazonEc2">The injected EC2 client.</param>
/// <param name="amazonIam">The injected IAM client.</param>
/// <param name="amazonSsm">The injected SSM client.</param>
public AutoScalerWrapper(
    IAmazonAutoScaling amazonAutoScaling,
    IAmazonEC2 amazonEc2,
    IAmazonSimpleSystemsManagement amazonSsm,
    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}
```



```

}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,
    string ssmOnlyPolicyFile,
    List<string>? awsManagedPolicies = null)
{
    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": {" +
            "\"Service\": [" +
                "\"ec2.amazonaws.com\"" +
            "]" +
            "}, " +
            "\"Action\": \"sts:AssumeRole\"" +
        "}] " +
    "}";

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {

```

```
var createPolicyResult = await _amazonIam.CreatePolicyAsync(
    new CreatePolicyRequest
    {
        PolicyName = policyName,
        PolicyDocument = policyDocument
    });
policyArn = createPolicyResult.Policy.Arn;
}
catch (EntityAlreadyExistsException)
{
    // The policy already exists, so we look it up to get the Arn.
    var policiesPaginator = _amazonIam.Paginators.ListPolicies(
        new ListPoliciesRequest()
        {
            Scope = PolicyScopeType.Local
        });
    // Get the entire list using the paginator.
    await foreach (var policy in policiesPaginator.Policies)
    {
        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
```

```
        {
            foreach (var awsPolicy in awsManagedPolicies)
            {
                await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                {
                    PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                    RoleName = roleName
                });
            }
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Policy already exists.");
    var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
        new GetInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
}
```

```
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyPairName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyPairName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
        File.Delete($"{deleteKeyPairName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
    }
}
```

```
    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            }
        );
    }
}
```

```
    });
    return launchTemplateResponse.LaunchTemplate;
}

/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
    return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
}

/// <summary>
/// Create an EC2 Auto Scaling group of a specified size and name.
/// </summary>
/// <param name="groupSize">The size for the group.</param>
/// <param name="groupName">The name for the group.</param>
/// <param name="availabilityZones">The availability zones for the group.</
param>
/// <returns>Async task.</returns>
public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
{
    try
    {
        await _amazonAutoScaling.CreateAutoScalingGroupAsync(
            new CreateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                AvailabilityZones = availabilityZones,
                LaunchTemplate =
                    new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                    {
                        LaunchTemplateName = _launchTemplateName,
                        Version = "$Default"
                    },
                MaxSize = groupSize,
                MinSize = groupSize
            });
    }
}
```

```
        Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
    }
}

/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
    return vpcResponse.Vpcs[0];
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    var subnets = new List<Subnet>();
    var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
        new DescribeSubnetsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("vpc-id", new List<string>() { vpcId}),
                new ("availability-zone", availabilityZones),
            }
        });
    while (subnetPaginator.HasNext())
    {
        subnets.AddRange(subnetPaginator.CurrentPage.Items);
    }
}
```

```
        new ("default-for-az", new List<string>() { "true" })
    }
});

// Get the entire list using the paginator.
await foreach (var subnet in subnetPaginator.Subnets)
{
    subnets.Add(subnet);
}

return subnets;
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
```



```
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                });
            // Delete the custom policies only.
            if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
            {
                await _amazonIam.DeletePolicyAsync(
                    new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                    {
                        PolicyArn = policy.PolicyArn
                    });
            }
        }

        await _amazonIam.DeleteRoleAsync(
            new DeleteRoleRequest() { RoleName = roleName });
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine($"Instance profile {profileName} does not exist.");
    }
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
```

```
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
    new DescribeAutoScalingGroupsRequest()
    {
        AutoScalingGroupNames = new List<string>() { group }
    });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
        new DescribeIamInstanceProfileAssociationsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("instance-id", new List<string>() { instanceId })
            },
        });
    return response.IamInstanceProfileAssociations[0];
}

/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
```

```
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
        new ReplaceIamInstanceProfileAssociationRequest()
        {
            AssociationId = associationId,
            IamInstanceProfile = new IamInstanceProfileSpecification()
            {
                Name = credsProfileName
            }
        });
    // Allow time before resetting.
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        await _amazonEc2.RebootInstancesAsync(
            new RebootInstancesRequest(new List<string>() { instanceId }));
        Thread.Sleep(10000);

        var instancesPaginator =
        _amazonSsm.Paginators.DescribeInstanceInformation(
            new DescribeInstanceInformationRequest());
        // Get the entire list using the paginator.
        await foreach (var instance in
instancesPaginator.InstanceInformationList)
        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },

```

```
        DocumentName = "AWS-RunShellScript",
        Parameters = new Dictionary<string, List<string>>()
        {
            {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
        }
    });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                });
            stopping = true;
        }
        catch (ScalingActivityInProgressException)
        {
            Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
```

```
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                });
            stopped = true;
        }
        catch (Exception e)
            when ((e is ScalingActivityInProgressException)
                || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
```

```
        new UpdateAutoScalingGroupRequest()
        {
            AutoScalingGroupName = groupName,
            MinSize = 0
        });
    var group = describeGroupsResponse.AutoScalingGroups[0];
    foreach (var instance in group.Instances)
    {
        await TryTerminateInstanceById(instance.InstanceId);
    }

    await TryDeleteGroupByName(groupName);
}
else
{
    Console.WriteLine($"No groups found with name {groupName}.");
}
}

/// <summary>
/// Get the default security group for a specified Vpc.
/// </summary>
/// <param name="vpc">The Vpc to search.</param>
/// <returns>The default security group.</returns>
public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
{
    var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
        new DescribeSecurityGroupsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("group-name", new List<string>() { "default" }),
                new ("vpc-id", new List<string>() { vpc.VpcId })
            }
        });
    return groupResponse.SecurityGroups[0];
}

/// <summary>
/// Verify the default security group of a Vpc allows ingress from the calling
computer.
/// This can be done by allowing ingress from this computer's IP address.
```

```
/// In some situations, such as connecting from a corporate network, you must
instead specify
/// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
/// If you do, be sure to remove public access when you're done.
/// </summary>
/// <param name="vpc">The group to check.</param>
/// <param name="port">The port to verify.</param>
/// <param name="ipAddress">This computer's IP address.</param>
/// <returns>True if the ip address is allowed on the group.</returns>
public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
{
    var portIsOpen = false;
    foreach (var ipPermission in group.IpPermissions)
    {
        if (ipPermission.FromPort == port)
        {
            foreach (var ipRange in ipPermission.Ipv4Ranges)
            {
                var cidr = ipRange.CidrIp;
                if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                {
                    portIsOpen = true;
                }
            }

            if (ipPermission.PrefixListIds.Any())
            {
                portIsOpen = true;
            }

            if (!portIsOpen)
            {
                Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
            }
            else
            {
                break;
            }
        }
    }
}
```

```
    }

    return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()
            {
                new IpPermission()
                {
                    FromPort = port,
                    ToPort = port,
                    IpProtocol = "tcp",
                    Ipv4Ranges = new List<IpRange>()
                    {
                        new IpRange() { CidrIp = $"{ipAddress}/32" }
                    }
                }
            }
        });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
```



```

    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}

```

Erstellen Sie eine Klasse, die Elastic-Load-Balancing-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
    }
}

```

```
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```

    public async Task<List<TargetHealthDescription>>
    CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
    how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
    and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
    unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>

```

```
/// <returns>The new TargetGroup object.</returns>
public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
{
    var createResponse = await
_amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
    new CreateTargetGroupRequest()
    {
        Name = groupName,
        Protocol = protocol,
        Port = port,
        HealthCheckPath = "/healthcheck",
        HealthCheckIntervalSeconds = 10,
        HealthCheckTimeoutSeconds = 5,
        HealthyThresholdCount = 2,
        UnhealthyThresholdCount = 2,
        VpcId = vpcId
    });
    var targetGroup = createResponse.TargetGroups[0];
    return targetGroup;
}

/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
/// </summary>
/// <param name="name">The name for the new load balancer.</param>
/// <param name="subnetIds">Subnets for the load balancer.</param>
/// <param name="targetGroup">Target group for forwarded requests.</param>
/// <returns>The new LoadBalancer object.</returns>
public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
{
    var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
    new CreateLoadBalancerRequest()
    {
        Name = name,
        Subnets = subnetIds
    });
    var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

    // Wait for load balancer to be available.
```

```
var loadBalancerReady = false;
while (!loadBalancerReady)
{
    try
    {
        var describeResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });

        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}

///
```

```
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }

    return success;
}

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
```

```
try
{
    var describeLoadBalancerResponse =
        await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
            new DescribeLoadBalancersRequest()
            {
                Names = new List<string>() { name }
            }
        );
    var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
    await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
        new DeleteLoadBalancerRequest()
        {
            LoadBalancerArn = lbArn
        }
    );
}
catch (LoadBalancerNotFoundException)
{
    Console.WriteLine($"Load balancer {name} not found.");
}
}

/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    }
                );

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
```

```

        new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
        Console.WriteLine($"Deleted load balancing target group
{groupName}.");
        done = true;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine(
            $"Target group {groupName} not found, could not delete.");
        done = true;
    }
    catch (ResourceInUseException)
    {
        Console.WriteLine("Target group not yet released, waiting...");
        Thread.Sleep(10000);
    }
    }
}
}
}

```

Erstellen Sie eine Klasse, die DynamoDB zum Simulieren eines Empfehlungsservices verwendet.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {

```



```
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()
                {
                    AttributeName = "MediaType",
                    KeyType = KeyType.HASH
                },
                new KeySchemaElement()
                {
                    AttributeName = "ItemId",
                    KeyType = KeyType.RANGE
                }
            },
        },
    }
}
```

```
        ProvisionedThroughput = new ProvisionedThroughput()
        {
            ReadCapacityUnits = 5,
            WriteCapacityUnits = 5
        }
    };
    await _amazonDynamoDb.CreateTableAsync(createRequest);

    // Wait until the table is ACTIVE and then report success.
    Console.WriteLine("\nWaiting for table to become active...");

    var request = new DescribeTableRequest
    {
        TableName = tableName
    };

    TableStatus status;
    do
    {
        Thread.Sleep(2000);

        var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
        status = describeTableResponse.Table.TableStatus;

        Console.WriteLine(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
    return false;
}
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
```

```
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}
}
```

Erstellen Sie eine Klasse, die Systems-Manager-Aktionen umschließt.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
parameters
```

```
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }
}
```

```
/// <summary>
/// Set the value of a named Systems Manager parameter.
/// </summary>
/// <param name="name">The name of the parameter.</param>
/// <param name="value">The value to set.</param>
/// <returns>Async task.</returns>
public async Task PutParameterByName(string name, string value)
{
    await _amazonSimpleSystemsManagement.PutParameterAsync(
        new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)

- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Erste Schritte mit Instances

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Schlüsselpaar und eine Sicherheitsgruppe.
- Wählen Sie ein Amazon Machine Image (AMI) und einen kompatiblen Instance-Typ aus und erstellen Sie anschließend eine Instance.
- Halten Sie die Instance an und starten Sie sie neu.
- Verknüpfen einer Elastic-IP-Adresse mit der Instance.
- Stellen Sie über SSH eine Verbindung zu Ihrer Instance her und bereinigen Sie dann die Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein Szenario an einer Eingabeaufforderung aus.

```
/// <summary>
/// Show Amazon Elastic Compute Cloud (Amazon EC2) Basics actions.
/// </summary>
public class EC2Basics
{
    /// <summary>
```

```
/// Perform the actions defined for the Amazon EC2 Basics scenario.
/// </summary>
/// <param name="args">Command line arguments.</param>
/// <returns>A Task object.</returns>
static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EC2 and Amazon Simple Systems
    // Management Service.
    using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonEC2>()
            .AddAWSService<IAmazonSimpleSystemsManagement>()
            .AddTransient<EC2Wrapper>()
            .AddTransient<SsmWrapper>()
    )
    .Build();

    // Now the client is available for injection.
    var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();
    var ec2Methods = new EC2Wrapper(ec2Client);

    var ssmClient =
host.Services.GetRequiredService<IAmazonSimpleSystemsManagement>();
    var ssmMethods = new SsmWrapper(ssmClient);
    var uiMethods = new UiMethods();

    var uniqueName = Guid.NewGuid().ToString();
    var keyPairName = "mvp-example-key-pair" + uniqueName;
    var groupName = "ec2-scenario-group" + uniqueName;
    var groupDescription = "A security group created for the EC2 Basics
scenario.";

    // Start the scenario.
    uiMethods.DisplayOverview();
    uiMethods.PressEnter();

    // Create the key pair.
    uiMethods.DisplayTitle("Create RSA key pair");
    Console.Write("Let's create an RSA key pair that you can be use to ");
    Console.WriteLine("securely connect to your EC2 instance.");
    var keyPair = await ec2Methods.CreateKeyPair(keyPairName);

    // Save key pair information to a temporary file.
```

```
var tempFileName = ec2Methods.SaveKeyPair(keyPair);

Console.WriteLine($"Created the key pair: {keyPair.KeyName} and saved it to:
{tempFileName}");
string? answer;
do
{
    Console.Write("Would you like to list your existing key pairs? ");
    answer = Console.ReadLine();
} while (answer!.ToLower() != "y" && answer.ToLower() != "n");

if (answer == "y")
{
    // List existing key pairs.
    uiMethods.DisplayTitle("Existing key pairs");

    // Passing an empty string to the DescribeKeyPairs method will return
    // a list of all existing key pairs.
    var keyPairs = await ec2Methods.DescribeKeyPairs("");
    keyPairs.ForEach(kp =>
    {
        Console.WriteLine($"{kp.KeyName} created at: {kp.CreateTime}
Fingerprint: {kp.KeyFingerprint}");
    });
    uiMethods.PressEnter();

    // Create the security group.
    Console.WriteLine("Let's create a security group to manage access to your
instance.");
    var secGroupId = await ec2Methods.CreateSecurityGroup(groupName,
groupDescription);
    Console.WriteLine("Let's add rules to allow all HTTP and HTTPS inbound
traffic and to allow SSH only from your current IP address.");

    uiMethods.DisplayTitle("Security group information");
    var secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);

    Console.WriteLine($"Created security group {groupName} in your default
VPC.");
    secGroups.ForEach(group =>
    {
        ec2Methods.DisplaySecurityGroupInfoAsync(group);
    });
}
```



```
    uiMethods.PressEnter();

    Console.WriteLine("Now we'll authorize the security group we just created so
that it can");
    Console.WriteLine("access the EC2 instances you create.");
    var success = await ec2Methods.AuthorizeSecurityGroupIngress(groupName);

    secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);
    Console.WriteLine($"Now let's look at the permissions again.");
    secGroups.ForEach(group =>
    {
        ec2Methods.DisplaySecurityGroupInfoAsync(group);
    });
    uiMethods.PressEnter();

    // Get list of available Amazon Linux 2 Amazon Machine Images (AMIs).
    var parameters = await ssmMethods.GetParametersByPath("/aws/service/ami-
amazon-linux-latest");

    List<string> imageIds = parameters.Select(param => param.Value).ToList();

    var images = await ec2Methods.DescribeImages(imageIds);

    var i = 1;
    images.ForEach(image =>
    {
        Console.WriteLine($"{i++}\t{image.Description}");
    });

    int choice;
    bool validNumber = false;

    do
    {
        Console.Write("Please select an image: ");
        var selImage = Console.ReadLine();
        validNumber = int.TryParse(selImage, out choice);
    } while (!validNumber);

    var selectedImage = images[choice - 1];

    // Display available instance types.
    uiMethods.DisplayTitle("Instance Types");
```

```
var instanceTypes = await
ec2Methods.DescribeInstanceTypes(selectedImage.Architecture);

i = 1;
instanceTypes.ForEach(instanceType =>
{
    Console.WriteLine($"{i++}\t{instanceType.InstanceType}");
});

do
{
    Console.Write("Please select an instance type: ");
    var selImage = Console.ReadLine();
    validNumber = int.TryParse(selImage, out choice);
} while (!validNumber);

var selectedInstanceType = instanceTypes[choice - 1].InstanceType;

// Create an EC2 instance.
uiMethods.DisplayTitle("Creating an EC2 Instance");
var instanceId = await ec2Methods.RunInstances(selectedImage.ImageId,
selectedInstanceType, keyPairName, secGroupId);
Console.Write("Waiting for the instance to start.");
var isRunning = false;
do
{
    isRunning = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Running);
} while (!isRunning);

uiMethods.PressEnter();

var instance = await ec2Methods.DescribeInstance(instanceId);
uiMethods.DisplayTitle("New Instance Information");
ec2Methods.DisplayInstanceInformation(instance);

Console.WriteLine("\nYou can use SSH to connect to your instance. For
example:");
Console.WriteLine($"{i}\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

uiMethods.PressEnter();
```

```
        Console.WriteLine("Now we'll stop the instance and then start it again to  
see what's changed.");  
  
        await ec2Methods.StopInstances(instanceId);  
        var hasStopped = false;  
        do  
        {  
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,  
InstanceStateName.Stopped);  
        } while (!hasStopped);  
  
        Console.WriteLine("\nThe instance has stopped.");  
  
        Console.WriteLine("Now let's start it up again.");  
        await ec2Methods.StartInstances(instanceId);  
        Console.WriteLine("Waiting for instance to start. ");  
  
        isRunning = false;  
        do  
        {  
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,  
InstanceStateName.Running);  
        } while (!isRunning);  
  
        Console.WriteLine("\nLet's see what changed.");  
  
        instance = await ec2Methods.DescribeInstance(instanceId);  
        uiMethods.DisplayTitle("New Instance Information");  
        ec2Methods.DisplayInstanceInformation(instance);  
  
        Console.WriteLine("\nNotice the change in the SSH information:");  
        Console.WriteLine($"\\tssh -i {tempFileName} ec2-  
user@{instance.PublicIpAddress}");  
  
        uiMethods.PressEnter();  
  
        Console.WriteLine("Now we will stop the instance again. Then we will create  
and associate an");  
        Console.WriteLine("Elastic IP address to use with our instance.");  
  
        await ec2Methods.StopInstances(instanceId);  
        hasStopped = false;  
        do  
        {
```

```
        hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Stopped);
    } while (!hasStopped);

    Console.WriteLine("\nThe instance has stopped.");
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("Allocate Elastic IP address");
    Console.WriteLine("You can allocate an Elastic IP address and associate
it with your instance\nto keep a consistent IP address even when your instance
restarts.");
    var allocationId = await ec2Methods.AllocateAddress();
    Console.WriteLine("Now we will associate the Elastic IP address with our
instance.");
    var associationId = await ec2Methods.AssociateAddress(allocationId,
instanceId);

    // Start the instance again.
    Console.WriteLine("Now let's start the instance again.");
    await ec2Methods.StartInstances(instanceId);
    Console.WriteLine("Waiting for instance to start. ");

    isRunning = false;
    do
    {
        isRunning = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Running);
    } while (!isRunning);

    Console.WriteLine("\nLet's see what changed.");

    instance = await ec2Methods.DescribeInstance(instanceId);
    uiMethods.DisplayTitle("Instance information");
    ec2Methods.DisplayInstanceInformation(instance);

    Console.WriteLine("\nHere is the SSH information:");
    Console.WriteLine($"\"tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

    Console.WriteLine("Let's stop and start the instance again.");
    uiMethods.PressEnter();

    await ec2Methods.StopInstances(instanceId);
```

```
        hasStopped = false;
    do
    {
        hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Stopped);
    } while (!hasStopped);

    Console.WriteLine("\nThe instance has stopped.");

    Console.WriteLine("Now let's start it up again.");
    await ec2Methods.StartInstances(instanceId);
    Console.WriteLine("Waiting for instance to start. ");

    isRunning = false;
    do
    {
        isRunning = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Running);
    } while (!isRunning);

    instance = await ec2Methods.DescribeInstance(instanceId);
    uiMethods.DisplayTitle("New Instance Information");
    ec2Methods.DisplayInstanceInformation(instance);
    Console.WriteLine("Note that the IP address did not change this time.");
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("Clean up resources");

    Console.WriteLine("Now let's clean up the resources we created.");

    // Terminate the instance.
    Console.WriteLine("Terminating the instance we created.");
    var stateChange = await ec2Methods.TerminateInstances(instanceId);

    // Wait for the instance state to be terminated.
    var hasTerminated = false;
    do
    {
        hasTerminated = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Terminated);
    } while (!hasTerminated);

    Console.WriteLine($"The instance {instanceId} has been terminated.");
```

```
        Console.WriteLine("Now we can disassociate the Elastic IP address and
release it.");

        // Disassociate the Elastic IP address.
        var disassociated = ec2Methods.DisassociateIp(associationId);

        // Delete the Elastic IP address.
        var released = ec2Methods.ReleaseAddress(allocationId);

        // Delete the security group.
        Console.WriteLine($"Deleting the Security Group: {groupName}.");
        success = await ec2Methods.DeleteSecurityGroup(secGroupId);
        if (success)
        {
            Console.WriteLine($"Successfully deleted {groupName}.");
        }

        // Delete the RSA key pair.
        Console.WriteLine($"Deleting the key pair: {keyPairName}");
        await ec2Methods.DeleteKeyPair(keyPairName);
        Console.WriteLine("Deleting the temporary file with the key information.");
        ec2Methods.DeleteTempFile(tempFileName);
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("EC2 Basics Scenario completed.");
        uiMethods.PressEnter();
    }
}
```

Definieren Sie eine Klasse, die EC2-Aktionen umschließt.

```
/// <summary>
/// Methods of this class perform Amazon Elastic Compute Cloud (Amazon EC2).
/// </summary>
public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEC2;

    public EC2Wrapper(IAmazonEC2 amazonService)
    {
        _amazonEC2 = amazonService;
    }
}
```

```
/// <summary>
/// Allocate an Elastic IP address.
/// </summary>
/// <returns>The allocation Id of the allocated address.</returns>
public async Task<string> AllocateAddress()
{
    var request = new AllocateAddressRequest();

    var response = await _amazonEC2.AllocateAddressAsync(request);
    return response.AllocationId;
}

/// <summary>
/// Associate an Elastic IP address to an EC2 instance.
/// </summary>
/// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
/// <param name="instanceId">The instance Id of the EC2 instance to
/// associate the address with.</param>
/// <returns>The association Id that represents
/// the association of the Elastic IP address with an instance.</returns>
public async Task<string> AssociateAddress(string allocationId, string
instanceId)
{
    var request = new AssociateAddressRequest
    {
        AllocationId = allocationId,
        InstanceId = instanceId
    };

    var response = await _amazonEC2.AssociateAddressAsync(request);
    return response.AssociationId;
}

/// <summary>
/// Authorize the local computer ingress to EC2 instances associated
/// with the virtual private cloud (VPC) security group.
/// </summary>
/// <param name="groupName">The name of the security group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
{
    // Get the IP address for the local computer.
```

```

    var ipAddress = await GetIpAddress();
    Console.WriteLine($"Your IP address is: {ipAddress}");
    var ipRanges = new List<IpRange> { new IpRange { CidrIp =
"${ipAddress}/32" } };
    var permission = new IpPermission
    {
        Ipv4Ranges = ipRanges,
        IpProtocol = "tcp",
        FromPort = 22,
        ToPort = 22
    };
    var permissions = new List<IpPermission> { permission };
    var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Authorize the local computer for ingress to
/// the Amazon EC2 SecurityGroup.
/// </summary>
/// <returns>The IPv4 address of the computer running the scenario.</returns>
private static async Task<string> GetIpAddress()
{
    var httpClient = new HttpClient();
    var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

    // The IP address is returned with a new line
    // character on the end. Trim off the whitespace and
    // return the value to the caller.
    return ipString.Trim();
}

/// <summary>
/// Create an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name for the new key pair.</param>
/// <returns>The Amazon EC2 key pair created.</returns>
public async Task<KeyPair?> CreateKeyPair(string keyPairName)
{
    var request = new CreateKeyPairRequest
    {
        KeyName = keyPairName,

```



```
};

var response = await _amazonEC2.CreateKeyPairAsync(request);

if (response.HttpStatusCode == HttpStatusCode.OK)
{
    var kp = response.KeyPair;
    return kp;
}
else
{
    Console.WriteLine("Could not create key pair.");
    return null;
}
}

/// <summary>
/// Save KeyPair information to a temporary file.
/// </summary>
/// <param name="keyPair">The name of the key pair.</param>
/// <returns>The full path to the temporary file.</returns>
public string SaveKeyPair(KeyPair keyPair)
{
    var tempPath = Path.GetTempPath();
    var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
    var pemFileName = Path.ChangeExtension(tempFileName, "pem");

    // Save the key pair to a file in a temporary folder.
    using var stream = new FileStream(pemFileName, FileMode.Create);
    using var writer = new StreamWriter(stream);
    writer.WriteLine(keyPair.KeyMaterial);

    return pemFileName;
}

/// <summary>
/// Create an Amazon EC2 security group.
/// </summary>
/// <param name="groupName">The name for the new security group.</param>
/// <param name="groupDescription">A description of the new security group.</
param>
/// <returns>The group Id of the new security group.</returns>
public async Task<string> CreateSecurityGroup(string groupName, string
groupDescription)
```

```
{
    var response = await _amazonEC2.CreateSecurityGroupAsync(
        new CreateSecurityGroupRequest(groupName, groupDescription));

    return response.GroupId;
}

/// <summary>
/// Create a new Amazon EC2 VPC.
/// </summary>
/// <param name="cidrBlock">The CIDR block for the new security group.</param>
/// <returns>The VPC Id of the new VPC.</returns>
public async Task<string?> CreateVPC(string cidrBlock)
{

    try
    {
        var response = await _amazonEC2.CreateVpcAsync(new CreateVpcRequest
        {
            CidrBlock = cidrBlock,
        });

        Vpc vpc = response.Vpc;
        Console.WriteLine($"Created VPC with ID: {vpc.VpcId}.");
        return vpc.VpcId;
    }
    catch (AmazonEC2Exception ex)
    {
        Console.WriteLine($"Couldn't create VPC because: {ex.Message}");
        return null;
    }
}

/// <summary>
/// Delete an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyPair(string keyPairName)
{
    try
    {
```

```
        await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Delete the temporary file where the key pair information was saved.
/// </summary>
/// <param name="tempFileName">The path to the temporary file.</param>
public void DeleteTempFile(string tempFileName)
{
    if (File.Exists(tempFileName))
    {
        File.Delete(tempFileName);
    }
}

/// <summary>
/// Delete an Amazon EC2 security group.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteSecurityGroup(string groupId)
{
    var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an Amazon EC2 VPC.
/// </summary>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteVpc(string vpcId)
{
    var request = new DeleteVpcRequest
    {
```

```
        VpcId = vpcId,
    };

    var response = await _amazonEC2.DeleteVpcAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Get information about existing Amazon EC2 images.
/// </summary>
/// <returns>A list of image information.</returns>
public async Task<List<Image>> DescribeImages(List<string>? imageIds)
{
    var request = new DescribeImagesRequest();
    if (imageIds is not null)
    {
        // If the imageIds list is not null, add the list
        // to the request object.
        request.ImageIds = imageIds;
    }

    var response = await _amazonEC2.DescribeImagesAsync(request);
    return response.Images;
}

/// <summary>
/// Display the information returned by DescribeImages.
/// </summary>
/// <param name="images">The list of image information to display.</param>
public void DisplayImageInfo(List<Image> images)
{
    images.ForEach(image =>
    {
        Console.WriteLine($"{image.Name} Created on: {image.CreationDate}");
    });
}

/// <summary>
/// Get information about an Amazon EC2 instance.
/// </summary>
/// <param name="instanceId">The instance Id of the EC2 instance.</param>
/// <returns>An EC2 instance.</returns>
```

```
public async Task<Instance> DescribeInstance(string instanceId)
{
    var response = await _amazonEC2.DescribeInstancesAsync(
        new DescribeInstancesRequest { InstanceIds = new List<string>
{ instanceId } });
    return response.Reservations[0].Instances[0];
}

/// <summary>
/// Display EC2 instance information.
/// </summary>
/// <param name="instance">The instance Id of the EC2 instance.</param>
public void DisplayInstanceInformation(Instance instance)
{
    Console.WriteLine($"ID: {instance.InstanceId}");
    Console.WriteLine($"Image ID: {instance.ImageId}");
    Console.WriteLine($"{{instance.InstanceType}}");
    Console.WriteLine($"Key Name: {instance.KeyName}");
    Console.WriteLine($"VPC ID: {instance.VpcId}");
    Console.WriteLine($"Public IP: {instance.PublicIpAddress}");
    Console.WriteLine($"State: {instance.State.Name}");
}

/// <summary>
/// Get information about existing EC2 images.
/// </summary>
/// <returns>Async task.</returns>
public async Task DescribeInstances()
{
    // List all EC2 instances.
    await GetInstanceDescriptions();

    string tagName = "IncludeInList";
    string tagValue = "Yes";
    await GetInstanceDescriptionsFiltered(tagName, tagValue);
}

/// <summary>
/// Get information for all existing Amazon EC2 instances.
/// </summary>
/// <returns>Async task.</returns>
public async Task GetInstanceDescriptions()
{
    Console.WriteLine("Showing all instances:");
```

```
var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

await foreach (var response in paginator.Responses)
{
    foreach (var reservation in response.Reservations)
    {
        foreach (var instance in reservation.Instances)
        {
            Console.WriteLine($"Instance ID: {instance.InstanceId}");
            Console.WriteLine($"Current State: {instance.State.Name}");
        }
    }
}

/// <summary>
/// Get information about EC2 instances filtered by a tag name and value.
/// </summary>
/// <param name="tagName">The name of the tag to filter on.</param>
/// <param name="tagValue">The value of the tag to look for.</param>
/// <returns>Async task.</returns>
public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
{
    // This tag filters the results of the instance list.
    var filters = new List<Filter>
    {
        new Filter
        {
            Name = $"tag:{tagName}",
            Values = new List<string>
            {
                tagValue,
            },
        },
    };
    var request = new DescribeInstancesRequest
    {
        Filters = filters,
    };

    Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to
\"Yes\".");
}
```

```
var paginator = _amazonEC2.Paginators.DescribeInstances(request);

await foreach (var response in paginator.Responses)
{
    foreach (var reservation in response.Reservations)
    {
        foreach (var instance in reservation.Instances)
        {
            Console.WriteLine($"Instance ID: {instance.InstanceId} ");
            Console.WriteLine($"\\tCurrent State: {instance.State.Name}");
        }
    }
}

/// <summary>
/// Describe the instance types available.
/// </summary>
/// <returns>A list of instance type information.</returns>
public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
{
    var request = new DescribeInstanceTypesRequest();

    var filters = new List<Filter>
        { new Filter("processor-info.supported-architecture", new List<string>
{ architecture.ToString() }) };
    filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

    request.Filters = filters;
    var instanceTypes = new List<InstanceTypeInfo>();

    var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
    await foreach (var instanceType in paginator.InstanceTypes)
    {
        instanceTypes.Add(instanceType);
    }
    return instanceTypes;
}

/// <summary>
/// Display the instance type information returned by
DescribeInstanceTypesAsync.
/// </summary>
```

```
/// <param name="instanceTypes">The list of instance type information.</param>
public void DisplayInstanceTypeInfo(List<InstanceTypeInfo> instanceTypes)
{
    instanceTypes.ForEach(type =>
    {
        Console.WriteLine($"{type.InstanceType}\t{type.MemoryInfo}");
    });
}

/// <summary>
/// Get information about an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair.</param>
/// <returns>A list of key pair information.</returns>
public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
{
    var request = new DescribeKeyPairsRequest();
    if (!string.IsNullOrEmpty(keyPairName))
    {
        request = new DescribeKeyPairsRequest
        {
            KeyNames = new List<string> { keyPairName }
        };
    }
    var response = await _amazonEC2.DescribeKeyPairsAsync(request);
    return response.KeyPairs.ToList();
}

/// <summary>
/// Retrieve information for an Amazon EC2 security group.
/// </summary>
/// <param name="groupId">The Id of the Amazon EC2 security group.</param>
/// <returns>A list of security group information.</returns>
public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
{
    var request = new DescribeSecurityGroupsRequest();
    var groupIds = new List<string> { groupId };
    request.GroupIds = groupIds;

    var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
    return response.SecurityGroups;
}
```



```
/// <summary>
/// Display the information returned by the call to
/// DescribeSecurityGroupsAsync.
/// </summary>
/// <param name="securityGroup">A list of security group information.</param>
public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
{
    Console.WriteLine($"{securityGroup.GroupName}");
    Console.WriteLine("Ingress permissions:");
    securityGroup.IpPermissions.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"  {range.CidrIp}"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.WriteLine($"  {range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"  {id.Id} "));

        Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");
    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"  {range.CidrIp}"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.WriteLine($"  {range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"  {id.Id} "));
    });
}
```

```
        Console.WriteLine($"{Environment.NewLine}To Port: {permission.ToPort}");
    });
}

/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    var response = await _amazonEC2.DisassociateAddressAsync(
        new DisassociateAddressRequest { AssociationId = associationId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Retrieve a list of available Amazon Linux images.
/// </summary>
/// <returns>A list of image information.</returns>
public async Task<List<Image>> GetEC2AmiList()
{
    var filter = new Filter { Name = "architecture", Values = new List<string>
{ "x86_64" } };
    var filters = new List<Filter> { filter };
    var response = await _amazonEC2.DescribeImagesAsync(new
DescribeImagesRequest { Filters = filters });
    return response.Images;
}

/// <summary>
/// Reboot EC2 instances.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the instances that will be
rebooted.</param>
/// <returns>Async task.</returns>
public async Task RebootInstances(string ec2InstanceId)
{
    var request = new RebootInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };
}
```

```
        var response = await _amazonEC2.RebootInstancesAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Instances successfully rebooted.");
        }
        else
        {
            Console.WriteLine("Could not reboot one or more instances.");
        }
    }

    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Create and run an EC2 instance.
    /// </summary>
    /// <param name="ImageId">The image Id of the image used as a basis for the
    /// EC2 instance.</param>
    /// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
    /// <param name="keyName">The name of the key pair to associate with the
    /// instance.</param>
    /// <param name="groupId">The Id of the Amazon EC2 security group that will be
    /// allowed to interact with the new EC2 instance.</param>
    /// <returns>The instance Id of the new EC2 instance.</returns>
    public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
    {
        var request = new RunInstancesRequest
```

```
    {
        ImageId = imageId,
        InstanceType = instanceType,
        KeyName = keyName,
        MinCount = 1,
        MaxCount = 1,
        SecurityGroupIds = new List<string> { groupId }
    };
    var response = await _amazonEC2.RunInstancesAsync(request);
    return response.Reservation.Instances[0].InstanceId;
}

/// <summary>
/// Start an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
/// to start.</param>
/// <returns>Async task.</returns>
public async Task StartInstances(string ec2InstanceId)
{
    var request = new StartInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.StartInstancesAsync(request);

    if (response.StartingInstances.Count > 0)
    {
        var instances = response.StartingInstances;
        instances.ForEach(i =>
        {
            Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
        });
    }
}

/// <summary>
/// Stop an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance to
/// stop.</param>
```

```
/// <returns>Async task.</returns>
public async Task StopInstances(string ec2InstanceId)
{
    // In addition to the list of instance Ids, the
    // request can also include the following properties:
    //     Force      When true, forces the instances to
    //                 stop but you must check the integrity
    //                 of the file system. Not recommended on
    //                 Windows instances.
    //     Hibernate  When true, hibernates the instance if the
    //                 instance was enabled for hibernation when
    //                 it was launched.
    var request = new StopInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.StopInstancesAsync(request);

    if (response.StoppingInstances.Count > 0)
    {
        var instances = response.StoppingInstances;
        instances.ForEach(i =>
        {
            Console.WriteLine($"Successfully stopped the EC2 Instance " +
                              $"with InstanceID: {i.InstanceId}.");
        });
    }
}

/// <summary>
/// Terminate an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance
/// to terminate.</param>
/// <returns>Async task.</returns>
public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
{
    var request = new TerminateInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId }
    };
};
```

```
        var response = await _amazonEC2.TerminateInstancesAsync(request);
        return response.TerminatingInstances;
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is running.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEC2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.WriteLine(". ");
        } while (!hasState);

        return hasState;
    }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)

- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Amazon-ECS-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon ECS verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon ECS

Das folgende Codebeispiel veranschaulicht die ersten Schritte mit Amazon ECS.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.Hosting;

namespace ECSActions;

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 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args).Build();

        // Now the client is available for injection.
        var amazonECSClient = new AmazonECSClient();

        // You can use await and any of the async methods to get a response.
        var response = await amazonECSClient.ListClustersAsync(new
ListClustersRequest { });

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
available in the your aws account");
        Console.WriteLine();
        foreach (var arn in response.ClusterArns.Take(5))
        {
            Console.WriteLine($"  \tARN: {arn}");
            Console.WriteLine($"  \tCluster Name: {arn.Split("/").Last()}");
            Console.WriteLine();
        }
    }
}
```



```
}  
}
```

- Weitere API-Informationen finden Sie unter [ListClusters](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Auflisten von Clustern

Das folgende Codebeispiel zeigt, wie Sie Ihre Amazon-ECS-Cluster auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// List cluster ARNs available.  
/// </summary>  
/// <returns>The ARN list of clusters.</returns>  
public async Task<List<string>> GetClusterARNsAsync()  
{  
  
    Console.WriteLine("Getting a list of all the clusters in your AWS  
account...");  
    List<string> clusterArnList = new List<string>();  
    // Get a list of all the clusters in your AWS account  
    try  
    {
```

```
        var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
    {
    });

    var clusterArns = listClustersResponse.ClusterArns;

    // Print the ARNs of the clusters
    await foreach (var clusterArn in clusterArns)
    {
        clusterArnList.Add(clusterArn);
    }

    if (clusterArnList.Count == 0)
    {
        _logger.LogWarning("No clusters found in your AWS account.");
    }
    return clusterArnList;
}
catch (Exception e)
{
    _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
    throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
}
}
```

- Weitere API-Informationen finden Sie unter [ListClusters](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Services in einem Cluster

Das folgende Codebeispiel zeigt, wie Sie Ihre Amazon-ECS-Services in einem Cluster auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List service ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of services in given cluster.</returns>
public async Task<List<string>> GetServiceARNsAsync(string clusterARN)
{
    List<string> serviceArns = new List<string>();

    var request = new ListServicesRequest
    {
        Cluster = clusterARN
    };
    // Call the ListServices API operation and get the list of service ARNs
    var serviceList = _ecsClient.Paginators.ListServices(request);

    await foreach (var serviceARN in serviceList.ServiceArns)
    {
        if (serviceARN is null)
            continue;

        serviceArns.Add(serviceARN);
    }

    if (serviceArns.Count == 0)
    {
        _logger.LogWarning($"No services found in cluster {clusterARN} .");
    }

    return serviceArns;
}
```

- Weitere API-Informationen finden Sie unter [ListServices](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Aufgaben in einem Cluster

Das folgende Codebeispiel zeigt, wie Sie Ihre Amazon-ECS-Aufgaben in einem Cluster auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List task ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of tasks in given cluster.</returns>
public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
{
    // Set up the request to describe the tasks in the service
    var listTasksRequest = new ListTasksRequest
    {
        Cluster = clusterARN
    };
    List<string> taskArns = new List<string>();

    // Call the ListTasks API operation and get the list of task ARNs
    var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

    await foreach (var task in tasks.TaskArns)
    {
        if (task is null)
            continue;

        taskArns.Add(task);
    }

    if (taskArns.Count == 0)
```

```
    {  
        _logger.LogWarning("No tasks found in cluster: " + clusterARN);  
    }  
  
    return taskArns;  
}
```

- Weitere API-Informationen finden Sie unter [ListTasks](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

ARN-Informationen für Cluster, Services und Aufgaben abrufen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Ruft eine Liste aller Cluster ab.
- Rufen Sie Services für einen Cluster ab.
- Rufen Sie Aufgaben für einen Cluster ab.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
using Amazon.ECS;  
using ECSActions;  
using Microsoft.Extensions.Hosting;  
using Microsoft.Extensions.Logging;  
using Microsoft.Extensions.Logging.Console;  
using Microsoft.Extensions.Logging.Debug;  
  
namespace ECSScenario;
```

```
public class ECSScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. List ECS Cluster ARNs.
        2. List services in every cluster
        3. List Task ARNs in every cluster.
    */

    private static ILogger logger = null!;
    private static ECSWrapper _ecsWrapper = 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))
            .Build();

        ILoggerFactory loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ECSScenario>();

        var loggerECSWarpper = LoggerFactory.Create(builder =>
        { builder.AddConsole(); })
            .CreateLogger<ECSWrapper>();

        var amazonECSClient = new AmazonECSClient();

        _ecsWrapper = new ECSWrapper(amazonECSClient, loggerECSWarpper);

        Console.WriteLine(new string('-', 80));
    }
}
```

```
Console.WriteLine("Welcome to the Amazon ECS example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    await ListClusterARNs();
    await ListServiceARNs();
    await ListTaskARNs();
}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
}
}

/// <summary>
/// List ECS Cluster ARNs
/// </summary>
private static async Task ListClusterARNs()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"1. List Cluster ARNs from ECS.");
    var arns = await _ecsWrapper.GetClusterARNsAsync();

    foreach (var arn in arns)
    {
        Console.WriteLine($"Cluster arn: {arn}");
        Console.WriteLine($"Cluster name: {arn.Split("/").Last()}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List services in every cluster
/// </summary>
private static async Task ListServiceARNs()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"2. List Service ARNs in every cluster.");
    var clusterARNs = await _ecsWrapper.GetClusterARNsAsync();
```

```
        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting services for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var serviceARNs = await _ecsWrapper.GetServiceARNsAsync(clusterARN);

            foreach (var serviceARN in serviceARNs)
            {
                Console.WriteLine($"Service arn: {serviceARN}");
                Console.WriteLine($"Service name: {serviceARN.Split("/").Last()}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List tasks in every cluster
    /// </summary>
    private static async Task ListTaskARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. List Task ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNsAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting tasks for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var taskARNs = await _ecsWrapper.GetTaskARNsAsync(clusterARN);

            foreach (var taskARN in taskARNs)
            {
                Console.WriteLine($"Task arn: {taskARN}");
            }
        }
        Console.WriteLine(new string('-', 80));
    }
}
```



```
}
```

Wrapper-Methoden, die vom Szenario aufgerufen werden, um Amazon-ECS-Aktionen zu verwalten.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.Logging;

namespace ECSActions;

public class ECSWrapper
{
    private readonly AmazonECSClient _ecsClient;
    private readonly ILogger<ECSWrapper> _logger;

    /// <summary>
    /// Constructor for the ECS wrapper.
    /// </summary>
    /// <param name="ecsClient">The injected ECS client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public ECSWrapper(AmazonECSClient ecsClient, ILogger<ECSWrapper> logger)

    {
        _logger = logger;
        _ecsClient = ecsClient;
    }

    /// <summary>
    /// List cluster ARNs available.
    /// </summary>
    /// <returns>The ARN list of clusters.</returns>
    public async Task<List<string>> GetClusterARNsAsync()
    {
        Console.WriteLine("Getting a list of all the clusters in your AWS
account...");
        List<string> clusterArnList = new List<string>();
        // Get a list of all the clusters in your AWS account
        try
        {
```

```
        var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
    {
    });

    var clusterArns = listClustersResponse.ClusterArns;

    // Print the ARNs of the clusters
    await foreach (var clusterArn in clusterArns)
    {
        clusterArnList.Add(clusterArn);
    }

    if (clusterArnList.Count == 0)
    {
        _logger.LogWarning("No clusters found in your AWS account.");
    }
    return clusterArnList;
}
catch (Exception e)
{
    _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
    throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
}
}

/// <summary>
/// List service ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of services in given cluster.</returns>
public async Task<List<string>> GetServiceARNSAsync(string clusterARN)
{
    List<string> serviceArns = new List<string>();

    var request = new ListServicesRequest
    {
        Cluster = clusterARN
    };
    // Call the ListServices API operation and get the list of service ARNs
    var serviceList = _ecsClient.Paginators.ListServices(request);
```

```
        await foreach (var serviceARN in serviceList.ServiceArns)
        {
            if (serviceARN is null)
                continue;

            serviceArns.Add(serviceARN);
        }

        if (serviceArns.Count == 0)
        {
            _logger.LogWarning($"No services found in cluster {clusterARN} .");
        }

        return serviceArns;
    }

    /// <summary>
    /// List task ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of tasks in given cluster.</returns>
    public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
    {
        // Set up the request to describe the tasks in the service
        var listTasksRequest = new ListTasksRequest
        {
            Cluster = clusterARN
        };
        List<string> taskArns = new List<string>();

        // Call the ListTasks API operation and get the list of task ARNs
        var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

        await foreach (var task in tasks.TaskArns)
        {
            if (task is null)
                continue;

            taskArns.Add(task);
        }

        if (taskArns.Count == 0)
        {
```

```
        _logger.LogWarning("No tasks found in cluster: " + clusterARN);
    }

    return taskArns;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [ListClusters](#)
 - [ListServices](#)
 - [ListTasks](#)

Beispiele für Elastic Load Balancing mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Elastic Load Balancing verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen eines Listeners für einen Load Balancer

Das folgende Codebeispiel zeigt, wie Sie einen Listener erstellen, der Anfragen von einem ELB Load Balancer an eine Zielgruppe weiterleitet.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
/// </summary>
/// <param name="name">The name for the new load balancer.</param>
/// <param name="subnetIds">Subnets for the load balancer.</param>
/// <param name="targetGroup">Target group for forwarded requests.</param>
/// <returns>The new LoadBalancer object.</returns>
public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
{
    var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
    new CreateLoadBalancerRequest()
    {
        Name = name,
        Subnets = subnetIds
    });
    var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

    // Wait for load balancer to be available.
    var loadBalancerReady = false;
    while (!loadBalancerReady)
    {
        try
        {
            var describeResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
        }
    }
}
```

```
        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}
```

- Weitere API-Informationen finden Sie unter [CreateListener](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Zielgruppe

Das folgende Codebeispiel zeigt, wie Sie eine ELB-Zielgruppe erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
/// <returns>The new TargetGroup object.</returns>
public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
{
    var createResponse = await
_amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
    new CreateTargetGroupRequest()
    {
        Name = groupName,
        Protocol = protocol,
        Port = port,
        HealthCheckPath = "/healthcheck",
        HealthCheckIntervalSeconds = 10,
        HealthCheckTimeoutSeconds = 5,
        HealthyThresholdCount = 2,
        UnhealthyThresholdCount = 2,
        VpcId = vpcId
    });
    var targetGroup = createResponse.TargetGroups[0];
}
```

```
        return targetGroup;
    }
```

- Weitere API-Informationen finden Sie unter [CreateTargetGroup](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines Application Load Balancers

Das folgende Codebeispiel zeigt, wie Sie einen ELB Application Load Balancer erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
/// </summary>
/// <param name="name">The name for the new load balancer.</param>
/// <param name="subnetIds">Subnets for the load balancer.</param>
/// <param name="targetGroup">Target group for forwarded requests.</param>
/// <returns>The new LoadBalancer object.</returns>
public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
{
    var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
    new CreateLoadBalancerRequest()
    {
        Name = name,
        Subnets = subnetIds
    });
    var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

    // Wait for load balancer to be available.
    var loadBalancerReady = false;
```



```
while (!loadBalancerReady)
{
    try
    {
        var describeResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });

        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}
```

- Weitere API-Informationen finden Sie unter [CreateLoadBalancer](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines -Load Balancers

Das folgende Codebeispiel zeigt, wie Sie einen ELB Load Balancer löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
    try
    {
        var describeLoadBalancerResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });
        var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
        await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
            new DeleteLoadBalancerRequest()
            {
                LoadBalancerArn = lbArn
            }
        );
    }
    catch (LoadBalancerNotFoundException)
    {
        Console.WriteLine($"Load balancer {name} not found.");
    }
}
```

```
    }  
}
```

- Weitere API-Informationen finden Sie unter [DeleteLoadBalancer](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Zielgruppe

Das folgende Codebeispiel zeigt, wie Sie eine ELB-Zielgruppe löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Delete a TargetGroup by its specified name.  
/// </summary>  
/// <param name="groupName">Name of the group to delete.</param>  
/// <returns>Async task.</returns>  
public async Task DeleteTargetGroupName(string groupName)  
{  
    var done = false;  
    while (!done)  
    {  
        try  
        {  
            var groupResponse =  
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(  
                    new DescribeTargetGroupsRequest()  
                    {  
                        Names = new List<string>() { groupName }  
                    });  
  
            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;  
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(  
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });  
        }  
    }  
}
```

```

        Console.WriteLine($"Deleted load balancing target group
{groupName}.");
        done = true;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine(
            $"Target group {groupName} not found, could not delete.");
        done = true;
    }
    catch (ResourceInUseException)
    {
        Console.WriteLine("Target group not yet released, waiting...");
        Thread.Sleep(10000);
    }
    }
}

```

- Weitere API-Informationen finden Sie unter [DeleteTargetGroup](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen des Endpunkts eines Load Balancers

Das folgende Codebeispiel zeigt, wie Sie den Endpunkt eines ELB Load Balancers abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Get the HTTP Endpoint of a load balancer by its name.
/// </summary>
/// <param name="loadBalancerName">The name of the load balancer.</param>
/// <returns>The HTTP endpoint.</returns>
public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)

```

```
{
    if (_endpoint == null)
    {
        var endpointResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { loadBalancerName }
                });
        _endpoint = endpointResponse.LoadBalancers[0].DNSName;
    }

    return _endpoint;
}
```

- Weitere API-Informationen finden Sie unter [DescribeLoadBalancers](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen des Zustands einer Zielgruppe

Das folgende Codebeispiel zeigt, wie Sie den Zustand von Instances in einer ELB-Zielgruppe abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the target health for a group by name.
/// </summary>
/// <param name="groupName">The name of the group.</param>
/// <returns>The collection of health descriptions.</returns>
public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
{
    List<TargetHealthDescription> result = null!;
```

```
try
{
    var groupResponse =
        await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
            new DescribeTargetGroupsRequest()
            {
                Names = new List<string>() { groupName }
            });
    var healthResponse =
        await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
            new DescribeTargetHealthRequest()
            {
                TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
            });
    ;
    result = healthResponse.TargetHealthDescriptions;
}
catch (TargetGroupNotFoundException)
{
    Console.WriteLine($"Target group {groupName} not found.");
}
return result;
}
```

- Weitere API-Informationen finden Sie unter [DescribeTargetHealth](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.

- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie AWS Systems Manager die Parameter aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    // Set up dependency injection for the AWS services.
    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<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonDynamoDB>()
                .AddAWSService<IAmazonElasticLoadBalancingV2>()
                .AddAWSService<IAmazonSimpleSystemsManagement>())
    ;
}
```

```
        .AddAWSService<IAmazonAutoScaling>()
        .AddAWSService<IAmazonEC2>()
        .AddTransient<AutoScalerWrapper>()
        .AddTransient<ElasticLoadBalancerWrapper>()
        .AddTransient<SmParameterWrapper>()
        .AddTransient<Recommendations>()
        .AddSingleton<IConfiguration>(_configuration)
    )
    .Build();

    ServicesSetup(host);
    ResourcesSetup();

    try
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
        Console.WriteLine(new string('-', 80));
        await Deploy(true);

        Console.WriteLine("Now let's begin the scenario.");
        Console.WriteLine(new string('-', 80));
        await Demo(true);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Finally, let's clean up our resources.");
        Console.WriteLine(new string('-', 80));

        await DestroyResources(true);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await DestroyResources(true);
        Console.WriteLine(new string('-', 80));
    }
}
```



```
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _recommendations = host.Services.GetRequiredService<Recommendations>();
    _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
    _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
}

/// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Deploy(bool interactive)
{
    var protocol = "HTTP";
    var port = 80;
    var sshPort = 22;

    Console.WriteLine(
        "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
        "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
        "against various kinds of failures.\n\n" +
        "Some of the resources create by this demo are:\n");
}
```

```
    Console.WriteLine(
        "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
    Console.WriteLine(
        "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
    Console.WriteLine(
        "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
    Console.WriteLine(
        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
    if (interactive)
        Console.ReadLine();

    // Create and populate the DynamoDB table.
    var databaseTableName = _configuration["databaseName"];
    var recommendationsPath = Path.Join(_configuration["resourcePath"],
        "recommendations_objects.json");
    Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
    await _recommendations.CreateDatabaseWithName(databaseTableName);
    await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
    Console.WriteLine(new string('-', 80));

    // Create the EC2 Launch Template.

    Console.WriteLine(
        $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
        + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
        + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
        + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
        + "run a web server, such as Apache, with least-privileged
credentials.");
    Console.WriteLine(
```

```
        "\n\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n\n"
        + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n\n"
        + "that control the flow of the demo.");

    var startupScriptPath = Path.Join(_configuration["resourcePath"],
        "server_startup_script.sh");
    var instancePolicyPath = Path.Join(_configuration["resourcePath"],
        "instance_policy.json");
    await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(
        "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n\n"
        + "Availability Zone.\n\n");
    var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
    await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(
        "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n\n"
        + "HTTP requests. You can see these instances in the console or continue
with the demo.\n\n");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to continue.");
    if (interactive)
        Console.ReadLine();

    Console.WriteLine("Creating variables that control the flow of the demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine(
        "\n\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n\n"
        + "defines how the load balancer connects to instances. The load
balancer provides a\n\n"
        + "single endpoint where clients connect and dispatches requests to
instances in the group.");
```

```
        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
            _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
            _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupProtocol,
            protocol, port, defaultVpc.VpcId);

        await
            _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadBalancerProtocol,
            subnetIds, targetGroup);
        await
            _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
            targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
            _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        var loadBalancerAccess = await
            _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
            that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
            checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
                _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
                _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
                _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
                ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
                    default VPC must\n"
```

```
        + "allows access from this computer. You can either add it
automatically from this\n"
        + "example or add it yourself using the AWS Management Console.
\n");

        if (!interactive || GetYesNoResponse(
            "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
        {
            await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
        }

        if (!sshPortIsOpen)
        {
            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
            }
        }
        loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
        Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
        Console.WriteLine($"http://{endPoint}\n");
    }
    else
    {
        Console.WriteLine(
            "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
            + "manually verifying that your VPC and security group are
configured correctly and that\n"
            + "you can successfully make a GET request to the load balancer
endpoint:\n");
    }
}
```

```
        Console.WriteLine($"\\thttp://{endPoint}\n");
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
    if (interactive)
        Console.ReadLine();
    return true;
}

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
        "to create situations where the web service fails, and
shows how using a resilient\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
}
```

```
        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
            "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
        _smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
        _autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
```

```
var badInstanceId = instances.First();
var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
Console.WriteLine(
    $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
    "bad credentials...\n"
);
await _autoScalerWrapper.ReplaceInstanceProfile(
    badInstanceId,
    _autoScalerWrapper.BadCredsProfileName,
    instanceProfile.AssociationId
);
Console.WriteLine(
    "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
    "depending on which instance is selected by the load balancer.\n"
);
if (interactive)
    await DemoActionChoices();

Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
Console.WriteLine("and take that instance out of rotation.");

await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

Console.WriteLine($"Now, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
```



```
        Console.WriteLine("instance. Sending a GET request to the load balancer  
endpoint always returns a recommendation, because");  
        Console.WriteLine("the load balancer takes unhealthy instances out of its  
rotation.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nBecause the instances in this demo are controlled by an  
auto scaler, the simplest way to fix an unhealthy");  
        Console.WriteLine("instance is to terminate it and let the auto scaler start  
a new instance to replace it.");  
  
        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);  
  
        Console.WriteLine($"Even while the instance is terminating and the new  
instance is starting, sending a GET");  
        Console.WriteLine("request to the web service continues to get a successful  
recommendation response because");  
        Console.WriteLine("starts and reports as healthy, it is included in the load  
balancing rotation.");  
        Console.WriteLine("Note that terminating and replacing an instance typically  
takes several minutes, during which time you");  
        Console.WriteLine("can see the changing health check status until the new  
instance is running and healthy.");  
  
        if (interactive)  
            await DemoActionChoices();  
  
        Console.WriteLine("\nIf the recommendation service fails now, deep health  
checks mean all instances report as unhealthy.");  
  
        await  
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-  
is-not-a-table");  
  
        Console.WriteLine($"When all instances are unhealthy, the load balancer  
continues to route requests even to");  
        Console.WriteLine("unhealthy instances, allowing them to fail open and  
return a static response rather than fail");  
        Console.WriteLine("closed and report failure to the customer.");  
  
        if (interactive)  
            await DemoActionChoices();
```

```

        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
                _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
                _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
                _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
                _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
                _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
                _recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +

```

```

        "Don't forget to delete them when you're done with them or you might
        incur unexpected charges."
    );
}

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Erstellen Sie eine Klasse, die Auto-Scaling- und Amazon-EC2-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>

```

```
/// Constructor for the AutoScalerWrapper.
/// </summary>
/// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
/// <param name="amazonEc2">The injected EC2 client.</param>
/// <param name="amazonIam">The injected IAM client.</param>
/// <param name="amazonSsm">The injected SSM client.</param>
public AutoScalerWrapper(
    IAmazonAutoScaling amazonAutoScaling,
    IAmazonEC2 amazonEc2,
    IAmazonSimpleSystemsManagement amazonSsm,
    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance.The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
```

```

    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
            "\"Version\": \"2012-10-17\"," +
            "\"Statement\": [{" +
                "\"Effect\": \"Allow\"," +
                "\"Principal\": {" +
                "\"Service\": [" +
                    "\"ec2.amazonaws.com\"" +
                "]" +
                "}," +
                "\"Action\": \"sts:AssumeRole\"" +
            "]" +
            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()

```

```
        {
            Scope = PolicyScopeType.Local
        });
// Get the entire list using the paginator.
await foreach (var policy in policiesPaginator.Policies)
{
    if (policy.PolicyName.Equals(policyName))
    {
        policyArn = policy.Arn;
    }
}

if (policyArn == null)
{
    throw new InvalidOperationException("Policy not found");
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
```

```
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Policy already exists.");
    var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
        new GetInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    profileArn = profileGetResponse.InstanceProfile.Arn;
}
return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
```

```
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyPairName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyPairName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
        File.Delete($"{deleteKeyPairName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
    }
}

/// <summary>
/// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
/// The launch template specifies a Bash script in its user data field that runs
after
/// the instance is started. This script installs the Python packages and starts
a Python
/// web server on the instance.
/// </summary>
/// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
/// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
```



```
/// <returns>The template object.</returns>
public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
{
    await CreateKeyPair(_keyPairName);
    await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

    var startServerText = await File.ReadAllTextAsync(startupScriptPath);
    var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

    var amiLatest = await _amazonSsm.GetParameterAsync(
        new GetParameterRequest() { Name = _amiParam });
    var amiId = amiLatest.Parameter.Value;
    var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
        new CreateLaunchTemplateRequest()
        {
            LaunchTemplateName = _launchTemplateName,
            LaunchTemplateData = new RequestLaunchTemplateData()
            {
                InstanceType = _instanceType,
                ImageId = amiId,
                IamInstanceProfile =
                    new
                        LaunchTemplateIamInstanceProfileSpecificationRequest()
                    {
                        Name = _instanceProfileName
                    },
                KeyName = _keyPairName,
                UserData = System.Convert.ToBase64String(plainTextBytes)
            }
        });
    return launchTemplateResponse.LaunchTemplate;
}

/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
}
```

```
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
    List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
```

```
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
    return vpcResponse.Vpcs[0];
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    var subnets = new List<Subnet>();
    var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
        new DescribeSubnetsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("vpc-id", new List<string>() { vpcId}),
                new ("availability-zone", availabilityZones),
                new ("default-for-az", new List<string>() { "true" })
            }
        });

    // Get the entire list using the paginator.
    await foreach (var subnet in subnetPaginator.Subnets)
    {
        subnets.Add(subnet);
    }

    return subnets;
}

/// <summary>
```

```
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
```

```
        await _amazonIam.DetachRolePolicyAsync(
            new DetachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policy.PolicyArn
            });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
    _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
        new DescribeAutoScalingGroupsRequest()
        {
            AutoScalingGroupNames = new List<string>() { group }
        });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}
```

```
/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
        new DescribeIamInstanceProfileAssociationsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("instance-id", new List<string>() { instanceId })
            },
        });
    return response.IamInstanceProfileAssociations[0];
}

/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
        new ReplaceIamInstanceProfileAssociationRequest()
        {
            AssociationId = associationId,
            IamInstanceProfile = new IamInstanceProfileSpecification()
            {
                Name = credsProfileName
            }
        })
}
```

```
    });
    // Allow time before resetting.
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        await _amazonEc2.RebootInstancesAsync(
            new RebootInstancesRequest(new List<string>() { instanceId }));
        Thread.Sleep(10000);

        var instancesPaginator =
            _amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
        // Get the entire list using the paginator.
        await foreach (var instance in
            instancesPaginator.InstanceInformationList)
        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
            DocumentName = "AWS-RunShellScript",
            Parameters = new Dictionary<string, List<string>>()
            {
                {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
            }
        });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
```

```
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                });
            stopping = true;
        }
        catch (ScalingActivityInProgressException)
        {
            Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                }
            );
            stopped = true;
        }
        catch (AutoScalingGroupNotFoundException)
        {
            Console.WriteLine($"Auto Scaling group {groupName} not found.");
        }
    }
}

```



```
        });
        stopped = true;
    }
    catch (Exception e)
        when ((e is ScalingActivityInProgressException)
            || (e is Amazon.AutoScaling.Model.ResourceInUseException))
    {
        Console.WriteLine($"Some instances are still running. Waiting...");
        Thread.Sleep(10000);
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
    else

```

```
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }

    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
    computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
    instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
    ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
```

```
    {
        if (ipPermission.FromPort == port)
        {
            foreach (var ipRange in ipPermission.Ipv4Ranges)
            {
                var cidr = ipRange.CidrIp;
                if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                {
                    portIsOpen = true;
                }
            }

            if (ipPermission.PrefixListIds.Any())
            {
                portIsOpen = true;
            }

            if (!portIsOpen)
            {
                Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
            }
            else
            {
                break;
            }
        }
    }

    return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
```

```

    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}

```

Erstellen Sie eine Klasse, die Elastic-Load-Balancing-Aktionen beinhaltet.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
```

```
        new DescribeLoadBalancersRequest()
        {
            Names = new List<string>() { loadBalancerName }
        });
        _endpoint = endpointResponse.LoadBalancers[0].DNSName;
    }

    return _endpoint;
}

/// <summary>
/// Return the GET response for an endpoint as text.
/// </summary>
/// <param name="endpoint">The endpoint for the request.</param>
/// <returns>The request response.</returns>
public async Task<string> GetEndPointResponse(string endpoint)
{
    var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
    var textResponse = await endpointResponse.Content.ReadAsStringAsync();
    return textResponse!;
}

/// <summary>
/// Get the target health for a group by name.
/// </summary>
/// <param name="groupName">The name of the group.</param>
/// <returns>The collection of health descriptions.</returns>
public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
{
    List<TargetHealthDescription> result = null!;
    try
    {
        var groupResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                new DescribeTargetGroupsRequest()
                {
                    Names = new List<string>() { groupName }
                });
        var healthResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                new DescribeTargetHealthRequest()
                {
```

```
        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
        });
    };
    result = healthResponse.TargetHealthDescriptions;
}
catch (TargetGroupNotFoundException)
{
    Console.WriteLine($"Target group {groupName} not found.");
}
return result;
}

/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
/// <returns>The new TargetGroup object.</returns>
public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
{
    var createResponse = await
_amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
    new CreateTargetGroupRequest()
    {
        Name = groupName,
        Protocol = protocol,
        Port = port,
        HealthCheckPath = "/healthcheck",
        HealthCheckIntervalSeconds = 10,
        HealthCheckTimeoutSeconds = 5,
        HealthyThresholdCount = 2,
        UnhealthyThresholdCount = 2,
```

```
        VpcId = vpcId
    });
    var targetGroup = createResponse.TargetGroups[0];
    return targetGroup;
}

/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
/// </summary>
/// <param name="name">The name for the new load balancer.</param>
/// <param name="subnetIds">Subnets for the load balancer.</param>
/// <param name="targetGroup">Target group for forwarded requests.</param>
/// <returns>The new LoadBalancer object.</returns>
public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
{
    var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
    new CreateLoadBalancerRequest()
    {
        Name = name,
        Subnets = subnetIds
    });
    var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

    // Wait for load balancer to be available.
    var loadBalancerReady = false;
    while (!loadBalancerReady)
    {
        try
        {
            var describeResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });

            var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;
```



```
        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}

/// <summary>
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
```

```
        Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

        if (endpointResponse.IsSuccessStatusCode)
        {
            success = true;
        }
        else
        {
            retries = 0;
        }
    }
    catch (HttpRequestException)
    {
        Console.WriteLine("Connection error, retrying...");
        retries--;
        Thread.Sleep(10000);
    }
}

return success;
}

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
    try
    {
        var describeLoadBalancerResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });
        var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
        await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
            new DeleteLoadBalancerRequest()
            {
                LoadBalancerArn = lbArn
            }
        );
    }
}
```

```
        );
    }
    catch (LoadBalancerNotFoundException)
    {
        Console.WriteLine($"Load balancer {name} not found.");
    }
}

/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
            Console.WriteLine($"Deleted load balancing target group
{groupName}.");
            done = true;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine(
                $"Target group {groupName} not found, could not delete.");
            done = true;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine("Target group not yet released, waiting...");
            Thread.Sleep(10000);
        }
    }
}
```

```

    }
  }
}

```

Erstellen Sie eine Klasse, die DynamoDB zum Simulieren eines Empfehlungsservices verwendet.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.WriteLine($"Creating table {tableName}...");

```

```
var createRequest = new CreateTableRequest()
{
    TableName = tableName,
    AttributeDefinitions = new List<AttributeDefinition>()
    {
        new AttributeDefinition()
        {
            AttributeName = "MediaType",
            AttributeType = ScalarAttributeType.S
        },
        new AttributeDefinition()
        {
            AttributeName = "ItemId",
            AttributeType = ScalarAttributeType.N
        }
    },
    KeySchema = new List<KeySchemaElement>()
    {
        new KeySchemaElement()
        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};

await _amazonDynamoDb.CreateTableAsync(createRequest);

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("\nWaiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = tableName
};
```

```
        TableStatus status;
    do
    {
        Thread.Sleep(2000);

        var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
        status = describeTableResponse.Table.TableStatus;

        Console.Write(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
    return false;
}
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}
```

```
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Erstellen Sie eine Klasse, die Systems-Manager-Aktionen umschließt.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
/// parameters
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";
}
```

```
public string TableParameter => _tableParameter;
public string TableName => _tableName;
public string HealthCheckParameter => _healthCheckParameter;
public string FailureResponseParameter => _failureResponseParameter;

/// <summary>
/// Constructor for the SmParameterWrapper.
/// </summary>
/// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
/// <param name="configuration">The injected configuration.</param>
public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
{
    _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Reset the Systems Manager parameters to starting values for the demo.
/// </summary>
/// <returns>Async task.</returns>
public async Task Reset()
{
    await this.PutParameterByName(_tableParameter, _tableName);
    await this.PutParameterByName(_failureResponseParameter, "none");
    await this.PutParameterByName(_healthCheckParameter, "shallow");
}

/// <summary>
/// Set the value of a named Systems Manager parameter.
/// </summary>
/// <param name="name">The name of the parameter.</param>
/// <param name="value">The value to set.</param>
/// <returns>Async task.</returns>
public async Task PutParameterByName(string name, string value)
{
    await _amazonSimpleSystemsManagement.PutParameterAsync(
        new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
}
}
```


- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplaceIamInstanceProfileAssociation](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

EventBridge -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden EventBridge.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo EventBridge

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von EventBridge beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.EventBridge;
using Amazon.EventBridge.Model;

namespace EventBridgeActions;

public static class HelloEventBridge
{
    static async Task Main(string[] args)
    {
```

```
var eventBridgeClient = new AmazonEventBridgeClient();

Console.WriteLine($"Hello Amazon EventBridge! Following are some of your
EventBuses:");
Console.WriteLine();

// You can use await and any of the async methods to get a response.
// Let's get the first five event buses.
var response = await eventBridgeClient.ListEventBusesAsync(
    new ListEventBusesRequest()
    {
        Limit = 5
    });

foreach (var eventBus in response.EventBuses)
{
    Console.WriteLine($"\\tEventBus: {eventBus.Name}");
    Console.WriteLine($"\\tArn: {eventBus.Arn}");
    Console.WriteLine($"\\tPolicy: {eventBus.Policy}");
    Console.WriteLine();
}
}
```

- Weitere API-Informationen finden Sie unter [ListEventBuses](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Hinzufügen eines Ziels

Das folgende Codebeispiel zeigt, wie Sie einem Amazon- EventBridge Ereignis ein Ziel hinzufügen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Fügen Sie ein Amazon-SNS-Thema als Ziel für eine Regel hinzu.

```
/// <summary>
/// Add an Amazon SNS target topic to a rule.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <param name="targetArn">The ARN of the Amazon SNS target.</param>
/// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    // Create the list of targets and add a new target.
    var targets = new List<Target>
    {
        new Target()
        {
            Arn = targetArn,
            Id = targetID
        }
    };

    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });

    if (response.FailedEntryCount > 0)
```

```

    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return targetID;
    }

```

Fügen Sie einen Eingabe-Transformator als Ziel für eine Regel hinzu.

```

/// <summary>
/// Update an Amazon S3 object created rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                InputPathsMap = new Dictionary<string, string>()
                {
                    {"bucket", "$.detail.bucket.name"},
                    {"time", "$.time"}
                },
                InputTemplate = @"\Notification: an object was uploaded to
bucket <bucket> at <time>.\\"

```

```
        }
    }
};
var response = await _amazonEventBridge.PutTargetsAsync(
    new PutTargetsRequest()
    {
        EventBusName = eventBusArn,
        Rule = ruleName,
        Targets = targets,
    });
if (response.FailedEntryCount > 0)
{
    response.FailedEntries.ForEach(e =>
    {
        _logger.LogError(
            $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
    });
}
return targetID;
}
```

- Weitere API-Informationen finden Sie unter [PutTargets](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen einer Regel

Das folgende Codebeispiel zeigt, wie Sie eine Amazon- EventBridge Regel erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Regel, die ausgelöst wird, wenn ein Objekt zu einem Amazon-Simple-Storage-Service-Bucket hinzugefügt wird.

```
/// <summary>
```

```

    /// Create a new event rule that triggers when an Amazon S3 object is created in
    a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
string bucketName)
    {
        string eventPattern = "{" +
            "\"source\": [\"aws.s3\"]," +
            "\"detail-type\": [\"Object Created\"]," +
            "\"detail\": {" +
                "\"bucket\": {" +
                    "\"name\": [\"" + bucketName + "\"" +
                "}" +
            "}" +
        "}]";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }

```

Erstellen Sie eine Regel, die ein benutzerdefiniertes Muster verwendet.

```

    /// <summary>
    /// Update a rule to use a custom defined event pattern.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <returns>The ARN of the updated rule.</returns>
    public async Task<string> UpdateCustomEventPattern(string ruleName)
    {

```

```
string customEventsPattern = "{" +
    "\"source\": [\"ExampleSource\"],\" +
    "\"detail-type\": [\"ExampleType\"]" +
    "}";

var response = await _amazonEventBridge.PutRuleAsync(
    new PutRuleRequest()
    {
        Name = ruleName,
        Description = "Custom test rule",
        EventPattern = customEventsPattern
    });

return response.RuleArn;
}
```

- Weitere API-Informationen finden Sie unter [PutRule](#) in der APIAWS SDK for .NET -Referenz für

Löschen einer Regel

Das folgende Codebeispiel zeigt, wie Sie eine Amazon- EventBridge Regel löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie eine Regel anhand ihres Namens.

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
```



```
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteRule](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben einer Regel

Das folgende Codebeispiel zeigt, wie Sie eine Amazon- EventBridge Regel beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie den Status einer Regel anhand der Regelbeschreibung ab.

```
/// <summary>
/// Get the state for a rule by the rule name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
/// <returns>The state of the rule.</returns>
public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
{
    var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
        new DescribeRuleRequest()
        {
            Name = ruleName,
            EventBusName = eventBusName
        }
    );
}
```

```
    });  
    return ruleResponse.State;  
}
```

- Weitere API-Informationen finden Sie unter [DescribeRule](#) in der APIAWS SDK for .NET - Referenz für .

Deaktivieren einer Regel

Das folgende Codebeispiel zeigt, wie Sie eine Amazon- EventBridge Regel deaktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Deaktivieren Sie eine Regel anhand ihres Regelnamens.

```
/// <summary>  
/// Disable a particular rule on an event bus.  
/// </summary>  
/// <param name="ruleName">The name of the rule.</param>  
/// <returns>True if successful.</returns>  
public async Task<bool> DisableRuleByName(string ruleName)  
{  
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(  
        new DisableRuleRequest()  
        {  
            Name = ruleName  
        });  
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;  
}
```

- Weitere API-Informationen finden Sie unter [DisableRule](#) in der APIAWS SDK for .NET - Referenz für .

Aktivieren einer Regel

Das folgende Codebeispiel zeigt, wie Sie eine Amazon- EventBridge Regel aktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Aktivieren Sie eine Regel anhand ihres Regelnamens.


```
/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
        new EnableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [EnableRule](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Regelnamen für ein Ziel

Das folgende Codebeispiel zeigt, wie Sie Amazon- EventBridge Regelnamen für ein Ziel auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie alle Regelnamen mithilfe des Ziels auf.

```
/// <summary>
/// List names of all rules matching a target.
/// </summary>
/// <param name="targetArn">The ARN of the target.</param>
/// <returns>The list of rule names.</returns>
public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
{
    var results = new List<string>();
    var request = new ListRuleNamesByTargetRequest()
    {
        TargetArn = targetArn
    };
    ListRuleNamesByTargetResponse response;
    do
    {
        response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
        results.AddRange(response.RuleNames);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Weitere API-Informationen finden Sie unter [ListRuleNamesByTarget](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Regeln

Das folgende Codebeispiel zeigt, wie Sie Amazon- EventBridge Regeln auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie alle Regeln für einen Event Bus auf.

```
/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
{
    var results = new List<Rule>();
    var request = new ListRulesRequest()
    {
        EventBusName = eventBusArn
    };
    // Get all of the pages of rules.
    ListRulesResponse response;
    do
    {
        response = await _amazonEventBridge.ListRulesAsync(request);
        results.AddRange(response.Rules);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Weitere API-Informationen finden Sie unter [ListRules](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Zielen für eine Regel

Das folgende Codebeispiel zeigt, wie Sie Amazon- EventBridge Ziele für eine Regel auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie alle Ziele für eine Regel mithilfe des Regelnamens auf.

```
/// <summary>
/// List all of the targets matching a rule by name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>The list of targets.</returns>
public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
{
    var results = new List<Target>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse response;
    do
    {
        response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        results.AddRange(response.Targets);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Weitere API-Informationen finden Sie unter [ListTargetsByRule](#) in der APIAWS SDK for .NET - Referenz für .

Entfernen von Zielen aus einer Regel

Das folgende Codebeispiel zeigt, wie Sie Amazon- EventBridge Ziele aus einer Regel entfernen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Entfernen Sie alle Ziele für eine Regel mithilfe des Regelnamens.

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
{
    var targetIds = new List<string>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse targetsResponse;
    do
    {
        targetsResponse = await
        _amazonEventBridge.ListTargetsByRuleAsync(request);
        targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
        request.NextToken = targetsResponse.NextToken;
    } while (targetsResponse.NextToken is not null);

    var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
        new RemoveTargetsRequest()
        {
            Rule = ruleName,
            Ids = targetIds
        });
}
```

```
        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [RemoveTargets](#) in der APIAWS SDK for .NET - Referenz für .

Senden von Ereignissen

Das folgende Codebeispiel zeigt, wie Sie Amazon- EventBridge Ereignisse senden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie ein Ereignis, das einem benutzerdefinierten Muster für eine Regel entspricht.

```
/// <summary>
/// Add an event to the event bus that includes an email, message, and time.
/// </summary>
/// <param name="email">The email to use in the event detail of the custom
event.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutCustomEmailEvent(string email)
{
    var eventDetail = new
    {
```



```
        userEmail = email,
        Message = "This event was generated by example code.",
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
                {
                    Source = "ExampleSource",
                    Detail = JsonSerializer.Serialize(eventDetail),
                    DetailType = "ExampleType"
                }
            }
        });

    return response.FailedEntryCount == 0;
}
```

- Weitere API-Informationen finden Sie unter [PutEvents](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit Regeln und Zielen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Regel und fügen Sie ihr ein Ziel hinzu.
- Aktivieren und deaktivieren Sie Regeln.
- Listen Sie Regeln und Ziele auf und aktualisieren Sie sie.
- Senden Sie Ereignisse und bereinigen Sie dann die Ressourcen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
public class EventBridgeScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET example performs the following tasks with Amazon EventBridge:
    - Create a rule.
    - Add a target to a rule.
    - Enable and disable rules.
    - List rules and targets.
    - Update rules and targets.
    - Send events.
    - Delete the rule.
    */

    private static ILogger logger = null!;
    private static EventBridgeWrapper _eventBridgeWrapper = null!;
    private static IConfiguration _configuration = null!;

    private static IAmazonIdentityManagementService? _iamClient = null!;
    private static IAmazonSimpleNotificationService? _snsClient = null!;
    private static IAmazonS3 _s3Client = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        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<IAmazonEventBridge>()
.AddAWSService<IAmazonIdentityManagementService>()
.AddAWSService<IAmazonS3>()
.AddAWSService<IAmazonSimpleNotificationService>()
.AddTransient<EventBridgeWrapper>()
)
.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<EventBridgeScenario>();

ServicesSetup(host);

string topicArn = "";
string roleArn = "";

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon EventBridge example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    roleArn = await CreateRole();

    await CreateBucketWithEventBridgeEvents();

    await AddEventRule(roleArn);

    await ListEventRules();

    topicArn = await CreateSnsTopic();

    var email = await SubscribeToSnsTopic(topicArn);

    await AddSnsTarget(topicArn);
```

```
        await ListTargets();

        await ListRulesForTarget(topicArn);

        await UploadS3File(_s3Client);

        await ChangeRuleState(false);

        await GetRuleState();

        await UpdateSnsEventRule(topicArn);

        await ChangeRuleState(true);

        await UploadS3File(_s3Client);

        await UpdateToCustomRule(topicArn);

        await TriggerCustomRule(email);

        await CleanupResources(topicArn);
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
        await CleanupResources(topicArn);
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("The Amazon EventBridge example scenario is complete.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _eventBridgeWrapper =
host.Services.GetRequiredService<EventBridgeWrapper>();
    _snsClient =
host.Services.GetRequiredService<IAmazonSimpleNotificationService>();
    _s3Client = host.Services.GetRequiredService<IAmazonS3>();
}
```

```
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    }

    /// <summary>
    /// Create a role to be used by EventBridge.
    /// </summary>
    /// <returns>The role Amazon Resource Name (ARN).</returns>
    public static async Task<string> CreateRole()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating a role to use with EventBridge and attaching
managed policy AmazonEventBridgeFullAccess.");
        Console.WriteLine(new string('-', 80));

        var roleName = _configuration["roleName"];

        var assumeRolePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                "\"Effect\": \"Allow\"," +
                                "\"Principal\": {" +
                                $"\"Service\": \"events.amazonaws.com\" +
                                "}," +
                                "\"Action\": \"sts:AssumeRole\" +
                                "}] +
                                "};

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = roleName
            });

        await _iamClient.AttachRolePolicyAsync(
            new AttachRolePolicyRequest()
            {
                PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
                RoleName = roleName
            });
        // Allow time for the role to be ready.
        Thread.Sleep(10000);
    }
}
```

```
        return roleResult.Role.Arn;
    }

    /// <summary>
    /// Create an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge
events enabled.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateBucketWithEventBridgeEvents()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating an S3 bucket with EventBridge events enabled.");

        var testBucketName = _configuration["testBucketName"];

        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            testBucketName);

        if (!bucketExists)
        {
            await _s3Client.PutBucketAsync(new PutBucketRequest()
            {
                BucketName = testBucketName,
                UseClientRegion = true
            });
        }

        await _s3Client.PutBucketNotificationAsync(new
PutBucketNotificationRequest()
        {
            BucketName = testBucketName,
            EventBridgeConfiguration = new EventBridgeConfiguration()
        });

        Console.WriteLine($"\\tAdded bucket {testBucketName} with EventBridge events
enabled.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and upload a file to an S3 bucket to trigger an event.
    /// </summary>
```

```
/// <returns>Async task.</returns>
private static async Task UploadS3File(IAmazonS3 s3Client)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Uploading a file to the test bucket. This will trigger a
subscription email.");

    var testBucketName = _configuration["testBucketName"];

    var fileName = $"example_upload_{DateTime.UtcNow.Ticks}.txt";

    // Create the file if it does not already exist.
    if (!File.Exists(fileName))
    {
        await using StreamWriter sw = File.CreateText(fileName);
        await sw.WriteLineAsync(
            "This is a sample file for testing uploads.");
    }

    await s3Client.PutObjectAsync(new PutObjectRequest()
    {
        FilePath = fileName,
        BucketName = testBucketName
    });

    Console.WriteLine($"\\tPress Enter to continue.");
    Console.ReadLine();

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create an Amazon Simple Notification Service (Amazon SNS) topic to use as an
EventBridge target.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string> CreateSnsTopic()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "Creating an Amazon Simple Notification Service (Amazon SNS) topic for
email subscriptions.");

    var topicName = _configuration["topicName"];
```

```

string topicPolicy = "{" +
    "\"Version\": \"2012-10-17\"," +
    "\"Statement\": [{" +
    "\"Sid\": \"EventBridgePublishTopic\"," +
    "\"Effect\": \"Allow\"," +
    "\"Principal\": {" +
    $ "\"Service\": \"events.amazonaws.com\"" +
    "}," +
    "\"Resource\": \"*\"," +
    "\"Action\": \"sns:Publish\"" +
    "}]}" +
    "};

var topicAttributes = new Dictionary<string, string>()
{
    { "Policy", topicPolicy }
};

var topicResponse = await _snsClient!.CreateTopicAsync(new
CreateTopicRequest()
{
    Name = topicName,
    Attributes = topicAttributes
});

Console.WriteLine($"Added topic {topicName} for email subscriptions.");

Console.WriteLine(new string('-', 80));

return topicResponse.TopicArn;
}

/// <summary>
/// Subscribe a user email to an SNS topic.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>The user's email.</returns>
private static async Task<string> SubscribeToSnsTopic(string topicArn)
{
    Console.WriteLine(new string('-', 80));
}

```



```
string email = "";
while (string.IsNullOrEmpty(email))
{
    Console.WriteLine("Enter your email to subscribe to the Amazon SNS
topic:");
    email = Console.ReadLine()!;
}

var subscriptions = new List<string>();
var paginatedSubscriptions =
_snsClient!.Paginators.ListSubscriptionsByTopic(
    new ListSubscriptionsByTopicRequest()
    {
        TopicArn = topicArn
    });

// Get the entire list using the paginator.
await foreach (var subscription in paginatedSubscriptions.Subscriptions)
{
    subscriptions.Add(subscription.Endpoint);
}

if (subscriptions.Contains(email))
{
    Console.WriteLine($"\\tYour email is already subscribed.");
    Console.WriteLine(new string('-', 80));
    return email;
}

await _snsClient.SubscribeAsync(new SubscribeRequest()
{
    TopicArn = topicArn,
    Protocol = "email",
    Endpoint = email
});

Console.WriteLine($"Use the link in the email you received to confirm your
subscription, then press Enter to continue.");

Console.ReadLine();

Console.WriteLine(new string('-', 80));
return email;
}
```

```
/// <summary>
/// Add a rule which triggers when a file is uploaded to an S3 bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role used by EventBridge.</param>
/// <returns>Async task.</returns>
private static async Task AddEventRule(string roleArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating an EventBridge event that sends an email when an
Amazon S3 object is created.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];

    await _eventBridgeWrapper.PutS3UploadRule(roleArn, eventRuleName,
testBucketName);
    Console.WriteLine($"\\tAdded event rule {eventRuleName} for bucket
{testBucketName}.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add an SNS target to the rule.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>Async task.</returns>
private static async Task AddSnsTarget(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Adding a target to the rule to that sends an email when
the rule is triggered.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];
    var topicName = _configuration["topicName"];
    await _eventBridgeWrapper.AddSnsTargetToRule(eventRuleName, topicArn);
    Console.WriteLine($"\\tAdded event rule {eventRuleName} with Amazon SNS
target {topicName} for bucket {testBucketName}.");

    Console.WriteLine(new string('-', 80));
}
```

```
/// <summary>
/// List the event rules on the default event bus.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListEventRules()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Current event rules:");

    var rules = await _eventBridgeWrapper.ListAllRulesForEventBus();
    rules.ForEach(r => Console.WriteLine($"{r.Name} Description:
{r.Description} State: {r.State}"));

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Update the event target to use a transform.
/// </summary>
/// <param name="topicArn">The SNS topic ARN target to update.</param>
/// <returns>Async task.</returns>
private static async Task UpdateSnsEventRule(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Let's update the event target with a transform.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];

    await
_eventBridgeWrapper.UpdateS3UploadRuleTargetWithTransform(eventRuleName, topicArn);
    Console.WriteLine($"{r.Name} Updated event rule {eventRuleName} with Amazon SNS
target {topicArn} for bucket {testBucketName}.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Update the rule to use a custom event pattern.
/// </summary>
/// <returns>Async task.</returns>
private static async Task UpdateToCustomRule(string topicArn)
{
    Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("Updating the event pattern to be triggered by a custom
event instead.");

        var eventRuleName = _configuration["eventRuleName"];

        await _eventBridgeWrapper.UpdateCustomEventPattern(eventRuleName);

        Console.WriteLine($"\\tUpdated event rule {eventRuleName} to custom
pattern.");
        await _eventBridgeWrapper.UpdateCustomRuleTargetWithTransform(eventRuleName,
            topicArn);

        Console.WriteLine($"\\tUpdated event target {topicArn}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Send rule events for a custom rule using the user's email address.
    /// </summary>
    /// <param name="email">The email address to include.</param>
    /// <returns>Async task.</returns>
    private static async Task TriggerCustomRule(string email)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Sending an event to trigger the rule. This will trigger a
subscription email.");

        await _eventBridgeWrapper.PutCustomEmailEvent(email);

        Console.WriteLine($"\\tEvents have been sent. Press Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the targets for a rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListTargets()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the targets for a particular rule.");
```

```
    var eventRuleName = _configuration["eventRuleName"];
    var targets = await _eventBridgeWrapper.ListAllTargetsOnRule(eventRuleName);
    targets.ForEach(t => Console.WriteLine($"\\tTarget: {t.Arn} Id: {t.Id} Input:
{t.Input}"));

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List all of the rules for a particular target.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>Async task.</returns>
private static async Task ListRulesForTarget(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("List all of the rules for a particular target.");

    var rules = await _eventBridgeWrapper.ListAllRuleNamesByTarget(topicArn);
    rules.ForEach(r => Console.WriteLine($"\\tRule: {r}"));

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Enable or disable a particular rule.
/// </summary>
/// <param name="isEnabled">True to enable the rule, otherwise false.</param>
/// <returns>Async task.</returns>
private static async Task ChangeRuleState(bool isEnabled)
{
    Console.WriteLine(new string('-', 80));
    var eventRuleName = _configuration["eventRuleName"];

    if (!isEnabled)
    {
        Console.WriteLine($"Disabling the rule: {eventRuleName}");
        await _eventBridgeWrapper.DisableRuleByName(eventRuleName);
    }
    else
    {
        Console.WriteLine($"Enabling the rule: {eventRuleName}");
        await _eventBridgeWrapper.EnableRuleByName(eventRuleName);
    }
}
```

```

    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get the current state of the rule.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetRuleState()
{
    Console.WriteLine(new string('-', 80));
    var eventRuleName = _configuration["eventRuleName"];

    var state = await _eventBridgeWrapper.GetRuleStateByRuleName(eventRuleName);
    Console.WriteLine($"Rule {eventRuleName} is in current state {state}.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic to clean up.</param>
/// <returns>Async task.</returns>
private static async Task CleanupResources(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    var eventRuleName = _configuration["eventRuleName"];
    if (GetYesNoResponse($"Delete all targets and event rule {eventRuleName}?
(y/n)"))
    {
        Console.WriteLine($"Removing all targets from the event rule.");
        await _eventBridgeWrapper.RemoveAllTargetsFromRule(eventRuleName);

        Console.WriteLine($"Deleting event rule.");
        await _eventBridgeWrapper.DeleteRuleByName(eventRuleName);
    }

    var topicName = _configuration["topicName"];
    if (GetYesNoResponse($"Delete Amazon SNS subscription topic {topicName}?
(y/n)"))

```

```
{
    Console.WriteLine($"\\tDeleting topic.");
    await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
    {
        TopicArn = topicArn
    });
}

var bucketName = _configuration["testBucketName"];
if (GetYesNoResponse($"\\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
{
    Console.WriteLine($"\\tDeleting bucket.");
    // Delete all objects in the bucket.
    var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
    {
        BucketName = bucketName
    });
    await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
    {
        BucketName = bucketName,
        Objects = deleteList.S3Objects
            .Select(o => new KeyVersion { Key = o.Key }).ToList()
    });
    // Now delete the bucket.
    await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
    {
        BucketName = bucketName
    });
}

var roleName = _configuration["roleName"];
if (GetYesNoResponse($"\\tDelete role {roleName}? (y/n)"))
{
    Console.WriteLine($"\\tDetaching policy and deleting role.");

    await _iamClient!.DetachRolePolicyAsync(new DetachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
    });

    await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
    {
```

```
        RoleName = roleName
    });
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to 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;
}
}
```

Erstellen Sie eine Klasse, die EventBridge Operationen umschließt.

```
/// <summary>
/// Wrapper for Amazon EventBridge operations.
/// </summary>
public class EventBridgeWrapper
{
    private readonly IAmazonEventBridge _amazonEventBridge;
    private readonly ILogger<EventBridgeWrapper> _logger;

    /// <summary>
    /// Constructor for the EventBridge wrapper.
    /// </summary>
    /// <param name="amazonEventBridge">The injected EventBridge client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public EventBridgeWrapper(IAmazonEventBridge amazonEventBridge,
        ILogger<EventBridgeWrapper> logger)
```



```
{
    _amazonEventBridge = amazonEventBridge;
    _logger = logger;
}

/// <summary>
/// Get the state for a rule by the rule name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
/// <returns>The state of the rule.</returns>
public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
{
    var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
        new DescribeRuleRequest()
        {
            Name = ruleName,
            EventBusName = eventBusName
        });
    return ruleResponse.State;
}

/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
        new EnableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Disable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
```

```
public async Task<bool> DisableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
{
    var results = new List<Rule>();
    var request = new ListRulesRequest()
    {
        EventBusName = eventBusArn
    };
    // Get all of the pages of rules.
    ListRulesResponse response;
    do
    {
        response = await _amazonEventBridge.ListRulesAsync(request);
        results.AddRange(response.Rules);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}

/// <summary>
/// List all of the targets matching a rule by name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>The list of targets.</returns>
public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
{
```

```
var results = new List<Target>();
var request = new ListTargetsByRuleRequest()
{
    Rule = ruleName
};
ListTargetsByRuleResponse response;
do
{
    response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
    results.AddRange(response.Targets);
    request.NextToken = response.NextToken;

} while (response.NextToken is not null);

return results;
}

/// <summary>
/// List names of all rules matching a target.
/// </summary>
/// <param name="targetArn">The ARN of the target.</param>
/// <returns>The list of rule names.</returns>
public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
{
    var results = new List<string>();
    var request = new ListRuleNamesByTargetRequest()
    {
        TargetArn = targetArn
    };
    ListRuleNamesByTargetResponse response;
    do
    {
        response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
        results.AddRange(response.RuleNames);
        request.NextToken = response.NextToken;

    } while (response.NextToken is not null);

    return results;
}

/// <summary>
/// Create a new event rule that triggers when an Amazon S3 object is created in
a bucket.
```

```

    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
string bucketName)
    {
        string eventPattern = "{" +
            "\"source\": [\"aws.s3\"],\" +
                "\"detail-type\": [\"Object Created\"],\" +
                "\"detail\": {\" +
                    "\"bucket\": {\" +
                        "\"name\": [\"" + bucketName + "\"" ]\" +
                    }\" +
                }\" +
            }";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

```

```
var targets = new List<Target>
{
    new Target()
    {
        Id = targetID,
        Arn = targetArn,
        InputTransformer = new InputTransformer()
        {
            InputPathsMap = new Dictionary<string, string>()
            {
                {"bucket", "$.detail.bucket.name"},
                {"time", "$.time"}
            },
            InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\\"
        }
    }
};
var response = await _amazonEventBridge.PutTargetsAsync(
    new PutTargetsRequest()
    {
        EventBusName = eventBusArn,
        Rule = ruleName,
        Targets = targets,
    });
if (response.FailedEntryCount > 0)
{
    response.FailedEntries.ForEach(e =>
    {
        _logger.LogError(
            $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
    });
}
return targetID;
}

/// <summary>
/// Update a custom rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
```

```
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateCustomRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                InputTemplate = "\"Notification: sample event was received.\"\""
            }
        }
    };
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });
    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }
    return targetID;
}

/// <summary>
/// Add an event to the event bus that includes an email, message, and time.
/// </summary>
/// <param name="email">The email to use in the event detail of the custom
event.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutCustomEmailEvent(string email)
```

```
{
    var eventDetail = new
    {
        UserEmail = email,
        Message = "This event was generated by example code.",
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
                {
                    Source = "ExampleSource",
                    Detail = JsonSerializer.Serialize(eventDetail),
                    DetailType = "ExampleType"
                }
            }
        });

    return response.FailedEntryCount == 0;
}

/// <summary>
/// Update a rule to use a custom defined event pattern.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <returns>The ARN of the updated rule.</returns>
public async Task<string> UpdateCustomEventPattern(string ruleName)
{
    string customEventsPattern = "{" +
        "\"source\": [\"ExampleSource\"]," +
        "\"detail-type\": [\"ExampleType\"]" +
        "}";

    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Custom test rule",
            EventPattern = customEventsPattern
        });
}
```

```
        return response.RuleArn;
    }

    /// <summary>
    /// Add an Amazon SNS target topic to a rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <param name="targetArn">The ARN of the Amazon SNS target.</param>
    /// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        // Create the list of targets and add a new target.
        var targets = new List<Target>
        {
            new Target()
            {
                Arn = targetArn,
                Id = targetID
            }
        };

        // Add the targets to the rule.
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });

        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
    }
}
```



```
        return targetID;
    }

    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
    {
        var targetIds = new List<string>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse targetsResponse;
        do
        {
            targetsResponse = await
                _amazonEventBridge.ListTargetsByRuleAsync(request);
            targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
            request.NextToken = targetsResponse.NextToken;
        } while (targetsResponse.NextToken is not null);

        var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
            new RemoveTargetsRequest()
            {
                Rule = ruleName,
                Ids = targetIds
            });

        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
                    {e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

```
}

/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [DeleteRule](#)
 - [DescribeRule](#)
 - [DisableRule](#)
 - [EnableRule](#)
 - [ListRuleNamesByTarget](#)
 - [ListRules](#)
 - [ListTargetsByRule](#)
 - [PutEvents](#)
 - [PutRule](#)
 - [PutTargets](#)

AWS Glue -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS Glue.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Glue

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Glue beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        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<IAmazonGlue>()
            .AddTransient<GlueWrapper>()
        )
        .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<HelloGlue>();
var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

var request = new ListJobsRequest();

var jobNames = new List<string>();

do
{
    var response = await glueClient.ListJobsAsync(request);
    jobNames.AddRange(response.JobNames);
    request.NextToken = response.NextToken;
}
while (request.NextToken is not null);

Console.Clear();
Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
if (jobNames.Count == 0)
{
    Console.WriteLine("You don't have any AWS Glue jobs.");
}
else
{
    jobNames.ForEach(Console.WriteLine);
}
}
```

- Weitere API-Informationen finden Sie unter [ListJobs](#) in der APIAWS SDK for .NET -Referenz für

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen eines Crawlers

Das folgende Codebeispiel zeigt, wie Sie einen AWS Glue -Crawler erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name for the crawler.</param>
/// <param name="crawlerDescription">A description of the crawler.</param>
/// <param name="role">The AWS Identity and Access Management (IAM) role to
/// be assumed by the crawler.</param>
/// <param name="schedule">The schedule on which the crawler will be executed.</
param>
/// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
S3)
/// bucket where the Python script has been stored.</param>
/// <param name="dbName">The name to use for the database that will be
/// created by the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateCrawlerAsync(
    string crawlerName,
    string crawlerDescription,
    string role,
    string schedule,
    string s3Path,
    string dbName)
{
    var s3Target = new S3Target
    {
        Path = s3Path,
    };

    var targetList = new List<S3Target>
```

```
    {
        s3Target,
    };

    var targets = new CrawlerTargets
    {
        S3Targets = targetList,
    };

    var crawlerRequest = new CreateCrawlerRequest
    {
        DatabaseName = dbName,
        Name = crawlerName,
        Description = crawlerDescription,
        Targets = targets,
        Role = role,
        Schedule = schedule,
    };


    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [CreateCrawler](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen Sie eine Auftragsdefinition

Das folgende Codebeispiel zeigt, wie Sie eine - AWS Glue Auftragsdefinition erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an AWS Glue job.
```

```
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="roleName">The name of the IAM role to be assumed by
/// the job.</param>
/// <param name="description">A description of the job.</param>
/// <param name="scriptUrl">The URL to the script.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
{
    var command = new JobCommand
    {
        PythonVersion = "3",
        Name = "glueetl",
        ScriptLocation = scriptUrl,
    };

    var arguments = new Dictionary<string, string>
    {
        { "--input_database", dbName },
        { "--input_table", tableName },
        { "--output_bucket_url", bucketUrl }
    };

    var request = new CreateJobRequest
    {
        Command = command,
        DefaultArguments = arguments,
        Description = description,
        GlueVersion = "3.0",
        Name = jobName,
        NumberOfWorkers = 10,
        Role = roleName,
        WorkerType = "G.1X"
    };

    var response = await _amazonGlue.CreateJobAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [CreateJob](#) in der APIAWS SDK for .NET -Referenz für .

Einen Crawler löschen

Das folgende Codebeispiel zeigt, wie Sie einen AWS Glue -Crawler löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteCrawlerAsync(string crawlerName)
{
    var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteCrawler](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Datenbank aus dem Data Catalog

Das folgende Codebeispiel zeigt, wie Sie eine Datenbank aus dem löschen AWS Glue Data Catalog.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete the AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteDatabaseAsync(string dbName)
{
    var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteDatabase](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Auftragsdefinition

Das folgende Codebeispiel zeigt, wie Sie eine - AWS Glue Auftragsdefinition und alle zugehörigen Ausführungen löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an AWS Glue job.
```

```
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteJobAsync(string jobName)
{
    var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteJob](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Tabelle aus einer Datenbank

Das folgende Codebeispiel zeigt, wie Sie eine Tabelle aus einer - AWS Glue Data Catalog Datenbank löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Delete a table from an AWS Glue database.
/// </summary>
/// <param name="tableName">The table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTableAsync(string dbName, string tableName)
{
    var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteTable](#) in der APIAWS SDK for .NET - Referenz für .

Holen Sie sich einen Crawler

Das folgende Codebeispiel zeigt, wie Sie einen AWS Glue -Crawler abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Crawler object describing the crawler.</returns>
public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new GetCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var databaseName = response.Crawler.DatabaseName;
        Console.WriteLine($"{crawlerName} has the database {databaseName}");
        return response.Crawler;
    }

    Console.WriteLine($"No information regarding {crawlerName} could be
found.");
    return null;
}
```

- Weitere API-Informationen finden Sie unter [GetCrawler](#) in der APIAWS SDK for .NET -Referenz für .

Holen Sie sich eine Datenbank aus dem Data Catalog

Das folgende Codebeispiel zeigt, wie Sie eine Datenbank aus der abrufen AWS Glue Data Catalog.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Database object containing information about the database.</
returns>
public async Task<Database> GetDatabaseAsync(string dbName)
{
    var databasesRequest = new GetDatabaseRequest
    {
        Name = dbName,
    };

    var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
    return response.Database;
}
```

- Weitere API-Informationen finden Sie unter [GetDatabase](#) in der APIAWS SDK for .NET -Referenz für .

Ausführen eines Auftrags

Das folgende Codebeispiel zeigt, wie Sie eine - AWS Glue Auftragsausführung abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Get information about a specific AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="jobRunId">The Id of the job run.</param>
/// <returns>A JobRun object with information about the job run.</returns>
public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
{
    var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
    { JobName = jobName, RunId = jobRunId });
    return response.JobRun;
}
```

- Weitere API-Informationen finden Sie unter [GetJobRun](#) in der APIAWS SDK for .NET -Referenz für .

Läufe eines Auftrags holen

Das folgende Codebeispiel zeigt, wie Sie Ausführungen eines - AWS Glue Auftrags abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Get information about all AWS Glue runs of a specific job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A list of JobRun objects.</returns>
public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };

    // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
    var paginatorForJobRuns =
        _amazonGlue.Paginators.GetJobRuns(request);

    await foreach (var response in paginatorForJobRuns.Responses)
    {
        response.JobRuns.ForEach(jobRun =>
        {
            jobRuns.Add(jobRun);
        });
    }

    return jobRuns;
}
```

- Weitere API-Informationen finden Sie unter [GetJobRuns](#) in der APIAWS SDK for .NET - Referenz für .

Tabellen aus einer Datenbank abrufen

Das folgende Codebeispiel zeigt, wie Tabellen aus einer Datenbank in abgerufen werden AWS Glue Data Catalog.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

    await foreach (var response in tablePaginator.Responses)
    {
        tables.AddRange(response.TableList);
    }

    return tables;
}
```

- Weitere API-Informationen finden Sie unter [GetTables](#) in der APIAWS SDK for .NET -Referenz für .

Liste der Auftragsdefinitionen

Das folgende Codebeispiel zeigt, wie Sie AWS Glue Auftragsdefinitionen auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List AWS Glue jobs using a paginator.
/// </summary>
/// <returns>A list of AWS Glue job names.</returns>
public async Task<List<string>> ListJobsAsync()
{
    var jobNames = new List<string>();

    var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
    await foreach (var response in listJobsPaginator.Responses)
    {
        jobNames.AddRange(response.JobNames);
    }

    return jobNames;
}
```

- Weitere API-Informationen finden Sie unter [ListJobs](#) in der APIAWS SDK for .NET -Referenz für

Starten eines Crawlers

Das folgende Codebeispiel zeigt, wie Sie einen AWS Glue -Crawler starten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);


    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [StartCrawler](#) in der APIAWS SDK for .NET - Referenz für .

Starten einer Auftragsausführung

Das folgende Codebeispiel zeigt, wie Sie eine - AWS Glue Auftragsausführung starten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- Weitere API-Informationen finden Sie unter [StartJobRun](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit Crawlern und Aufträgen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Crawler, der einen öffentlichen Amazon-S3-Bucket crawlt und eine Datenbank mit CSV-formatierten Metadaten generiert.
- Auflisten von Informationen zu Datenbanken und Tabellen in Ihrem AWS Glue Data Catalog.

- Erstellen Sie einen Auftrag, um CSV-Daten aus dem S3-Bucket zu extrahieren, die Daten umzuwandeln und die JSON-formatierte Ausgabe in einen anderen S3-Bucket zu laden.
- Listen Sie Informationen zu Auftragsausführungen auf, zeigen Sie transformierte Daten an und bereinigen Sie Ressourcen.

Weitere Informationen finden Sie unter [Tutorial: Erste Schritte mit AWS Glue Studio](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Klasse, die AWS Glue Funktionen umschließt, die im Szenario verwendet werden.

```
using System.Net;

namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
```

```
/// <param name="role">The AWS Identity and Access Management (IAM) role to
/// be assumed by the crawler.</param>
/// <param name="schedule">The schedule on which the crawler will be executed.</
param>
/// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
S3)
/// bucket where the Python script has been stored.</param>
/// <param name="dbName">The name to use for the database that will be
/// created by the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateCrawlerAsync(
    string crawlerName,
    string crawlerDescription,
    string role,
    string schedule,
    string s3Path,
    string dbName)
{
    var s3Target = new S3Target
    {
        Path = s3Path,
    };

    var targetList = new List<S3Target>
    {
        s3Target,
    };

    var targets = new CrawlerTargets
    {
        S3Targets = targetList,
    };

    var crawlerRequest = new CreateCrawlerRequest
    {
        DatabaseName = dbName,
        Name = crawlerName,
        Description = crawlerDescription,
        Targets = targets,
        Role = role,
        Schedule = schedule,
    };

    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
}
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

```
}

/// <summary>
/// Delete an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteCrawlerAsync(string crawlerName)
{
    var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete the AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteDatabaseAsync(string dbName)
{
    var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteJobAsync(string jobName)
{
    var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a table from an AWS Glue database.
```

```
/// </summary>
/// <param name="tableName">The table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTableAsync(string dbName, string tableName)
{
    var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get information about an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Crawler object describing the crawler.</returns>
public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new GetCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var databaseName = response.Crawler.DatabaseName;
        Console.WriteLine($"{crawlerName} has the database {databaseName}");
        return response.Crawler;
    }

    Console.WriteLine($"No information regarding {crawlerName} could be
found.");
    return null;
}

/// <summary>
/// Get information about the state of an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A value describing the state of the crawler.</returns>
public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
{
```

```
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }

    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }

    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }

    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
```



```
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };

    // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
    var paginatorForJobRuns =
        _amazonGlue.Paginators.GetJobRuns(request);

    await foreach (var response in paginatorForJobRuns.Responses)
    {
        response.JobRuns.ForEach(jobRun =>
        {
            jobRuns.Add(jobRun);
        });
    }

    return jobRuns;
}

/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

    await foreach (var response in tablePaginator.Responses)
    {
        tables.AddRange(response.TableList);
    }

    return tables;
}
```

```
}

/// <summary>
/// List AWS Glue jobs using a paginator.
/// </summary>
/// <returns>A list of AWS Glue job names.</returns>
public async Task<List<string>> ListJobsAsync()
{
    var jobNames = new List<string>();

    var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
    await foreach (var response in listJobsPaginator.Responses)
    {
        jobNames.AddRange(response.JobNames);
    }

    return jobNames;
}

/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
```

```
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
}
```

Erstellen Sie eine Klasse, die das Szenario ausführt.

```
global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
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.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;
```

```
namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        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<IAmazonGlue>()
                    .AddTransient<GlueWrapper>()
                    .AddTransient<UiWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<GlueBasics>();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // These values are stored in settings.json
        // Once you have run the CDK script to deploy the resources,
        // edit the file to set "BucketName", "RoleName", and "ScriptURL"
        // to the appropriate values. Also set "CrawlerName" to the name
        // you want to give the crawler when it is created.
        string bucketName = _configuration["BucketName"]!;
        string bucketUrl = _configuration["BucketUrl"]!;
        string crawlerName = _configuration["CrawlerName"]!;
        string roleName = _configuration["RoleName"]!;
        string sourceData = _configuration["SourceData"]!;
        string dbName = _configuration["DbName"]!;
    }
}
```

```
string cron = _configuration["Cron"]!;
string scriptUrl = _configuration["ScriptURL"]!;
string jobName = _configuration["JobName"]!;

var wrapper = host.Services.GetRequiredService<GlueWrapper>();
var uiWrapper = host.Services.GetRequiredService<UiWrapper>();

uiWrapper.DisplayOverview();
uiWrapper.PressEnter();

// Create the crawler and wait for it to be ready.
uiWrapper.DisplayTitle("Create AWS Glue crawler");
Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
if (crawlerCreated)
{
    Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
    CrawlerState crawlerState;
    do
    {
        crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
    }
    while (crawlerState != "READY");
    Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
}
else
{
    Console.WriteLine($"Couldn't create crawler {crawlerName}.");
    return; // Exit the application.
}

uiWrapper.DisplayTitle("Start AWS Glue crawler");
Console.WriteLine("Now let's wait until the crawler has successfully
started.");
var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
if (crawlerStarted)
{
    CrawlerState crawlerState;
    do
```

```
        {
            crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
        }
        while (crawlerState != "READY");
        Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
    }
    else
    {
        Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
        return; // Exit the application.
    }

    uiWrapper.PressEnter();

    Console.WriteLine($"\\nLet's take a look at the database: {dbName}");
    var database = await wrapper.GetDatabaseAsync(dbName);

    if (database != null)
    {
        uiWrapper.DisplayTitle($"{database.Name} Details");
        Console.WriteLine($"{database.Name} created on {database.CreateTime}");
        Console.WriteLine(database.Description);
    }

    uiWrapper.PressEnter();

    var tables = await wrapper.GetTablesAsync(dbName);
    if (tables.Count > 0)
    {
        tables.ForEach(table =>
        {
            Console.WriteLine($"{table.Name}\\tCreated:
{table.CreateTime}\\tUpdated: {table.UpdateTime}");
        });
    }

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Create AWS Glue job");
    Console.WriteLine("Creating a new AWS Glue job.");
    var description = "An AWS Glue job created using the AWS SDK for .NET";
    await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
    roleName, description, scriptUrl);
```

```
    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Starting AWS Glue job");
    Console.WriteLine("Starting the new AWS Glue job...");
    var jobId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
    var jobRunComplete = false;
    var jobRun = new JobRun();
    do
    {
        jobRun = await wrapper.GetJobRunAsync(jobName, jobId);
        if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState == "STOPPED"
||
            jobRun.JobRunState == "FAILED" || jobRun.JobRunState == "TIMEOUT")
        {
            jobRunComplete = true;
        }
    } while (!jobRunComplete);

    uiWrapper.DisplayTitle($"Data in {bucketName}");

    // Get the list of data stored in the S3 bucket.
    var s3Client = new AmazonS3Client();

    var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
    response.S3Objects.ForEach(s3Object =>
    {
        Console.WriteLine(s3Object.Key);
    });

    uiWrapper.DisplayTitle("AWS Glue jobs");
    var jobNames = await wrapper.ListJobsAsync();
    jobNames.ForEach(jobName =>
    {
        Console.WriteLine(jobName);
    });

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Get AWS Glue job run information");
    Console.WriteLine("Getting information about the AWS Glue job.");
    var jobRuns = await wrapper.GetJobRunsAsync(jobName);
```

```
        jobRuns.ForEach(jobRun =>
        {
            Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Deleting resources");
        Console.WriteLine("Deleting the AWS Glue job used by the example.");
        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });

        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);

        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}

namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");
    }
}
```



```
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the URL
to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.WriteLine("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</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);
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

IAM-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit IAM verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo IAM

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von IAM beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();

        var listPoliciesPaginator = iamClient.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }
    }
}
```

```
        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
{policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [ListPolicies](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Hinzufügen eines Benutzers zu einer Gruppe

Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer zu einer IAM-Gruppe hinzufügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add an existing IAM user to an existing IAM group.
/// </summary>
/// <param name="userName">The username of the user to add.</param>
/// <param name="groupName">The name of the group to add the user to.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
```

```
{
    var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
    {
        GroupName = groupName,
        UserName = userName,
    });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [AddUserToGroup](#) in der APIAWS SDK for .NET - Referenz für .

Anfügen einer Richtlinie an eine Rolle

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Richtlinie an eine Rolle anfügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Attach an IAM policy to a role.
/// </summary>
/// <param name="policyArn">The policy to attach.</param>
/// <param name="roleName">The role that the policy will be attached to.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });
}
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Weitere API-Informationen finden Sie unter [AttachRolePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Anfügen einer Inline-Richtlinie an eine Rolle

Das folgende Codebeispiel veranschaulicht, wie Sie eine Inline-Richtlinie an eine IAM-Rolle anfügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Update the inline policy document embedded in a role.
/// </summary>
/// <param name="policyName">The name of the policy to embed.</param>
/// <param name="roleName">The name of the role to update.</param>
/// <param name="policyDocument">The policy document that defines the role.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)
{
    var request = new PutRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutRolePolicyAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [PutRolePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Gruppe

Die folgenden Codebeispiele veranschaulichen, wie Sie eine IAM-Gruppe erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an IAM group.
/// </summary>
/// <param name="groupName">The name to give the IAM group.</param>
/// <returns>The IAM group that was created.</returns>
public async Task<Group> CreateGroupAsync(string groupName)
{
    var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
    { GroupName = groupName });
    return response.Group;
}
```

- Weitere API-Informationen finden Sie unter [CreateGroup](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Richtlinie

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Richtlinie erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an IAM policy.
/// </summary>
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
    {
        PolicyDocument = policyDocument,
        PolicyName = policyName,
    });

    return response.Policy;
}
```

- Weitere API-Informationen finden Sie unter [CreatePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Rolle

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Rolle erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = rolePolicyDocument,
    };


    var response = await _IAMService.CreateRoleAsync(request);
    return response.Role.Arn;
}
```

- Weitere API-Informationen finden Sie unter [CreateRole](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen einer serviceverknüpften Rolle

Das folgende Codebeispiel zeigt, wie Sie eine serviceverknüpfte IAM-Rolle erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Create an IAM service-linked role.
/// </summary>
/// <param name="serviceName">The name of the AWS Service.</param>
/// <param name="description">A description of the IAM service-linked role.</
param>
/// <returns>The IAM role that was created.</returns>
public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}
```

- Weitere API-Informationen finden Sie unter [CreateServiceLinkedRole](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Benutzers

Das folgende Codebeispiel veranschaulicht, wie Sie einen IAM-Benutzer erstellen.

 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](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an IAM user.
/// </summary>
/// <param name="userName">The username for the new IAM user.</param>
/// <returns>The IAM user that was created.</returns>
public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ Username = userName });
    return response.User;
}
```

- Weitere API-Informationen finden Sie unter [CreateUser](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Zugriffsschlüssels

Das folgende Codebeispiel veranschaulicht, wie Sie IAM-Zugriffsschlüssel erstellen.

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](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an IAM access key for a user.
/// </summary>
/// <param name="userName">The username for which to create the IAM access
/// key.</param>
/// <returns>The AccessKey.</returns>
public async Task<AccessKey> CreateAccessKeyAsync(string userName)
{
    var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
    {
        UserName = userName,
    });

    return response.AccessKey;
}
```

- Weitere API-Informationen finden Sie unter [CreateAccessKey](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Inline-Richtlinie für eine Gruppe

Das folgende Codebeispiel veranschaulicht, wie Sie eine Inline-IAM-Richtlinie für eine Gruppe erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add or update an inline policy document that is embedded in an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group.</param>
/// <param name="policyName">The name of the IAM policy.</param>
/// <param name="policyDocument">The policy document defining the IAM policy.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
string policyDocument)
{
    var request = new PutGroupPolicyRequest
    {
        GroupName = groupName,
        PolicyName = policyName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutGroupPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [PutGroupPolicy](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines Instance-Profils

Das folgende Codebeispiel zeigt, wie ein IAM-Instance-Profil erstellt wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
    /// specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance. The role has attached policies that specify the AWS permissions
    granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
    role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
            "\"Version\": \"2012-10-17\", " +
            "\"Statement\": [{" +
                "\"Effect\": \"Allow\", " +
                "\"Principal\": {" +
                "\"Service\": [" +
                "\"ec2.amazonaws.com\"" +
                "]" +
                "}, " +
            "\"Action\": \"sts:AssumeRole\"" +
            "}] " +

```

```
        }"}";

var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

var policyArn = "";

try
{
    var createPolicyResult = await _amazonIam.CreatePolicyAsync(
        new CreatePolicyRequest
        {
            PolicyName = policyName,
            PolicyDocument = policyDocument
        });
    policyArn = createPolicyResult.Policy.Arn;
}
catch (EntityAlreadyExistsException)
{
    // The policy already exists, so we look it up to get the Arn.
    var policiesPaginator = _amazonIam.Paginators.ListPolicies(
        new ListPoliciesRequest()
        {
            Scope = PolicyScopeType.Local
        });
    // Get the entire list using the paginator.
    await foreach (var policy in policiesPaginator.Policies)
    {
        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
```

```
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
}
}
```



```
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Policy already exists.");
    var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
        new GetInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    profileArn = profileGetResponse.InstanceProfile.Arn;
}
return profileArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateInstanceProfile](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Gruppe

Das folgende Code-Beispiel veranschaulicht, wie Sie eine IAM-Gruppe löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteGroupAsync(string groupName)
{
    var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
    { GroupName = groupName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteGroup](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Gruppenrichtlinie

Das folgende Code-Beispiel veranschaulicht, wie Sie eine IAM-Gruppenrichtlinie löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM policy associated with an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group associated with the
/// policy.</param>
/// <param name="policyName">The name of the policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
{
    var request = new DeleteGroupPolicyRequest()
    {
        GroupName = groupName,
        PolicyName = policyName,
    };

    var response = await _IAMService.DeleteGroupPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteGroupPolicy](#) in der APIAWS SDK for .NET - Referenz für .

Löschen Sie eine Richtlinie

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Richtlinie löschen.

AWS SDK for .NET

Note

Auf [GitHub](#) gibt es mehr Beispiele. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM policy.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
/// delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeletePolicyAsync(string policyArn)
{
    var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
    { PolicyArn = policyArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeletePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Löschen Sie eine Rolle

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Rolle löschen.

AWS SDK for .NET

Note

Auf [GitHub](#) gibt es mehr Beispiele. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRoleAsync(string roleName)
{
    var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteRole](#) in der APIAWS SDK for .NET -Referenz für .

Löschen Sie eine Rollenrichtlinie

Das folgende Code-Beispiel veranschaulicht, wie Sie eine IAM-Rollenrichtlinie löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM role policy.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="policyName">The name of the IAM role policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
{
    var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
{
```

```
        PolicyName = policyName,  
        RoleName = roleName,  
    });  
  
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
}
```

- Weitere API-Informationen finden Sie unter [DeleteRolePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Benutzers

Das folgende Codebeispiel veranschaulicht, wie Sie einen IAM-Benutzer löschen.

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](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Delete an IAM user.  
/// </summary>  
/// <param name="userName">The username of the IAM user to delete.</param>  
/// <returns>A Boolean value indicating the success of the action.</returns>  
public async Task<bool> DeleteUserAsync(string userName)  
{
```

```
var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteUser](#) in der APIAWS SDK for .NET -Referenz für .

Löschen eines Zugriffsschlüssels

Das folgende Codebeispiel veranschaulicht, wie Sie einen IAM-Zugriffsschlüssel löschen.

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](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM user's access key.
/// </summary>
/// <param name="accessKeyId">The Id for the IAM access key.</param>
/// <param name="userName">The username of the user that owns the IAM
/// access key.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
```


```
{
    var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
    {
        AccessKeyId = accessKeyId,
        UserName = userName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteAccessKey](#) in der APIAWS SDK for .NET - Referenz für .


Löschen Sie eine Inline-Richtlinie von einem Benutzer

Das folgende Codebeispiel zeigt, wie Sie eine Inline-IAM-Richtlinie von einem Benutzer löschen.

 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](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an IAM user policy.
/// </summary>
/// <param name="policyName">The name of the IAM policy to delete.</param>
```

```
/// <param name="userName">The username of the IAM user.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
{
    var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteUserPolicy](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Instance-Profils

Das folgende Codebeispiel zeigt, wie ein IAM-Instance-Profil gelöscht wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
```



```
        InstanceProfileName = profileName,
        RoleName = roleName
    });
    await _amazonIam.DeleteInstanceProfileAsync(
        new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
    var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
        new ListAttachedRolePoliciesRequest() { RoleName = roleName });
    foreach (var policy in attachedPolicies.AttachedPolicies)
    {
        await _amazonIam.DetachRolePolicyAsync(
            new DetachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policy.PolicyArn
            });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}
```

- Weitere API-Informationen finden Sie unter [DeleteInstanceProfile](#) in der APIAWS SDK for .NET -Referenz für .

Trennen Sie eine Richtlinie von einer Rolle

Das folgende Codebeispiel veranschaulicht, wie Sie eine IAM-Richtlinie von einer Rolle trennen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Detach an IAM policy from an IAM role.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
/// <param name="roleName">The name of the IAM role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DetachRolePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen einer Richtlinie

Das folgende Codebeispiel zeigt, wie Sie eine IAM-Richtlinie abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about an IAM policy.
/// </summary>
/// <param name="policyArn">The IAM policy to retrieve information for.</param>
/// <returns>The IAM policy.</returns>
public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
{
    var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
    { PolicyArn = policyArn });
    return response.Policy;
}
```

- Weitere API-Informationen finden Sie unter [GetPolicy](#) in der APIAWS SDK for .NET -Referenz für .

Rufen Sie eine Rolle ab

Das folgende Codebeispiel zeigt, wie Sie eine IAM-Rolle abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about an IAM role.
```

```
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}
```

- Weitere API-Informationen finden Sie unter [GetRole](#) in der APIAWS SDK for .NET -Referenz für

Abrufen eines Benutzers

Das folgenden Codebeispiel veranschaulicht, wie Sie einen IAM-Benutzer abrufen.

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](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Get information about an IAM user.
/// </summary>
/// <param name="userName">The username of the user.</param>
/// <returns>An IAM user object.</returns>
public async Task<User> GetUserAsync(string userName)
{
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
    return response.User;
}
```

- Weitere API-Informationen finden Sie unter [GetUser](#) in der APIAWS SDK for .NET -Referenz für

Rufen Sie die Passwort-Richtlinie ab

Das folgende Codebeispiel zeigt, wie Sie die Passworrichtlinie für das IAM-Konto abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Gets the IAM password policy for an AWS account.
/// </summary>
/// <returns>The PasswordPolicy for the AWS account.</returns>
public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
{
    var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
    return response.PasswordPolicy;
}
```

- Weitere API-Informationen finden Sie unter [GetAccountPasswordPolicy](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von SAML-Anbietern

Das folgende Codebeispiel zeigt, wie Sie SAML-Anbieter für IAM auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List SAML authentication providers.
/// </summary>
/// <returns>A list of SAML providers.</returns>
public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
{
    var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
    return response.SAMLProviderList;
}
```

- Weitere API-Informationen finden Sie unter [ListSAMLProviders](#) in der API-Referenz für AWS SDK for .NET .

Auflisten von Gruppen

Das folgende Codebeispiel zeigt, wie Sie IAM-Gruppen auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
    var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
    var groups = new List<Group>();

    await foreach (var response in groupsPaginator.Responses)
    {
        groups.AddRange(response.Groups);
    }

    return groups;
}
```

- Weitere API-Informationen finden Sie unter [ListGroups](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Inline-Richtlinien für eine Rolle

Das folgende Codebeispiel zeigt, wie Sie Inline-Richtlinien für eine IAM-Rolle auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List IAM role policies.
/// </summary>
/// <param name="roleName">The IAM role for which to list IAM policies.</param>
/// <returns>A list of IAM policy names.</returns>
public async Task<List<string>> ListRolePoliciesAsync(string roleName)
{
    var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
    var policyNames = new List<string>();

    await foreach (var response in listRolePoliciesPaginator.Responses)
    {
        policyNames.AddRange(response.PolicyNames);
    }

    return policyNames;
}
```

- Weitere API-Informationen finden Sie unter [ListRolePolicies](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Richtlinien

Das folgende Codebeispiel zeigt, wie Sie IAM-Richtlinien auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List IAM policies.
/// </summary>
/// <returns>A list of the IAM policies.</returns>
public async Task<List<ManagedPolicy>> ListPoliciesAsync()
{
    var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
    var policies = new List<ManagedPolicy>();

    await foreach (var response in listPoliciesPaginator.Responses)
    {
        policies.AddRange(response.Policies);
    }

    return policies;
}
```

- Weitere API-Informationen finden Sie unter [ListPolicies](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Richtlinien, die an eine Rolle angefügt sind

Das folgende Codebeispiel zeigt, wie Sie Richtlinien auflisten, die einer IAM-Rolle zugeordnet sind.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}
```

- Weitere API-Informationen finden Sie unter [ListAttachedRolePolicies](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Rollen

Das folgende Codebeispiel zeigt, wie Sie IAM-Rollen auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List IAM roles.
/// </summary>
/// <returns>A list of IAM roles.</returns>
public async Task<List<Role>> ListRolesAsync()
{
    var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
    var roles = new List<Role>();


    await foreach (var response in listRolesPaginator.Responses)
    {
        roles.AddRange(response.Roles);
    }

    return roles;
}
```

- Weitere API-Informationen finden Sie unter [ListRoles](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Benutzern

Das folgende Codebeispiel veranschaulicht, wie Sie IAM-Benutzer auflisten.

 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](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List IAM users.
/// </summary>
/// <returns>A list of IAM users.</returns>
public async Task<List<User>> ListUsersAsync()
{
    var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
    var users = new List<User>();

    await foreach (var response in listUsersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- Weitere API-Informationen finden Sie unter [ListUsers](#) in der APIAWS SDK for .NET -Referenz für .

Entfernen eines Benutzers aus einer Gruppe

Das folgenden Codebeispiel veranschaulichen, wie Sie einen Benutzer aus einer IAM-Gruppe entfernen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Remove a user from an IAM group.
/// </summary>
/// <param name="userName">The username of the user to remove.</param>
/// <param name="groupName">The name of the IAM group to remove the user from.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
{
    // Remove the user from the group.
    var removeUserRequest = new RemoveUserFromGroupRequest()
    {
        UserName = userName,
        GroupName = groupName,
    };

    var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [RemoveUserFromGroup](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie AWS Systems Manager die Parameter aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();
}
```

```
// Set up dependency injection for the AWS services.
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<IAmazonIdentityManagementService>()
            .AddAWSService<IAmazonDynamoDB>()
            .AddAWSService<IAmazonElasticLoadBalancingV2>()
            .AddAWSService<IAmazonSimpleSystemsManagement>()
            .AddAWSService<IAmazonAutoScaling>()
            .AddAWSService<IAmazonEC2>()
            .AddTransient<AutoScalerWrapper>()
            .AddTransient<ElasticLoadBalancerWrapper>()
            .AddTransient<SmParameterWrapper>()
            .AddTransient<Recommendations>()
            .AddSingleton<IConfiguration>(_configuration)
        )
    .Build();

ServicesSetup(host);
ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await DestroyResources(true);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _recommendations = host.Services.GetRequiredService<Recommendations>();
    _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
    _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
}

/// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
```



```
/// <returns>True if successful.</returns>
public static async Task<bool> Deploy(bool interactive)
{
    var protocol = "HTTP";
    var port = 80;
    var sshPort = 22;

    Console.WriteLine(
        "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
        "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
        "against various kinds of failures.\n\n" +
        "Some of the resources create by this demo are:\n");

    Console.WriteLine(
        "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
    Console.WriteLine(
        "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
    Console.WriteLine(
        "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
    Console.WriteLine(
        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
    if (interactive)
        Console.ReadLine();

    // Create and populate the DynamoDB table.
    var databaseTableName = _configuration["databaseName"];
    var recommendationsPath = Path.Join(_configuration["resourcePath"],
        "recommendations_objects.json");
    Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
    await _recommendations.CreateDatabaseWithName(databaseTableName);
    await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
    Console.WriteLine(new string('-', 80));
}
```

```
// Create the EC2 Launch Template.

Console.WriteLine(
    $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
    + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
    + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
    + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
    + "run a web server, such as Apache, with least-privileged
credentials.");
Console.WriteLine(
    "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
    + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
    + "that control the flow of the demo.");

var startupScriptPath = Path.Join(_configuration["resourcePath"],
    "server_startup_script.sh");
var instancePolicyPath = Path.Join(_configuration["resourcePath"],
    "instance_policy.json");
await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
    + "Availability Zone.\n");
var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
Console.WriteLine(new string('-', 80));

Console.WriteLine(
    "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
    + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

Console.WriteLine(new string('-', 80));
```

```
Console.WriteLine("Press Enter when you're ready to continue.");
if (interactive)
    Console.ReadLine();

Console.WriteLine("Creating variables that control the flow of the demo.");
await _smParameterWrapper.Reset();

Console.WriteLine(
    "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
    + "defines how the load balancer connects to instances. The load
balancer provides a\n"
    + "single endpoint where clients connect and dispatches requests to
instances in the group.");

var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
var subnetIds = subnets.Select(s => s.SubnetId).ToList();
var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
subnetIds, targetGroup);
await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
Console.WriteLine("\nVerifying access to the load balancer endpoint...");
var endPoint = await
_elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
var loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

if (!loadBalancerAccess)
{
    Console.WriteLine("\nCouldn't connect to the load balancer, verifying
that the port is open...");

    var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
    ipString = ipString.Trim();
```

```
        var defaultSecurityGroup = await
_autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
        var portIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
        var sshPortIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
ipString);

        if (!portIsOpen)
        {
            Console.WriteLine(
                "\nFor this example to work, the default security group for your
default VPC must\n"
                + "allows access from this computer. You can either add it
automatically from this\n"
                + "example or add it yourself using the AWS Management Console.
\n");

            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
            }
        }

        if (!sshPortIsOpen)
        {
            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
            }
        }
        loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
```

```

        Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
        Console.WriteLine($"\\thttp://{endPoint}\\n");
    }
    else
    {
        Console.WriteLine(
            "\\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\\n"
            + "manually verifying that your VPC and security group are
configured correctly and that\\n"
            + "you can successfully make a GET request to the load balancer
endpoint:\\n");
        Console.WriteLine($"\\thttp://{endPoint}\\n");
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
    if (interactive)
        Console.ReadLine();
    return true;
}

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\\nThis part of the demonstration shows how to toggle
different parts of the system\\n" +
        "to create situations where the web service fails, and
shows how using a resilient\\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
}

```

```
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
    Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
    Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("Let's reinstate the recommendation service.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
    Console.WriteLine(
        "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
```

```
        "access the DynamoDB recommendation table.\n"
    );
    await _autoScalerWrapper.CreateInstanceProfileWithName(
        _autoScalerWrapper.BadCredsPolicyName,
        _autoScalerWrapper.BadCredsRoleName,
        _autoScalerWrapper.BadCredsProfileName,
        ssmOnlyPolicy,
        new List<string> { "AmazonSSMManagedInstanceCore" }
    );
    var instances = await
    _autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
    var badInstanceId = instances.First();
    var instanceProfile = await
    _autoScalerWrapper.GetInstanceProfile(badInstanceId);
    Console.WriteLine(
        $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
        "bad credentials...\n"
    );
    await _autoScalerWrapper.ReplaceInstanceProfile(
        badInstanceId,
        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");
```

```
        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($" \nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($" \nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");
```



```
        await
        _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
        }
    }
}
```

```

        await
        _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
        _recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Erstellen Sie eine Klasse, die Auto-Scaling- und Amazon-EC2-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
}

```

```
private readonly string _badCredsRoleName = "";
private readonly string _badCredsPolicyName = "";
private readonly string _keyPairName = "";

public string GroupName => _groupName;
public string KeyPairName => _keyPairName;
public string LaunchTemplateName => _launchTemplateName;
public string InstancePolicyName => _instancePolicyName;
public string BadCredsProfileName => _badCredsProfileName;
public string BadCredsRoleName => _badCredsRoleName;
public string BadCredsPolicyName => _badCredsPolicyName;

/// <summary>
/// Constructor for the AutoScalerWrapper.
/// </summary>
/// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
/// <param name="amazonEc2">The injected EC2 client.</param>
/// <param name="amazonIam">The injected IAM client.</param>
/// <param name="amazonSsm">The injected SSM client.</param>
public AutoScalerWrapper(
    IAmazonAutoScaling amazonAutoScaling,
    IAmazonEC2 amazonEc2,
    IAmazonSimpleSystemsManagement amazonSsm,
    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}
```

```

}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,
    string ssmOnlyPolicyFile,
    List<string>? awsManagedPolicies = null)
{
    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\"," +
            "\"Principal\": {" +
            "\"Service\": [" +
                "\"ec2.amazonaws.com\"" +
            "]" +
            "}," +
            "\"Action\": \"sts:AssumeRole\"" +
        "}]}" +
        "};

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {

```

```
var createPolicyResult = await _amazonIam.CreatePolicyAsync(
    new CreatePolicyRequest
    {
        PolicyName = policyName,
        PolicyDocument = policyDocument
    });
policyArn = createPolicyResult.Policy.Arn;
}
catch (EntityAlreadyExistsException)
{
    // The policy already exists, so we look it up to get the Arn.
    var policiesPaginator = _amazonIam.Paginators.ListPolicies(
        new ListPoliciesRequest()
        {
            Scope = PolicyScopeType.Local
        });
    // Get the entire list using the paginator.
    await foreach (var policy in policiesPaginator.Policies)
    {
        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
```

```
        {
            foreach (var awsPolicy in awsManagedPolicies)
            {
                await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                {
                    PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                    RoleName = roleName
                });
            }
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Policy already exists.");
    var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
        new GetInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
}
```

```
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyPairName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyPairName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
        File.Delete($"{deleteKeyPairName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
    }
}
```

```
    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            }
        );
    }
}
```



```
    });
    return launchTemplateResponse.LaunchTemplate;
}

/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
    return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
}

/// <summary>
/// Create an EC2 Auto Scaling group of a specified size and name.
/// </summary>
/// <param name="groupSize">The size for the group.</param>
/// <param name="groupName">The name for the group.</param>
/// <param name="availabilityZones">The availability zones for the group.</
param>
/// <returns>Async task.</returns>
public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
{
    try
    {
        await _amazonAutoScaling.CreateAutoScalingGroupAsync(
            new CreateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                AvailabilityZones = availabilityZones,
                LaunchTemplate =
                    new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                    {
                        LaunchTemplateName = _launchTemplateName,
                        Version = "$Default"
                    },
                MaxSize = groupSize,
                MinSize = groupSize
            });
    }
}
```

```
        Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
    }
}

/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
    return vpcResponse.Vpcs[0];
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    var subnets = new List<Subnet>();
    var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
        new DescribeSubnetsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("vpc-id", new List<string>() { vpcId}),
                new ("availability-zone", availabilityZones),
            }
        });
    while (subnetPaginator.HasNext())
    {
        subnets.AddRange(subnetPaginator.CurrentPage);
    }
}
```

```
        new ("default-for-az", new List<string>() { "true" })
    }
});

// Get the entire list using the paginator.
await foreach (var subnet in subnetPaginator.Subnets)
{
    subnets.Add(subnet);
}

return subnets;
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
```

```
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                });
            // Delete the custom policies only.
            if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
            {
                await _amazonIam.DeletePolicyAsync(
                    new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                    {
                        PolicyArn = policy.PolicyArn
                    });
            }
        }

        await _amazonIam.DeleteRoleAsync(
            new DeleteRoleRequest() { RoleName = roleName });
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine($"Instance profile {profileName} does not exist.");
    }
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
```

```
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
    new DescribeAutoScalingGroupsRequest()
    {
        AutoScalingGroupNames = new List<string>() { group }
    });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
        new DescribeIamInstanceProfileAssociationsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("instance-id", new List<string>() { instanceId })
            },
        });
    return response.IamInstanceProfileAssociations[0];
}

/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
```

```
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
            new ReplaceIamInstanceProfileAssociationRequest()
            {
                AssociationId = associationId,
                IamInstanceProfile = new IamInstanceProfileSpecification()
                {
                    Name = credsProfileName
                }
            });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
            _amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
            }
        );
    }
}
```

```
        DocumentName = "AWS-RunShellScript",
        Parameters = new Dictionary<string, List<string>>()
        {
            {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
        }
    });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                });
            stopping = true;
        }
        catch (ScalingActivityInProgressException)
        {
            Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
```

```
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                });
            stopped = true;
        }
        catch (Exception e)
            when ((e is ScalingActivityInProgressException)
                || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
```



```
        new UpdateAutoScalingGroupRequest()
        {
            AutoScalingGroupName = groupName,
            MinSize = 0
        });
    var group = describeGroupsResponse.AutoScalingGroups[0];
    foreach (var instance in group.Instances)
    {
        await TryTerminateInstanceById(instance.InstanceId);
    }

    await TryDeleteGroupByName(groupName);
}
else
{
    Console.WriteLine($"No groups found with name {groupName}.");
}
}

/// <summary>
/// Get the default security group for a specified Vpc.
/// </summary>
/// <param name="vpc">The Vpc to search.</param>
/// <returns>The default security group.</returns>
public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
{
    var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
        new DescribeSecurityGroupsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("group-name", new List<string>() { "default" }),
                new ("vpc-id", new List<string>() { vpc.VpcId })
            }
        });
    return groupResponse.SecurityGroups[0];
}

/// <summary>
/// Verify the default security group of a Vpc allows ingress from the calling
computer.
/// This can be done by allowing ingress from this computer's IP address.
```

```
    /// In some situations, such as connecting from a corporate network, you must
    /// instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    /// while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }

                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }
    }
}
```

```
    }

    return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()
            {
                new IpPermission()
                {
                    FromPort = port,
                    ToPort = port,
                    IpProtocol = "tcp",
                    Ipv4Ranges = new List<IpRange>()
                    {
                        new IpRange() { CidrIp = $"{ipAddress}/32" }
                    }
                }
            }
        });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
```

```

    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}

```

Erstellen Sie eine Klasse, die Elastic-Load-Balancing-Aktionen beinhaltet.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
    }
}

```

```
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```
public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
{
    List<TargetHealthDescription> result = null!;
    try
    {
        var groupResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                new DescribeTargetGroupsRequest()
                {
                    Names = new List<string>() { groupName }
                });
        var healthResponse =
            await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                new DescribeTargetHealthRequest()
                {
                    TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                });
        ;
        result = healthResponse.TargetHealthDescriptions;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine($"Target group {groupName} not found.");
    }
    return result;
}

/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
```

```
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
        ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
    subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
    }
}
```

```
var loadBalancerReady = false;
while (!loadBalancerReady)
{
    try
    {
        var describeResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });

        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
return createLbResponse.LoadBalancers[0];
}

///  
/// <summary>
```



```
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }

    return success;
}

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
```

```
try
{
    var describeLoadBalancerResponse =
        await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
            new DescribeLoadBalancersRequest()
            {
                Names = new List<string>() { name }
            }
        );
    var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
    await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
        new DeleteLoadBalancerRequest()
        {
            LoadBalancerArn = lbArn
        }
    );
}
catch (LoadBalancerNotFoundException)
{
    Console.WriteLine($"Load balancer {name} not found.");
}
}

/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    }
                );

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
```

```

        new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
        Console.WriteLine($"Deleted load balancing target group
{groupName}.");
        done = true;
    }
    catch (TargetGroupNotFoundException)
    {
        Console.WriteLine(
            $"Target group {groupName} not found, could not delete.");
        done = true;
    }
    catch (ResourceInUseException)
    {
        Console.WriteLine("Target group not yet released, waiting...");
        Thread.Sleep(10000);
    }
    }
}
}

```

Erstellen Sie eine Klasse, die DynamoDB zum Simulieren eines Empfehlungsservices verwendet.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {

```

```
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()
                {
                    AttributeName = "MediaType",
                    KeyType = KeyType.HASH
                },
                new KeySchemaElement()
                {
                    AttributeName = "ItemId",
                    KeyType = KeyType.RANGE
                }
            },
        },
    }
}
```

```
        ProvisionedThroughput = new ProvisionedThroughput()
        {
            ReadCapacityUnits = 5,
            WriteCapacityUnits = 5
        }
    };
    await _amazonDynamoDb.CreateTableAsync(createRequest);

    // Wait until the table is ACTIVE and then report success.
    Console.WriteLine("\nWaiting for table to become active...");

    var request = new DescribeTableRequest
    {
        TableName = tableName
    };

    TableStatus status;
    do
    {
        Thread.Sleep(2000);

        var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
        status = describeTableResponse.Table.TableStatus;

        Console.WriteLine(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
    return false;
}
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
```

```
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}
}
```

Erstellen Sie eine Klasse, die Systems-Manager-Aktionen umschließt.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
parameters
```

```
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }
}
```

```
/// <summary>
/// Set the value of a named Systems Manager parameter.
/// </summary>
/// <param name="name">The name of the parameter.</param>
/// <param name="value">The value to set.</param>
/// <returns>Async task.</returns>
public async Task PutParameterByName(string name, string value)
{
    await _amazonSimpleSystemsManagement.PutParameterAsync(
        new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)

- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Erstellen einer Benutzergruppe und Hinzufügen eines Benutzers

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Gruppe und gewähren Sie ihr vollständige Amazon-S3-Zugriffsberechtigungen.
- Erstellen Sie einen neuen Benutzer ohne Berechtigungen für den Zugriff auf Amazon S3.
- Fügen Sie den Benutzer der Gruppe hinzu und zeigen Sie, dass er jetzt über Berechtigungen für Amazon S3 verfügt. Bereinigen Sie dann die Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
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;
```

```
namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
```

```
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;
    }

    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }

    /// <summary>
    /// Create an IAM policy.
    /// </summary>
```

```
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
    {
        PolicyDocument = policyDocument,
        PolicyName = policyName,
    });

    return response.Policy;
}

/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = rolePolicyDocument,
    };

    var response = await _IAMService.CreateRoleAsync(request);
    return response.Role.Arn;
}

/// <summary>
/// Create an IAM service-linked role.
/// </summary>
/// <param name="serviceName">The name of the AWS Service.</param>
/// <param name="description">A description of the IAM service-linked role.</
param>
/// <returns>The IAM role that was created.</returns>
```

```
public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}

/// <summary>
/// Create an IAM user.
/// </summary>
/// <param name="userName">The username for the new IAM user.</param>
/// <returns>The IAM user that was created.</returns>
public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
    return response.User;
}

/// <summary>
/// Delete an IAM user's access key.
/// </summary>
/// <param name="accessKeyId">The Id for the IAM access key.</param>
/// <param name="userName">The username of the user that owns the IAM
/// access key.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
{
    var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
{
    AccessKeyId = accessKeyId,
    UserName = userName,
});
}
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
    { GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
    policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
public async Task<bool> DeletePolicyAsync(string policyArn)
{
    var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRoleAsync(string roleName)
{
    var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM role policy.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="policyName">The name of the IAM role policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
{
    var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM user.
/// </summary>
```

```
/// <param name="userName">The username of the IAM user to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserAsync(string userName)
{
    var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM user policy.
/// </summary>
/// <param name="policyName">The name of the IAM policy to delete.</param>
/// <param name="userName">The username of the IAM user.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
{
    var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Detach an IAM policy from an IAM role.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
/// <param name="roleName">The name of the IAM role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```



```
}

/// <summary>
/// Gets the IAM password policy for an AWS account.
/// </summary>
/// <returns>The PasswordPolicy for the AWS account.</returns>
public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
{
    var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
    return response.PasswordPolicy;
}

/// <summary>
/// Get information about an IAM policy.
/// </summary>
/// <param name="policyArn">The IAM policy to retrieve information for.</param>
/// <returns>The IAM policy.</returns>
public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
{
    var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
    return response.Policy;
}

/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}
```

```
/// <summary>
/// Get information about an IAM user.
/// </summary>
/// <param name="userName">The username of the user.</param>
/// <returns>An IAM user object.</returns>
public async Task<User> GetUserAsync(string userName)
{
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
    return response.User;
}

/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}

/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
```

```
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }

    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }

    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
```

```
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }

    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }

    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
```

```
{
    var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
    var users = new List<User>();

    await foreach (var response in listUsersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}

/// <summary>
/// Remove a user from an IAM group.
/// </summary>
/// <param name="userName">The username of the user to remove.</param>
/// <param name="groupName">The name of the IAM group to remove the user from.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
{
    // Remove the user from the group.
    var removeUserRequest = new RemoveUserFromGroupRequest()
    {
        UserName = userName,
        GroupName = groupName,
    };

    var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Add or update an inline policy document that is embedded in an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group.</param>
/// <param name="policyName">The name of the IAM policy.</param>
/// <param name="policyDocument">The policy document defining the IAM policy.</
param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
```

```
    /// <param name="policyDocument">The policy document defining the IAM policy.</  
param>  
    /// <returns>A Boolean value indicating the success of the action.</returns>  
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,  
string policyDocument)  
    {  
        var request = new PutUserPolicyRequest  
        {  
            UserName = userName,  
            PolicyName = policyName,  
            PolicyDocument = policyDocument  
        };  
  
        var response = await _IAMService.PutUserPolicyAsync(request);  
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
    }  
  
    /// <summary>  
    /// Wait for a new access key to be ready to use.  
    /// </summary>  
    /// <param name="accessKeyId">The Id of the access key.</param>  
    /// <returns>A boolean value indicating the success of the action.</returns>  
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)  
    {  
        var keyReady = false;  
  
        do  
        {  
            try  
            {  
                var response = await _IAMService.GetAccessKeyLastUsedAsync(  
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });  
                if (response.UserName is not null)  
                {  
                    keyReady = true;  
                }  
            }  
            catch (NoSuchEntityException)  
            {  
                keyReady = false;  
            }  
        } while (!keyReady);  
  
        return keyReady;  
    }  
}
```

```
    }  
}  
  
using Microsoft.Extensions.Configuration;  
  
namespace IAMGroups;  
  
public class IAMGroups  
{  
    private static ILogger logger = null!;  
  
    // Represents JSON code for AWS full access policy for Amazon Simple  
    // Storage Service (Amazon S3).  
    private const string S3FullAccessPolicyDocument = "{" +  
        " \"Statement\" : [{" +  
            " \"Action\" : [\"s3:*\"],\" +  
            " \"Effect\" : \"Allow\",\" +  
            " \"Resource\" : \"*\"\" +  
        "}]\" +  
        "}\"";  
  
    static async Task Main(string[] args)  
    {  
        // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>()  
                    .AddTransient<IAMWrapper>()  
                    .AddTransient<UIWrapper>()  
                )  
            .Build();  
  
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })  
            .CreateLogger<IAMGroups>();  
  
        IConfiguration 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 groupUserName = configuration["GroupUserName"];
var groupName = configuration["GroupName"];
var groupPolicyName = configuration["GroupPolicyName"];
var groupBucketName = configuration["GroupBucketName"];

var wrapper = host.Services.GetRequiredService<IAMWrapper>();
var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

uiWrapper.DisplayGroupsOverview();
uiWrapper.PressEnter();

// Create an IAM group.
uiWrapper.DisplayTitle("Create IAM group");
Console.WriteLine("Let's begin by creating a new IAM group.");
var group = await wrapper.CreateGroupAsync(groupName);

// Add an inline IAM policy to the group.
uiWrapper.DisplayTitle("Add policy to group");
Console.WriteLine("Add an inline policy to the group that allows members to
have full access to");
Console.WriteLine("Amazon Simple Storage Service (Amazon S3) buckets.");

await wrapper.PutGroupPolicyAsync(group.GroupName, groupPolicyName,
S3FullAccessPolicyDocument);

uiWrapper.PressEnter();

// Now create a new user.
uiWrapper.DisplayTitle("Create an IAM user");
Console.WriteLine("Now let's create a new IAM user.");
var groupUser = await wrapper.CreateUserAsync(groupUserName);

// Add the new user to the group.
uiWrapper.DisplayTitle("Add the user to the group");
Console.WriteLine("Adding the user to the group, which will give the user
the same permissions as the group.");
await wrapper.AddUserToGroupAsync(groupUser.UserName, group.GroupName);
```

```
    Console.WriteLine($"User, {groupUser.UserName}, has been added to the group,
{group.GroupName}.");
    uiWrapper.PressEnter();

    Console.WriteLine("Now that we have created a user, and added the user to
the group, let's create an IAM access key.");

    // Create access and secret keys for the user.
    var accessKey = await wrapper.CreateAccessKeyAsync(groupUserName);
    Console.WriteLine("Key created.");
    uiWrapper.WaitABit(15, "Waiting for the access key to be ready for use.");

    uiWrapper.DisplayTitle("List buckets");
    Console.WriteLine("To prove that the user has access to Amazon S3, list the
S3 buckets for the account.");

    var s3Client = new AmazonS3Client(accessKey.AccessKeyId,
accessKey.SecretAccessKey);
    var stsClient = new AmazonSecurityTokenServiceClient(accessKey.AccessKeyId,
accessKey.SecretAccessKey);

    var s3Wrapper = new S3Wrapper(s3Client, stsClient);

    var buckets = await s3Wrapper.ListMyBucketsAsync();

    if (buckets is not null)
    {
        buckets.ForEach(bucket =>
        {
            Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
        });
    }

    // Show that the user also has write access to Amazon S3 by creating
    // a new bucket.
    uiWrapper.DisplayTitle("Create a bucket");
    Console.WriteLine("Since group members have full access to Amazon S3, let's
create a bucket.");
    var success = await s3Wrapper.PutBucketAsync(groupBucketName);

    if (success)
    {
```

```
        Console.WriteLine($"Successfully created the bucket:
{groupBucketName}.");
    }

    uiWrapper.PressEnter();

    Console.WriteLine("Let's list the user's S3 buckets again to show the new
bucket.");

    buckets = await s3Wrapper.ListMyBucketsAsync();

    if (buckets is not null)
    {
        buckets.ForEach(bucket =>
        {
            Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
        });
    }

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Clean up resources");
    Console.WriteLine("First delete the bucket we created.");
    await s3Wrapper.DeleteBucketAsync(groupBucketName);

    Console.WriteLine($"Now remove the user, {groupUserName}, from the group,
{groupName}.");
    await wrapper.RemoveUserFromGroupAsync(groupUserName, groupName);

    Console.WriteLine("Delete the user's access key.");
    await wrapper.DeleteAccessKeyAsync(accessKey.AccessKeyId, groupUserName);

    // Now we can safely delete the user.
    Console.WriteLine("Now we can delete the user.");
    await wrapper.DeleteUserAsync(groupUserName);

    uiWrapper.PressEnter();

    Console.WriteLine("Now we will delete the IAM policy attached to the
group.");
    await wrapper.DeleteGroupPolicyAsync(groupName, groupPolicyName);

    Console.WriteLine("Now we delete the IAM group.");
```

```
        await wrapper.DeleteGroupAsync(groupName);

        uiWrapper.PressEnter();

        Console.WriteLine("The IAM groups demo has completed.");

        uiWrapper.PressEnter();
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
}
```

```
public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
{
    // Create the request to use with the AssumeRoleAsync call.
    var request = new AssumeRoleRequest()
    {
        RoleSessionName = roleSession,
        RoleArn = roleToAssume,
    };

    var response = await _stsService.AssumeRoleAsync(request);

    return response.Credentials;
}

/// <summary>
/// Delete an S3 bucket.
/// </summary>
/// <param name="bucketName">Name of the S3 bucket to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
    return result.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the buckets that are owned by the user's account.
/// </summary>
/// <returns>Async Task.</returns>
public async Task<List<S3Bucket?>> ListMyBucketsAsync()
{
    try
    {
        // Get the list of buckets accessible by the new user.
        var response = await _s3Service.ListBucketsAsync();

        return response.Buckets;
    }
    catch (AmazonS3Exception ex)
    {
        // Something else went wrong. Display the error message.
    }
}
```

```
        Console.WriteLine($"Error: {ex.Message}");
        return null;
    }
}

/// <summary>
/// Create a new S3 bucket.
/// </summary>
/// <param name="bucketName">The name for the new bucket.</param>
/// <returns>A Boolean value indicating whether the action completed
/// successfully.</returns>
public async Task<bool> PutBucketAsync(string bucketName)
{
    var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Update the client objects with new client objects. This is available
/// because the scenario uses the methods of this class without and then
/// with the proper permissions to list S3 buckets.
/// </summary>
/// <param name="s3Service">The Amazon S3 client object.</param>
/// <param name="stsService">The AWS STS client object.</param>
public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
{
    _s3Service = s3Service;
    _stsService = stsService;
}
}

namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
```

```
{
    Console.Clear();

    DisplayTitle("Welcome to the IAM Groups Demo");
    Console.WriteLine("This example application does the following:");
    Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
    Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
access to Amazon S3.");
    Console.WriteLine("\t3. Creates a new IAM user.");
    Console.WriteLine("\t4. Creates an IAM access key for the user.");
    Console.WriteLine("\t5. Adds the user to the IAM group.");
    Console.WriteLine("\t6. Lists the buckets on the account.");
    Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
    Console.WriteLine("\t8. List the buckets again to show the new bucket.");
    Console.WriteLine("\t9. Cleans up all the resources created.");
}

/// <summary>
/// Show information about the IAM Basics scenario.
/// </summary>
public void DisplayBasicsOverview()
{
    Console.Clear();

    DisplayTitle("Welcome to IAM Basics");
    Console.WriteLine("This example application does the following:");
    Console.WriteLine("\t1. Creates a user with no permissions.");
    Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
    Console.WriteLine("\t3. Grants the user permission to assume the role.");
    Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
    Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
    Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
    Console.WriteLine("\t7. Deletes all the resources.");
}

/// <summary>
/// Display a message and wait until the user presses enter.
/// </summary>
public void PressEnter()
```

```
{
    Console.WriteLine("\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();  
    }  
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AddUserToGroup](#)
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreateGroup](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeleteGroup](#)
 - [DeleteGroupPolicy](#)
 - [DeleteUser](#)
 - [PutGroupPolicy](#)
 - [RemoveUserFromGroup](#)

Erstellen Sie einen Benutzer und nehmen Sie eine Rolle an

Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer erstellen und eine Rolle annehmen lassen.

 Warning

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

- Erstellen Sie einen Benutzer ohne Berechtigungen.

- Erstellen einer Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets für das Konto erteilt.
- Hinzufügen einer Richtlinie, damit der Benutzer die Rolle übernehmen kann.
- Übernehmen Sie die Rolle und listen Sie S3-Buckets mit temporären Anmeldeinformationen auf, und bereinigen Sie dann die Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
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;

namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }
}
```

```
/// <summary>
/// Add an existing IAM user to an existing IAM group.
/// </summary>
/// <param name="userName">The username of the user to add.</param>
/// <param name="groupName">The name of the group to add the user to.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
{
    var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
    {
        GroupName = groupName,
        UserName = userName,
    });

    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Attach an IAM policy to a role.
/// </summary>
/// <param name="policyArn">The policy to attach.</param>
/// <param name="roleName">The role that the policy will be attached to.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Create an IAM access key for a user.
/// </summary>
/// <param name="userName">The username for which to create the IAM access
/// key.</param>
```

```
/// <returns>The AccessKey.</returns>
public async Task<AccessKey> CreateAccessKeyAsync(string userName)
{
    var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
    {
        UserName = userName,
    });

    return response.AccessKey;
}

/// <summary>
/// Create an IAM group.
/// </summary>
/// <param name="groupName">The name to give the IAM group.</param>
/// <returns>The IAM group that was created.</returns>
public async Task<Group> CreateGroupAsync(string groupName)
{
    var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
    return response.Group;
}

/// <summary>
/// Create an IAM policy.
/// </summary>
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
    {
        PolicyDocument = policyDocument,
        PolicyName = policyName,
    });

    return response.Policy;
}
```

```
/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = rolePolicyDocument,
    };

    var response = await _IAMService.CreateRoleAsync(request);
    return response.Role.Arn;
}

/// <summary>
/// Create an IAM service-linked role.
/// </summary>
/// <param name="serviceName">The name of the AWS Service.</param>
/// <param name="description">A description of the IAM service-linked role.</
param>
/// <returns>The IAM role that was created.</returns>
public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}
```

```
/// <summary>
/// Create an IAM user.
/// </summary>
/// <param name="userName">The username for the new IAM user.</param>
/// <returns>The IAM user that was created.</returns>
public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
    return response.User;
}

/// <summary>
/// Delete an IAM user's access key.
/// </summary>
/// <param name="accessKeyId">The Id for the IAM access key.</param>
/// <param name="userName">The username of the user that owns the IAM
/// access key.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
{
    var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
    {
        AccessKeyId = accessKeyId,
        UserName = userName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteGroupAsync(string groupName)
{
    var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

```
}

/// <summary>
/// Delete an IAM policy associated with an IAM group.
/// </summary>
/// <param name="groupName">The name of the IAM group associated with the
/// policy.</param>
/// <param name="policyName">The name of the policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
{
    var request = new DeleteGroupPolicyRequest()
    {
        GroupName = groupName,
        PolicyName = policyName,
    };

    var response = await _IAMService.DeleteGroupPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM policy.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
/// delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeletePolicyAsync(string policyArn)
{
    var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRoleAsync(string roleName)
```

```
{
    var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM role policy.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="policyName">The name of the IAM role policy to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
{
    var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM user.
/// </summary>
/// <param name="userName">The username of the IAM user to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserAsync(string userName)
{
    var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete an IAM user policy.
/// </summary>
```



```
/// <param name="policyName">The name of the IAM policy to delete.</param>
/// <param name="userName">The username of the IAM user.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
{
    var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Detach an IAM policy from an IAM role.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
/// <param name="roleName">The name of the IAM role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Gets the IAM password policy for an AWS account.
/// </summary>
/// <returns>The PasswordPolicy for the AWS account.</returns>
public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
{
    var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
    return response.PasswordPolicy;
}
```

```
/// <summary>
/// Get information about an IAM policy.
/// </summary>
/// <param name="policyArn">The IAM policy to retrieve information for.</param>
/// <returns>The IAM policy.</returns>
public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
{
    var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
    return response.Policy;
}

/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
{
    RoleName = roleName,
});
    return response.Role;
}

/// <summary>
/// Get information about an IAM user.
/// </summary>
/// <param name="userName">The username of the user.</param>
/// <returns>An IAM user object.</returns>
public async Task<User> GetUserAsync(string userName)
{
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
    return response.User;
}
```

```
/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}

/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
    var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
    var groups = new List<Group>();

    await foreach (var response in groupsPaginator.Responses)
    {
        groups.AddRange(response.Groups);
    }

    return groups;
}

/// <summary>
/// List IAM policies.
```

```
/// </summary>
/// <returns>A list of the IAM policies.</returns>
public async Task<List<ManagedPolicy>> ListPoliciesAsync()
{
    var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
    var policies = new List<ManagedPolicy>();

    await foreach (var response in listPoliciesPaginator.Responses)
    {
        policies.AddRange(response.Policies);
    }

    return policies;
}

/// <summary>
/// List IAM role policies.
/// </summary>
/// <param name="roleName">The IAM role for which to list IAM policies.</param>
/// <returns>A list of IAM policy names.</returns>
public async Task<List<string>> ListRolePoliciesAsync(string roleName)
{
    var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
    var policyNames = new List<string>();

    await foreach (var response in listRolePoliciesPaginator.Responses)
    {
        policyNames.AddRange(response.PolicyNames);
    }

    return policyNames;
}

/// <summary>
/// List IAM roles.
/// </summary>
/// <returns>A list of IAM roles.</returns>
public async Task<List<Role>> ListRolesAsync()
{
```

```
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }

    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }
}
```

```
    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

```
    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}

using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
```



```
{
    // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>()
                .AddTransient<IAMWrapper>()
                .AddTransient<UIWrapper>()
            )
        .Build();

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<IAMBasics>();

    IConfiguration 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();

    // Values needed for user, role, and policies.
    string userName = configuration["UserName"]!;
    string s3PolicyName = configuration["S3PolicyName"]!;
    string roleName = configuration["RoleName"]!;

    var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
    var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

    uiWrapper.DisplayBasicsOverview();
    uiWrapper.PressEnter();

    // First create a user. By default, the new user has
    // no permissions.
    uiWrapper.DisplayTitle("Create User");
    Console.WriteLine($"Creating a new user with user name: {userName}.");
    var user = await iamWrapper.CreateUserAsync(userName);
    var userArn = user.Arn;
}
```

```
    Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
    uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

    // Define a role policy document that allows the new user
    // to assume the role.
    string assumeRolePolicyDocument = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\"," +
            "\"Principal\": {" +
                "\"AWS\": \"{userArn}\"" +
            "}," +
            "\"Action\": \"sts:AssumeRole\"" +
        "}]}" +
    ";

    // Permissions to list all buckets.
    string policyDocument = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\" : [{" +
            "\"Action\" : [\"s3:ListAllMyBuckets\"]," +
            "\"Effect\" : \"Allow\"," +
            "\"Resource\" : \"*\\"" +
        "}]}" +
    ";

    // Create an AccessKey for the user.
    uiWrapper.DisplayTitle("Create access key");
    Console.WriteLine("Now let's create an access key for the new user.");
    var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

    var accessKeyId = accessKey.AccessKeyId;
    var secretAccessKey = accessKey.SecretAccessKey;

    Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

    Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
    var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

    // Now try listing the Amazon Simple Storage Service (Amazon S3)
```

```
// buckets. This should fail at this point because the user doesn't
// have permissions to perform this task.
uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
Console.WriteLine("Now let's try to display a list of the user's Amazon S3
buckets.");
var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
var buckets = await s3Wrapper.ListMyBucketsAsync();

Console.WriteLine(buckets is null
    ? "As expected, the call to list the buckets has returned a null list."
    : "Something went wrong. This shouldn't have worked.");

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Create IAM role");
Console.WriteLine($"Creating the role: {roleName}");

// Creating an IAM role to allow listing the S3 buckets. A role name
// is not case sensitive and must be unique to the account for which it
// is created.
var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

uiWrapper.PressEnter();

// Create a policy with permissions to list S3 buckets.
uiWrapper.DisplayTitle("Create IAM policy");
Console.WriteLine($"Creating the policy: {s3PolicyName}");
Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

// Wait 15 seconds for the IAM policy to be available.
uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

// Attach the policy to the role you created earlier.
uiWrapper.DisplayTitle("Attach new IAM policy");
Console.WriteLine("Now let's attach the policy to the role.");
await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);
```

```
// Wait 15 seconds for the role to be updated.
Console.WriteLine();
uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

// Use the AWS Security Token Service (AWS STS) to have the user
// assume the role we created.
var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

// Wait for the new credentials to become valid.
uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

// Try again to list the buckets using the client created with
// the new user's credentials. This time, it should work.
var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

s3Wrapper.UpdateClients(s3Client2, stsClient2);

buckets = await s3Wrapper.ListMyBucketsAsync();

uiWrapper.DisplayTitle("List Amazon S3 buckets");
Console.WriteLine("This time we should have buckets to list.");
if (buckets is not null)
{
    buckets.ForEach(bucket =>
    {
        Console.WriteLine($"{bucket.BucketName} created:
{bucket.CreationDate}");
    });
}

uiWrapper.PressEnter();

// Now clean up all the resources used in the example.
uiWrapper.DisplayTitle("Clean up resources");
Console.WriteLine("Thank you for watching. The IAM Basics demo is
complete.");
Console.WriteLine("Please wait while we clean up the resources we
created.");
```

```
        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
patience.");
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
```

```
/// Assumes an AWS Identity and Access Management (IAM) role that allows
/// Amazon S3 access for the current session.
/// </summary>
/// <param name="roleSession">A string representing the current session.</param>
/// <param name="roleToAssume">The name of the IAM role to assume.</param>
/// <returns>Credentials for the newly assumed IAM role.</returns>
public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
{
    // Create the request to use with the AssumeRoleAsync call.
    var request = new AssumeRoleRequest()
    {
        RoleSessionName = roleSession,
        RoleArn = roleToAssume,
    };

    var response = await _stsService.AssumeRoleAsync(request);

    return response.Credentials;
}

/// <summary>
/// Delete an S3 bucket.
/// </summary>
/// <param name="bucketName">Name of the S3 bucket to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
    return result.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the buckets that are owned by the user's account.
/// </summary>
/// <returns>Async Task.</returns>
public async Task<List<S3Bucket>?> ListMyBucketsAsync()
{
    try
    {
        // Get the list of buckets accessible by the new user.
        var response = await _s3Service.ListBucketsAsync();
    }
}
```

```
        return response.Buckets;
    }
    catch (AmazonS3Exception ex)
    {
        // Something else went wrong. Display the error message.
        Console.WriteLine($"Error: {ex.Message}");
        return null;
    }
}

/// <summary>
/// Create a new S3 bucket.
/// </summary>
/// <param name="bucketName">The name for the new bucket.</param>
/// <returns>A Boolean value indicating whether the action completed
/// successfully.</returns>
public async Task<bool> PutBucketAsync(string bucketName)
{
    var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Update the client objects with new client objects. This is available
/// because the scenario uses the methods of this class without and then
/// with the proper permissions to list S3 buckets.
/// </summary>
/// <param name="s3Service">The Amazon S3 client object.</param>
/// <param name="stsService">The AWS STS client object.</param>
public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
{
    _s3Service = s3Service;
    _stsService = stsService;
}
}

namespace IamScenariosCommon;

public class UIWrapper
{
```

```
public readonly string SepBar = new('-', Console.WindowWidth);

/// <summary>
/// Show information about the IAM Groups scenario.
/// </summary>
public void DisplayGroupsOverview()
{
    Console.Clear();

    DisplayTitle("Welcome to the IAM Groups Demo");
    Console.WriteLine("This example application does the following:");
    Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
    Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
access to Amazon S3.");
    Console.WriteLine("\t3. Creates a new IAM user.");
    Console.WriteLine("\t4. Creates an IAM access key for the user.");
    Console.WriteLine("\t5. Adds the user to the IAM group.");
    Console.WriteLine("\t6. Lists the buckets on the account.");
    Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
    Console.WriteLine("\t8. List the buckets again to show the new bucket.");
    Console.WriteLine("\t9. Cleans up all the resources created.");
}

/// <summary>
/// Show information about the IAM Basics scenario.
/// </summary>
public void DisplayBasicsOverview()
{
    Console.Clear();

    DisplayTitle("Welcome to IAM Basics");
    Console.WriteLine("This example application does the following:");
    Console.WriteLine("\t1. Creates a user with no permissions.");
    Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
    Console.WriteLine("\t3. Grants the user permission to assume the role.");
    Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
    Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
    Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
    Console.WriteLine("\t7. Deletes all the resources.");
}
```



```
}

/// <summary>
/// Display a message and wait until the user presses enter.
/// </summary>
public void PressEnter()
{
    Console.WriteLine("\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();
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Beispiele für Amazon Keyspaces mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Keyspaces verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Keyspaces

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon Keyspaces.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace KeyspacesActions;

public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache Cassandra).
        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<IAmazonKeyspaces>()
                    .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
```

```
        .CreateLogger<HelloKeyspaces>());

        var keyspacesClient = host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- Weitere API-Informationen finden Sie unter [ListKeyspaces](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen eines Keyspace

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Keyspaces Keyspace erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
```

```
var response =
    await _amazonKeyspaces.CreateKeyspaceAsync(
        new CreateKeyspaceRequest { KeyspaceName = keySpaceName });
return response.ResourceArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateKeyspace](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Tabelle

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Keyspaces-Tabelle erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a new Amazon Keyspaces table.
/// </summary>
/// <param name="keySpaceName">The keyspace where the table will be created.</
param>
/// <param name="schema">The schema for the new table.</param>
/// <param name="tableName">The name of the new table.</param>
/// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
public async Task<string> CreateTable(string keySpaceName, SchemaDefinition
schema, string tableName)
{
    var request = new CreateTableRequest
    {
        KeyspaceName = keySpaceName,
        SchemaDefinition = schema,
        TableName = tableName,
        PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
    }
}
```

```
};

var response = await _amazonKeyspaces.CreateTableAsync(request);
return response.ResourceArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateTable](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Keyspace

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Keyspaces Keyspace löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteKeyspace](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Tabelle

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Keyspaces-Tabelle löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTable(string keyspaceName, string tableName)
{
    var response = await _amazonKeyspaces.DeleteTableAsync(
        new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteTable](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Daten zu einem Keyspace

Das folgende Codebeispiel zeigt, wie Sie Daten über einen Amazon Keyspaces-Keyspace abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.GetKeyspaceAsync(
        new GetKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- Weitere API-Informationen finden Sie unter [GetKeyspace](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Daten zu einer Tabelle

Das folgende Codebeispiel zeigt, wie Sie Daten zu einer Amazon Keyspaces-Tabelle abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information about an Amazon Keyspaces table.
/// </summary>
```



```

    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }

```

- Weitere API-Informationen finden Sie unter [GetTable](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Keyspaces

Das folgende Codebeispiel zeigt, wie Sie Amazon Keyspaces Keyspaces auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)

```

```
    {  
  
    Console.WriteLine($"{keyspace.KeyspaceName, -30}\t{keyspace.ResourceArn}");  
    }  
}
```

- Weitere API-Informationen finden Sie unter [ListKeyspaces](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Tabellen in einem Keyspace

Das folgende Codebeispiel zeigt, wie Sie Amazon-Keyspaces-Tabellen in einem Keyspace auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Lists the Amazon Keyspaces tables in a keyspace.  
/// </summary>  
/// <param name="keyspaceName">The name of the keyspace.</param>  
/// <returns>A list of TableSummary objects.</returns>  
public async Task<List<TableSummary>> ListTables(string keyspaceName)  
{  
    var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest  
{ KeyspaceName = keyspaceName });  
    response.Tables.ForEach(table =>  
    {  
  
Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");  
    });  
  
    return response.Tables;  
}
```

- Weitere API-Informationen finden Sie unter [ListTables](#) in der APIAWS SDK for .NET -Referenz für .

Wiederherstellen einer Tabelle zu einem bestimmten Zeitpunkt

Das folgende Codebeispiel zeigt, wie eine Amazon Keyspaces-Tabelle zu einem bestimmten Zeitpunkt wiederhergestellt wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Restores the specified table to the specified point in time.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to restore.</param>
/// <param name="timestamp">The time to which the table will be restored.</
param>
/// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
public async Task<string> RestoreTable(string keyspaceName, string tableName,
string restoredTableName, DateTime timestamp)
{
    var request = new RestoreTableRequest
    {
        RestoreTimestamp = timestamp,
        SourceKeyspaceName = keyspaceName,
        SourceTableName = tableName,
        TargetKeyspaceName = keyspaceName,
        TargetTableName = restoredTableName
    };

    var response = await _amazonKeyspaces.RestoreTableAsync(request);
    return response.RestoredTableARN;
}
```

- Weitere API-Informationen finden Sie unter [RestoreTable](#) in der APIAWS SDK for .NET - Referenz für .

Aktualisieren einer Tabelle

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Keyspaces-Tabelle aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
        KeyspaceName = keyspaceName,
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
```

- Weitere API-Informationen finden Sie unter [UpdateTable](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit Keyspaces und Tabellen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Keyspace und eine Tabelle. Das Tabellenschema enthält Filmdaten und die point-in-time Wiederherstellung ist aktiviert.
- Stellen Sie über eine sichere TLS-Verbindung mit SigV4-Authentifizierung eine Verbindung zum Keyspace her.
- Fragen Sie die Tabelle ab. Fügen Sie Filmdaten hinzu, rufen Sie sie ab und aktualisieren Sie sie.
- Aktualisieren Sie die Tabelle. Fügen Sie eine Spalte hinzu, um angesehene Filme zu verfolgen.
- Setzen Sie die Tabelle in den vorherigen Zustand zurück und bereinigen Sie Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
global using System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
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;
global using Newtonsoft.Json;

namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
```

```
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = 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<IAmazonKeyspaces>()
                    .AddTransient<KeyspacesWrapper>()
                    .AddTransient<CassandraWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        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 keyspacesWrapper = host.Services.GetRequiredService<KeyspacesWrapper>();
        var uiMethods = new UiMethods();

        var keyspaceName = configuration["KeyspaceName"];
        var tableName = configuration["TableName"];

        bool success; // Used to track the results of some operations.

        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the keyspace.
    }
}
```

```
var keySpaceArn = await keyspacesWrapper.CreateKeyspace(keySpaceName);

// Wait for the keyspace to be available. GetKeyspace results in a
// resource not found error until it is ready for use.
try
{
    var getKeySpaceArn = "";
    Console.WriteLine($"Created {keySpaceName}. Waiting for it to become
available. ");
    do
    {
        getKeySpaceArn = await keyspacesWrapper.GetKeyspace(keySpaceName);
        Console.WriteLine(". ");
    } while (getKeySpaceArn != keySpaceArn);
}
catch (ResourceNotFoundException)
{
    Console.WriteLine("Waiting for keyspace to be created.");
}

Console.WriteLine($"\\nThe keyspace {keySpaceName} is ready for use.");

uiMethods.PressEnter();

// Create the table.
// First define the schema.
var allColumns = new List<ColumnDefinition>
{
    new ColumnDefinition { Name = "title", Type = "text" },
    new ColumnDefinition { Name = "year", Type = "int" },
    new ColumnDefinition { Name = "release_date", Type = "timestamp" },
    new ColumnDefinition { Name = "plot", Type = "text" },
};

var partitionKeys = new List<PartitionKey>
{
    new PartitionKey { Name = "year", },
    new PartitionKey { Name = "title" },
};

var tableSchema = new SchemaDefinition
{
    AllColumns = allColumns,
    PartitionKeys = partitionKeys,
```

```
};

    var tableArn = await keyspacesWrapper.CreateTable(keyspaceName, tableSchema,
tableName);

    // Wait for the table to be active.
    try
    {
        var resp = new GetTableResponse();
        Console.WriteLine("Waiting for the new table to be active. ");
        do
        {
            try
            {
                resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
                Console.WriteLine(".");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine(".");
            }
        } while (resp.Status != TableStatus.ACTIVE);

        // Display the table's schema.
        Console.WriteLine($"\\nTable {tableName} has been created in
{keyspaceName}");
        Console.WriteLine("Let's take a look at the schema.");
        uiMethods.DisplayTitle("All columns");
        resp.SchemaDefinition.AllColumns.ForEach(column =>
        {
            Console.WriteLine($"{column.Name, -40}\\t{column.Type, -20}");
        });

        uiMethods.DisplayTitle("Cluster keys");
        resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
        {
            Console.WriteLine($"{clusterKey.Name, -40}\\t{clusterKey.OrderBy, -20}");
        });

        uiMethods.DisplayTitle("Partition keys");
        resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
        {
            Console.WriteLine($"{partitionKey.Name}");
        });
    }
}
```



```
    });

    uiMethods.PressEnter();
}
catch (ResourceNotFoundException ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}

// Access Apache Cassandra using the Cassandra drive for C#.
var cassandraWrapper = host.Services.GetRequiredService<CassandraWrapper>();
var movieFilePath = configuration["MovieFile"];

Console.WriteLine("Let's add some movies to the table we created.");
var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

uiMethods.PressEnter();

Console.WriteLine("Added the following movies to the table:");
var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
uiMethods.DisplayTitle("All Movies");

foreach (var row in rows)
{
    var title = row.GetValue<string>("title");
    var year = row.GetValue<int>("year");
    var plot = row.GetValue<string>("plot");
    var release_date = row.GetValue<DateTime>("release_date");
    Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
    Console.WriteLine(uiMethods.SepBar);
}

// Update the table schema
uiMethods.DisplayTitle("Update table schema");
Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

// First save the current time as a UTC Date so the original
// table can be restored later.
var timeChanged = DateTime.UtcNow;

// Now update the schema.
```

```
var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
uiMethods.PressEnter();

Console.WriteLine("Now let's mark some of the movies as watched.");

// Pick some files to mark as watched.
var movieToWatch = rows[2].GetValue<string>("title");
var watchedMovieYear = rows[2].GetValue<int>("year");
var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[6].GetValue<string>("title");
watchedMovieYear = rows[6].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[9].GetValue<string>("title");
watchedMovieYear = rows[9].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[10].GetValue<string>("title");
watchedMovieYear = rows[10].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[13].GetValue<string>("title");
watchedMovieYear = rows[13].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

uiMethods.DisplayTitle("Watched movies");
Console.WriteLine("These movies have been marked as watched:");
rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
foreach (var row in rows)
{
    var title = row.GetValue<string>("title");
    var year = row.GetValue<int>("year");
    Console.WriteLine($"{title, -40}\t{year, 8}");
}
uiMethods.PressEnter();
```

```
    Console.WriteLine("We can restore the table to its previous state but that
can take up to 20 minutes to complete.");
    string answer;
    do
    {
        Console.WriteLine("Do you want to restore the table? (y/n)");
        answer = Console.ReadLine();
    } while (answer.ToLower() != "y" && answer.ToLower() != "n");

    if (answer == "y")
    {
        var restoredTableName = $"{tableName}_restored";
        var restoredTableArn = await keyspacesWrapper.RestoreTable(
            keyspaceName,
            tableName,
            restoredTableName,
            timeChanged);
        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasRestored = false;

        try
        {
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                wasRestored = (resp.Status == TableStatus.ACTIVE);
            } while (!wasRestored);
        }
        catch (ResourceNotFoundException)
        {
            // If the restored table raised an error, it isn't
            // ready yet.
            Console.Write(".");
        }
    }

    uiMethods.DisplayTitle("Clean up resources.");

    // Delete the table.
    success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);
```

```
        Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
        Console.WriteLine("Waiting for the table to be removed completely. ");

        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasDeleted = false;

        try
        {
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
            } while (!wasDeleted);
        }
        catch (ResourceNotFoundException ex)
        {
            wasDeleted = true;
            Console.WriteLine($"{ex.Message} indicates that the table has been
deleted.");
        }

        // Delete the keyspace.
        success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
        Console.WriteLine("The keyspace has been deleted and the demo is now
complete.");
    }
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
```

```
/// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
{
    _amazonKeyspaces = amazonKeyspaces;
}

/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
    var response =
        await _amazonKeyspaces.CreateKeyspaceAsync(
            new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}

/// <summary>
/// Create a new Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace where the table will be created.</
param>
/// <param name="schema">The schema for the new table.</param>
/// <param name="tableName">The name of the new table.</param>
/// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
{
    var request = new CreateTableRequest
    {
        KeyspaceName = keyspaceName,
        SchemaDefinition = schema,
        TableName = tableName,
        PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
    };

    var response = await _amazonKeyspaces.CreateTableAsync(request);
    return response.ResourceArn;
}
```

```
/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTable(string keyspaceName, string tableName)
{
    var response = await _amazonKeyspaces.DeleteTableAsync(
        new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.GetKeyspaceAsync(
        new GetKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}

/// <summary>
/// Get information about an Amazon Keyspaces table.
```

```

    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }

    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {
            Console.WriteLine($"{keyspace.KeyspaceName, -30}\t{keyspace.ResourceArn}");
        }
    }

    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

```

```
Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
    });

    return response.Tables;
}

/// <summary>
/// Restores the specified table to the specified point in time.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to restore.</param>
/// <param name="timestamp">The time to which the table will be restored.</
param>
/// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
public async Task<string> RestoreTable(string keyspaceName, string tableName,
string restoredTableName, DateTime timestamp)
{
    var request = new RestoreTableRequest
    {
        RestoreTimestamp = timestamp,
        SourceKeyspaceName = keyspaceName,
        SourceTableName = tableName,
        TargetKeyspaceName = keyspaceName,
        TargetTableName = restoredTableName
    };

    var response = await _amazonKeyspaces.RestoreTableAsync(request);
    return response.RestoredTableARN;
}

/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
```



```

        KeyspaceName = keyspaceName,
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
}

```

```

using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
/// database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/repository/
sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _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();

_localPathToFile = Path.GetTempPath();

// Get the Starfield digital certificate and save it locally.
var client = new WebClient();
client.DownloadFile(_certLocation, $"{_localPathToFile}/{_certFileName}");

//var httpClient = new HttpClient();
//var httpResult = httpClient.Get(fileUrl);
//using var resultStream = await httpResult.Content.ReadAsStreamAsync();
//using var fileStream = File.Create(pathToSave);
//resultStream.CopyTo(fileStream);

_certCollection = new X509Certificate2Collection();
_amazoncert = new X509Certificate2($"{_localPathToFile}/{_certFileName}");

// Get the user name and password stored in the configuration file.
_userName = _configuration["UserName"]!;
_pwd = _configuration["Password"]!;

// For a list of Service Endpoints for Amazon Keyspaces, see:
// https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
var awsEndpoint = _configuration["ServiceEndpoint"];

_cluster = Cluster.Builder()
    .AddContactPoints(awsEndpoint)
    .WithPort(9142)
    .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
    .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
    .WithQueryOptions(
        new QueryOptions()
            .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
            .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
    .Build();
}

/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the Apache Cassandra table.
```

```
/// </summary>
/// <param name="movieFileName">The full path to the JSON file.</param>
/// <returns>A list of movie objects.</returns>
public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport =
0)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();

    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    // If numToImport = 0, return all movies in the collection.
    if (numToImport == 0)
    {
        // Now return the entire list of movies.
        return allMovies;
    }
    else
    {
        // Now return the first numToImport entries.
        return allMovies.GetRange(0, numToImport);
    }
}

/// <summary>
/// Insert movies into the movie table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="movieTableName">The Amazon Keyspaces table.</param>
/// <param name="movieFilePath">The path to the resource file containing
/// movie data to insert into the table.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
{
    // Get some movie data from the movies.json file
    var movies = ImportMoviesFromJson(movieFilePath, numToImport);

    var session = _cluster.Connect(keyspaceName);
```

```
string insertCql;

RowSet rs;

// Now we insert the numToImport movies into the table.
foreach (var movie in movies)
{
    // Escape single quote characters in the plot.
    insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values({{movie.Title}}$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")} ', $${movie.Info.Plot}}$)";
    rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
}

return true;
}

/// <summary>
/// Gets all of the movies in the movies table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <returns>A list of row objects containing movie data.</returns>
public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
{
    var session = _cluster.Connect();
    RowSet rs;
    try
    {
        rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

        // Extract the row data from the returned RowSet.
        var rows = rs.GetRows().ToList();
        return rows;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}
```

```
/// <summary>
/// Mark a movie in the movie table as watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <param name="title">The title of the movie to mark as watched.</param>
/// <param name="year">The year the movie was released.</param>
/// <returns>A set of rows containing the changed data.</returns>
public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
{
    var session = _cluster.Connect();
    string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
WHERE title = ${title} AND year = {year}";
    var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
    var rows = rs.GetRows().ToList();
    return rows;
}

/// <summary>
/// Retrieve the movies in the movies table where watched is true.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <returns>A list of row objects containing information about movies
/// where watched is true.</returns>
public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
tableName)
{
    var session = _cluster.Connect();
    RowSet rs;
    try
    {
        rs = await session.ExecuteAsync(new SimpleStatement($"SELECT title,
year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW FILTERING"));

        // Extract the row data from the returned RowSet.
        var rows = rs.GetRows().ToList();
        return rows;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}
```

```
    }  
  }  
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateKeyspace](#)
 - [CreateTable](#)
 - [DeleteKeyspace](#)
 - [DeleteTable](#)
 - [GetKeyspace](#)
 - [GetTable](#)
 - [ListKeyspaces](#)
 - [ListTables](#)
 - [RestoreTable](#)
 - [UpdateTable](#)

Kinesis-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Kinesis verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Serverless-Beispiele](#)

Aktionen

Tags hinzufügen

Das folgende Codebeispiel zeigt, wie Sie einem Kinesis-Stream Tags hinzufügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        var tags = new Dictionary<string, string>
        {
            { "Project", "Sample Kinesis Project" },
            { "Application", "Sample Kinesis App" },
        };

        var success = await ApplyTagsToStreamAsync(client, streamName, tags);

        if (success)
        {
            Console.WriteLine($"Tags successfully added to {streamName}.");
        }
    }
}
```

```
    }
    else
    {
        Console.WriteLine("Tags were not added to the stream.");
    }
}

/// <summary>
/// Applies the set of tags to the named Kinesis stream.
/// </summary>
/// <param name="client">The initialized Kinesis client.</param>
/// <param name="streamName">The name of the Kinesis stream to which
/// the tags will be attached.</param>
/// <param name="tags">A dictionary containing key/value pairs which
/// will be used to create the Kinesis tags.</param>
/// <returns>A Boolean value which represents the success or failure
/// of AddTagsToStreamAsync.</returns>
public static async Task<bool> ApplyTagsToStreamAsync(
    IAmazonKinesis client,
    string streamName,
    Dictionary<string, string> tags)
{
    var request = new AddTagsToStreamRequest
    {
        StreamName = streamName,
        Tags = tags,
    };

    var response = await client.AddTagsToStreamAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
}
```

- Weitere API-Informationen finden Sie unter [AddTagsToStream](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines -Streams

Das folgende Codebeispiel zeigt, wie Sie einen Kinesis-Stream erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to create a new Amazon Kinesis stream.
/// </summary>
public class CreateStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        int shardCount = 1;

        var success = await CreateNewStreamAsync(client, streamName,
shardCount);
        if (success)
        {
            Console.WriteLine($"The stream, {streamName} successfully
created.");
        }
    }

    /// <summary>
    /// Creates a new Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client.</param>
    /// <param name="streamName">The name for the new stream.</param>
    /// <param name="shardCount">The number of shards the new stream will
    /// use. The throughput of the stream is a function of the number of
    /// shards; more shards are required for greater provisioned
```

```
    /// throughput.</param>
    /// <returns>A Boolean value indicating whether the stream was created.</
returns>
    public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis client,
string streamName, int shardCount)
    {
        var request = new CreateStreamRequest
        {
            StreamName = streamName,
            ShardCount = shardCount,
        };

        var response = await client.CreateStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateStream](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Streams

Das folgende Codebeispiel zeigt, wie Sie einen Kinesis-Stream löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to delete an Amazon Kinesis stream.
```

```
/// </summary>
public class DeleteStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        var success = await DeleteStreamAsync(client, streamName);

        if (success)
        {
            Console.WriteLine($"Stream, {streamName} successfully deleted.");
        }
        else
        {
            Console.WriteLine("Stream not deleted.");
        }
    }

    /// <summary>
    /// Deletes a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the string to delete.</param>
    /// <returns>A Boolean value representing the success of the operation.</
returns>
    public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
string streamName)
    {
        // If EnforceConsumerDeletion is true, any consumers
        // of this stream will also be deleted. If it is set
        // to false and this stream has any consumers, the
        // call will fail with a ResourceInUseException.
        var request = new DeleteStreamRequest
        {
            StreamName = streamName,
            EnforceConsumerDeletion = true,
        };

        var response = await client.DeleteStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [DeleteStream](#) in der APIAWS SDK for .NET - Referenz für .

Einen Verbraucher abmelden

Das folgende Codebeispiel zeigt, wie Sie einen Verbraucher von einem Kinesis-Stream abmelden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to deregister a consumer from an Amazon Kinesis stream.
/// </summary>
public class DeregisterConsumer
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
        string consumerName = "CONSUMER_NAME";
        string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

        var success = await DeregisterConsumerAsync(client, streamARN,
consumerARN, consumerName);
```

```
        if (success)
        {
            Console.WriteLine($"{consumerName} successfully deregistered.");
        }
        else
        {
            Console.WriteLine($"{consumerName} was not successfully
deregistered.");
        }
    }

    /// <summary>
    /// Deregisters a consumer from a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamARN">The ARN of a Kinesis stream.</param>
    /// <param name="consumerARN">The ARN of the consumer.</param>
    /// <param name="consumerName">The name of the consumer.</param>
    /// <returns>A Boolean value representing the success of the operation.</
returns>
    public static async Task<bool> DeregisterConsumerAsync(
        IAmazonKinesis client,
        string streamARN,
        string consumerARN,
        string consumerName)
    {
        var request = new DeregisterStreamConsumerRequest
        {
            StreamARN = streamARN,
            ConsumerARN = consumerARN,
            ConsumerName = consumerName,
        };

        var response = await client.DeregisterStreamConsumerAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeregisterStreamConsumer](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Streams

Das folgende Codebeispiel zeigt, wie Informationen zu einem oder mehreren Kinesis-Streams aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Retrieves and displays a list of existing Amazon Kinesis streams.
/// </summary>
public class ListStreams
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        var response = await client.ListStreamsAsync(new ListStreamsRequest());

        List<string> streamNames = response.StreamNames;

        if (streamNames.Count > 0)
        {
            streamNames
                .ForEach(s => Console.WriteLine($"Stream name: {s}"));
        }
        else
        {
            Console.WriteLine("No streams were found.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListStreams](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Tags

Das folgende Codebeispiel zeigt, wie Sie die Tags auflisten, die einem Kinesis-Stream zugeordnet sind.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to list the tags that have been attached to an Amazon Kinesis
/// stream.
/// </summary>
public class ListTags
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        await ListTagsAsync(client, streamName);
    }

    /// <summary>
    /// List the tags attached to a Kinesis stream.

```

```

    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the Kinesis stream for which you
    /// wish to display tags.</param>
    public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
    {
        var request = new ListTagsForStreamRequest
        {
            StreamName = streamName,
            Limit = 10,
        };

        var response = await client.ListTagsForStreamAsync(request);
        DisplayTags(response.Tags);

        while (response.HasMoreTags)
        {
            request.ExclusiveStartTagKey = response.Tags[response.Tags.Count -
1].Key;
            response = await client.ListTagsForStreamAsync(request);
        }
    }

    /// <summary>
    /// Displays the items in a list of Kinesis tags.
    /// </summary>
    /// <param name="tags">A list of the Tag objects to be displayed.</param>
    public static void DisplayTags(List<Tag> tags)
    {
        tags
            .ForEach(t => Console.WriteLine($"Key: {t.Key} Value: {t.Value}"));
    }
}

```

- Weitere API-Informationen finden Sie unter [ListTagsForStream](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten der Verbraucher eines Streams

Das folgende Codebeispiel zeigt, wie Sie die Verbraucher eines Kinesis-Streams auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// List the consumers of an Amazon Kinesis stream.
/// </summary>
public class ListConsumers
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
        int maxResults = 10;

        var consumers = await ListConsumersAsync(client, streamARN, maxResults);

        if (consumers.Count > 0)
        {
            consumers
                .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
{c.ConsumerARN}"));
        }
        else
        {
            Console.WriteLine("No consumers found.");
        }
    }

    /// <summary>
```

```
    /// Retrieve a list of the consumers for a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamARN">The ARN of the stream for which we want to
    /// retrieve a list of clients.</param>
    /// <param name="maxResults">The maximum number of results to return.</
param>
    /// <returns>A list of Consumer objects.</returns>
    public static async Task<List<Consumer>> ListConsumersAsync(IAmazonKinesis
client, string streamARN, int maxResults)
    {
        var request = new ListStreamConsumersRequest
        {
            StreamARN = streamARN,
            MaxResults = maxResults,
        };

        var response = await client.ListStreamConsumersAsync(request);

        return response.Consumers;
    }
}
```

- Weitere API-Informationen finden Sie unter [ListStreamConsumers](#) in der APIAWS SDK for .NET -Referenz für .

Registrieren eines Verbrauchers

Das folgende Codebeispiel zeigt, wie Sie einen Verbraucher in einem Kinesis-Stream registrieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to register a consumer to an Amazon Kinesis
/// stream.
/// </summary>
public class RegisterConsumer
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string consumerName = "NEW_CONSUMER_NAME";
        string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

        var consumer = await RegisterConsumerAsync(client, consumerName,
streamARN);

        if (consumer is not null)
        {
            Console.WriteLine($"{consumer.ConsumerName}");
        }
    }

    /// <summary>
    /// Registers the consumer to a Kinesis stream.
    /// </summary>
    /// <param name="client">The initialized Kinesis client object.</param>
    /// <param name="consumerName">A string representing the consumer.</param>
    /// <param name="streamARN">The ARN of the stream.</param>
    /// <returns>A Consumer object that contains information about the
consumer.</returns>
    public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
client, string consumerName, string streamARN)
    {
        var request = new RegisterStreamConsumerRequest
        {
            ConsumerName = consumerName,
            StreamARN = streamARN,
        };

        var response = await client.RegisterStreamConsumerAsync(request);
        return response.Consumer;
    }
}
```

```
}  
}
```

- Weitere API-Informationen finden Sie unter [RegisterStreamConsumer](#) in der APIAWS SDK for .NET -Referenz für .

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von .NET.

```
using System.Text;  
using Amazon.Lambda.Core;  
using Amazon.Lambda.KinesisEvents;  
using AWS.Lambda.Powertools.Logging;  
  
// Assembly attribute to enable the Lambda function's JSON input to be converted  
// into a .NET class.  
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali  
  
namespace KinesisIntegrationSampleCode;  
  
public class Function  
{  
    // Powertools Logger requires an environment variables against your function
```

```
// POWERTOOLS_SERVICE_NAME
[Logging(LogEvent = true)]
public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
{
    if (evnt.Records.Count == 0)
    {
        Logger.LogInformation("Empty Kinesis Event received");
        return;
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            throw;
        }
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von .NET.

```
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }
    }
}
```

```
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            /* Since we are working with streams, we can return the failed item
immediately.
            Lambda will immediately begin to retry processing from this
failed item onwards. */
            return new StreamsEventResponse
            {
                BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                }
            };
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

```
public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

AWS KMS -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS KMS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen


- [Aktionen](#)

Aktionen

Erstellen einer Erteilung für einen Schlüssel

Das folgende Codebeispiel zeigt, wie Sie eine Erteilung für einen KMS-Schlüssel erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
public static async Task Main()
{
    var client = new AmazonKeyManagementServiceClient();

    // The identity that is given permission to perform the operations
    // specified in the grant.
    var grantee = "arn:aws:iam::111122223333:role/ExampleRole";

    // The identifier of the AWS KMS key to which the grant applies. You
    // can use the key ID or the Amazon Resource Name (ARN) of the KMS key.
    var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";

    var request = new CreateGrantRequest
    {
        GranteePrincipal = grantee,
        KeyId = keyId,

        // A list of operations that the grant allows.
        Operations = new List<string>
        {
            "Encrypt",
            "Decrypt",
        },
    };

    var response = await client.CreateGrantAsync(request);

    string grantId = response.GrantId; // The unique identifier of the
grant.
    string grantToken = response.GrantToken; // The grant token.

    Console.WriteLine($"Id: {grantId}, Token: {grantToken}");
}
}
```

- Weitere API-Informationen finden Sie unter [CreateGrant](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines -Schlüssels

Das folgende Codebeispiel zeigt, wie Sie ein erstellen AWS KMS key.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to create a new AWS Key Management Service (AWS KMS)
/// key.
/// </summary>
public class CreateKey
{
    public static async Task Main()
    {
        // Note that if you need to create a Key in an AWS Region
        // other than the Region defined for the default user, you need to
        // pass the Region to the client constructor.
        var client = new AmazonKeyManagementServiceClient();

        // The call to CreateKeyAsync will create a symmetrical AWS KMS
        // key. For more information about symmetrical and asymmetrical
        // keys, see:
        //
        // https://docs.aws.amazon.com/kms/latest/developerguide/symm-asymm-
        choose.html
    }
}
```

```
        var response = await client.CreateKeyAsync(new CreateKeyRequest());

        // The KeyMetadata object contains information about the new AWS KMS
key.
        KeyMetadata keyMetadata = response.KeyMetadata;

        if (keyMetadata is not null)
        {
            Console.WriteLine($"KMS Key: {keyMetadata.KeyId} was successfully
created.");
        }
        else
        {
            Console.WriteLine("Could not create KMS Key.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateKey](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Alias für einen Schlüssel

Das folgende Codebeispiel zeigt, wie Sie einen Alias für einen KMS-Schlüssel erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Creates an alias for an AWS Key Management Service (AWS KMS) key.
```

```
/// </summary>
public class CreateAlias
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The alias name must start with alias/ and can be
        // up to 256 alphanumeric characters long.
        var aliasName = "alias/ExampleAlias";

        // The value supplied as the TargetKeyId can be either
        // the key ID or key Amazon Resource Name (ARN) of the
        // AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new CreateAliasRequest
        {
            AliasName = aliasName,
            TargetKeyId = keyId,
        };

        var response = await client.CreateAliasAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Alias, {aliasName}, successfully created.");
        }
        else
        {
            Console.WriteLine($"Could not create alias.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateAlias](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben eines Schlüssels

Das folgende Codebeispiel zeigt, wie Sie einen KMS-Schlüssel beschreiben.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Retrieve information about an AWS Key Management Service (AWS KMS) key.
/// You can supply either the key Id or the key Amazon Resource Name (ARN)
/// to the DescribeKeyRequest KeyId property.
/// </summary>
public class DescribeKey
{
    public static async Task Main()
    {
        var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
        var request = new DescribeKeyRequest
        {
            KeyId = keyId,
        };

        var client = new AmazonKeyManagementServiceClient();

        var response = await client.DescribeKeyAsync(request);
        var metadata = response.KeyMetadata;

        Console.WriteLine($"{metadata.KeyId} created on:
{metadata.CreationDate}");
        Console.WriteLine($"State: {metadata.KeyState}");
        Console.WriteLine($"{metadata.Description}");
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeKey](#) in der APIAWS SDK for .NET - Referenz für .

Deaktivieren eines Schlüssels

Das folgende Codebeispiel zeigt, wie Sie einen KMS-Schlüssel deaktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Disable an AWS Key Management Service (AWS KMS) key and then retrieve
/// the key's status to show that it has been disabled.
/// </summary>
public class DisableKey
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The identifier of the AWS KMS key to disable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new DisableKeyRequest
        {
            KeyId = keyId,
        };

        var response = await client.DisableKeyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {
            // Retrieve information about the key to show that it has now
            // been disabled.
            var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
            {
                KeyId = keyId,
            });
            Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [DisableKey](#) in der APIAWS SDK for .NET -Referenz für .

Aktivieren eines Schlüssels

Das folgende Codebeispiel zeigt, wie Sie einen KMS-Schlüssel aktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Enable an AWS Key Management Service (AWS KMS) key.
/// </summary>
public class EnableKey
{
```

```
public static async Task Main()
{
    var client = new AmazonKeyManagementServiceClient();

    // The identifier of the AWS KMS key to enable. You can use the
    // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
    var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

    var request = new EnableKeyRequest
    {
        KeyId = keyId,
    };

    var response = await client.EnableKeyAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        // Retrieve information about the key to show that it has now
        // been enabled.
        var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
        {
            KeyId = keyId,
        });
        Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
    }
}
```

- Weitere API-Informationen finden Sie unter [EnableKey](#) in der APIAWS SDK for .NET -Referenz zu .

Auflisten von Aliassen für einen Schlüssel

Das folgende Codebeispiel zeigt, wie Sie Aliase für einen KMS-Schlüssel auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// List the AWS Key Management Service (AWS KMS) aliases that have been defined
for
/// the keys in the same AWS Region as the default user. If you want to list
/// the aliases in a different Region, pass the Region to the client
/// constructor.
/// </summary>
public class ListAliases
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListAliasesRequest();
        var response = new ListAliasesResponse();

        do
        {
            response = await client.ListAliasesAsync(request);

            response.Aliases.ForEach(alias =>
            {
                Console.WriteLine($"Created: {alias.CreationDate} Last Update:
{alias.LastUpdatedDate} Name: {alias.AliasName}");
            });

            request.Marker = response.NextMarker;
        }
        while (response.Truncated);
    }
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [ListAliases](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Erteilungen für einen Schlüssel

Das folgende Codebeispiel zeigt, wie Sie Erteilungen für einen KMS-Schlüssel auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// List the AWS Key Management Service (AWS KMS) grants that are associated
with
/// a specific key.
/// </summary>
public class ListGrants
{
    public static async Task Main()
    {
        // The identifier of the AWS KMS key to disable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListGrantsRequest
        {
            KeyId = keyId,
        };
    };
}
```

```
var response = new ListGrantsResponse();

do
{
    response = await client.ListGrantsAsync(request);

    response.Grants.ForEach(grant =>
    {
        Console.WriteLine($"{grant.GrantId}");
    });

    request.Marker = response.NextMarker;
}
while (response.Truncated);
}
```

- Weitere API-Informationen finden Sie unter [ListGrants](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Schlüsseln

Das folgende Codebeispiel zeigt, wie Sie KMS-Schlüssel auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

///  
/// <summary>
```

```
/// List the AWS Key Managements Service (AWS KMS) keys for the AWS Region
/// of the default user. To list keys in another AWS Region, supply the Region
/// as a parameter to the client constructor.
/// </summary>
public class ListKeys
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListKeysRequest();
        var response = new ListKeysResponse();

        do
        {
            response = await client.ListKeysAsync(request);

            response.Keys.ForEach(key =>
            {
                Console.WriteLine($"ID: {key.KeyId}, {key.KeyArn}");
            });

            // Set the Marker property when response.Truncated is true
            // in order to get the next keys.
            request.Marker = response.NextMarker;
        }
        while (response.Truncated);
    }
}
```

- Weitere API-Informationen finden Sie unter [ListKeys](#) in der APIAWS SDK for .NET -Referenz für

Lambda-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Lambda verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Lambda

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von Lambda beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {
            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [ListFunctions](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

Erstellen einer -Funktion

Das folgende Codebeispiel zeigt, wie Sie eine Lambda-Funktion erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
```

```
string s3Key,
string role,
string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //           the source code is stored.
    // S3Key - The name of the file containing the code.
    var functionCode = new FunctionCode
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };

    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateFunction](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer -Funktion

Das folgende Codebeispiel zeigt, wie Sie eine Lambda-Funktion löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}
```

- Weitere API-Informationen finden Sie unter [DeleteFunction](#) in der APIAWS SDK for .NET - Referenz für .

Funktion abrufen

Das folgende Codebeispiel veranschaulicht, wie eine Lambda-Funktion aufgerufen wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };


    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}
```

- Weitere API-Informationen finden Sie unter [GetFunction](#) in der APIAWS SDK for .NET - Referenz für .

Aufruf einer -Funktion

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion aufgerufen wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {
        FunctionName = functionName,
        Payload = payload,
    };

    var response = await _lambdaService.InvokeAsync(request);
    MemoryStream stream = response.Payload;
    string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
    return returnValue;
}
```

- Weitere API-Informationen finden Sie unter [Invoke](#) in der AWS SDK for .NET -API-Referenz.

Listenfunktionen

Im folgenden Codebeispiel wird veranschaulicht, wie Lambda-Funktionen aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Weitere API-Informationen finden Sie unter [ListFunctions](#) in der APIAWS SDK for .NET - Referenz für .

Funktionscode aktualisieren

Das folgende Codebeispiel veranschaulicht, wie eine Lambda-Funktion aktualisiert wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
```

```
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{
    var functionCodeRequest = new UpdateFunctionCodeRequest
    {
        FunctionName = functionName,
        Publish = true,
        S3Bucket = bucketName,
        S3Key = key,
    };

    var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
    Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
}
```

- Weitere API-Informationen finden Sie unter [UpdateFunctionCode](#) in der APIAWS SDK for .NET -Referenz für .

Funktionskonfiguration aktualisieren

Im folgenden Codebeispiel wird veranschaulicht, wie die Lambda-Funktionskonfiguration aktualisiert wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Update the code of a Lambda function.
/// </summary>
/// <param name="functionName">The name of the function to update.</param>
/// <param name="functionHandler">The code that performs the function's
actions.</param>
/// <param name="environmentVariables">A dictionary of environment variables.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> UpdateFunctionConfigurationAsync(
    string functionName,
    string functionHandler,
    Dictionary<string, string> environmentVariables)
{
    var request = new UpdateFunctionConfigurationRequest
    {
        Handler = functionHandler,
        FunctionName = functionName,
        Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
    };

    var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

    Console.WriteLine(response.LastModified);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [UpdateFunctionConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit Funktionen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine IAM-Rolle und eine Lambda-Funktion und laden Sie den Handlercode hoch.
- Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.

- Aktualisieren Sie den Funktionscode und konfigurieren Sie mit einer Umgebungsvariablen.
- Rufen Sie die Funktion mit neuen Parametern auf und erhalten Sie Ergebnisse. Zeigt das zurückgegebene Ausführungsprotokoll an.
- Listen Sie die Funktionen für Ihr Konto auf und bereinigen Sie dann die Ressourcen.

Weitere Informationen zur Verwendung von Lambda finden Sie unter [Erstellen einer Lambda-Funktion mit der Konsole](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie Methoden, die Lambda-Aktionen ausführen.

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
```

```
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //           the source code is stored.
    // S3Key    - The name of the file containing the code.
    var functionCode = new FunctionCode
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };

    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}

/// <summary>
```

```
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}

/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}

/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
```



```
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }

    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }

    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
```

```
    /// <param name="functionName">The name of the Lambda function to update.</  
param>  
    /// <param name="bucketName">The bucket where the zip file containing  
    /// the Lambda function code is stored.</param>  
    /// <param name="key">The key name of the source code file.</param>  
    /// <returns>Async Task.</returns>  
    public async Task UpdateFunctionCodeAsync(  
        string functionName,  
        string bucketName,  
        string key)  
    {  
        var functionCodeRequest = new UpdateFunctionCodeRequest  
        {  
            FunctionName = functionName,  
            Publish = true,  
            S3Bucket = bucketName,  
            S3Key = key,  
        };  
  
        var response = await  
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);  
        Console.WriteLine($"The Function was last modified at  
{response.LastModified}.");  
    }  
  
    /// <summary>  
    /// Update the code of a Lambda function.  
    /// </summary>  
    /// <param name="functionName">The name of the function to update.</param>  
    /// <param name="functionHandler">The code that performs the function's  
actions.</param>  
    /// <param name="environmentVariables">A dictionary of environment variables.</  
param>  
    /// <returns>A Boolean value indicating the success of the action.</returns>  
    public async Task<bool> UpdateFunctionConfigurationAsync(  
        string functionName,  
        string functionHandler,  
        Dictionary<string, string> environmentVariables)  
    {  
        var request = new UpdateFunctionConfigurationRequest  
        {  
            Handler = functionHandler,  
            FunctionName = functionName,  
        }  
    }  
}
```

```
        Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

Erstellen Sie eine Funktion, die das Szenario ausführt.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
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.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = 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<IAmazonLambda>()
        .AddAWSService<IAmazonIdentityManagementService>()
        .AddTransient<LambdaWrapper>()
        .AddTransient<LambdaRoleWrapper>()
        .AddTransient<UIWrapper>()
    )
    .Build();

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();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<LambdaBasics>();

var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

string functionName = configuration["FunctionName"]!;
string roleName = configuration["RoleName"]!;
string policyDocument = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Principal\": { " +
    "        \"Service\": \"lambda.amazonaws.com\" " +
    "      }, " +
    "      \"Action\": \"sts:AssumeRole\" " +
    "    } " +
    "]" +

```

```
    "});

    var incrementHandler = configuration["IncrementHandler"];
    var calculatorHandler = configuration["CalculatorHandler"];
    var bucketName = configuration["BucketName"];
    var incrementKey = configuration["IncrementKey"];
    var calculatorKey = configuration["CalculatorKey"];
    var policyArn = configuration["PolicyArn"];

    uiWrapper.DisplayLambdaBasicsOverview();

    // Create the policy to use with the AWS Lambda functions and then attach
the
    // policy to a new role.
    var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

    Console.WriteLine("Waiting for role to become active.");
    uiWrapper.WaitABit(15, "Wait until the role is active before trying to use
it.");

    // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
    var success = await lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn,
roleName);
    uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to the
role.");

    // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
    // (Amazon S3) bucket.
    uiWrapper.DisplayTitle("Create Lambda Function");
    Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
    var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
        functionName,
        bucketName,
        incrementKey,
        roleArn,
        incrementHandler);

    Console.WriteLine("Waiting for the new function to be available.");
    Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

    // Get the Lambda function.
```

```
Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
FunctionConfiguration config;
do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.State != State.Active);

Console.WriteLine($"
The function, {functionName} has been created.");
Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

uiWrapper.PressEnter();

// List the Lambda functions.
uiWrapper.DisplayTitle("Listing all Lambda functions.");
var functions = await lambdaWrapper.ListFunctionsAsync();
DisplayFunctionList(functions);

uiWrapper.DisplayTitle("Invoke increment function");
Console.WriteLine("Now that it has been created, invoke the Lambda increment
function.");
string? value;
do
{
    Console.WriteLine("Enter a value to increment: ");
    value = Console.ReadLine();
}
while (string.IsNullOrEmpty(value));

string functionParameters = "{" +
    "\"action\": \"increment\", " +
    "\"x\": \"" + value + "\"" +
    "}";
var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
Console.WriteLine($"{value} + 1 = {answer}.");

uiWrapper.DisplayTitle("Update function");
Console.WriteLine("Now update the Lambda function code.");
await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);
```

```
do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

await lambdaWrapper.UpdateFunctionConfigurationAsync(
    functionName,
    calculatorHandler,
    new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

uiWrapper.DisplayTitle("Call updated function");
Console.WriteLine("Now call the updated function...");

bool done = false;

do
{
    string? opSelected;

    Console.WriteLine("Select the operation to perform:");
    Console.WriteLine("\t1. add");
    Console.WriteLine("\t2. subtract");
    Console.WriteLine("\t3. multiply");
    Console.WriteLine("\t4. divide");
    Console.WriteLine("\t0r enter \"q\" to quit.");
    Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the operation
you want to perform: ");
    do
    {
        Console.Write("Your choice? ");
        opSelected = Console.ReadLine();
    }
    while (opSelected == string.Empty);

    var operation = (opSelected) switch
```

```
{
    "1" => "add",
    "2" => "subtract",
    "3" => "multiply",
    "4" => "divide",
    "q" => "quit",
    _ => "add",
};

if (operation == "quit")
{
    done = true;
}
else
{
    // Get two numbers and an action from the user.
    value = string.Empty;
    do
    {
        Console.Write("Enter the first value: ");
        value = Console.ReadLine();
    }
    while (value == string.Empty);

    string? value2;
    do
    {
        Console.Write("Enter a second value: ");
        value2 = Console.ReadLine();
    }
    while (value2 == string.Empty);

    functionParameters = "{" +
        "\"action\": \"" + operation + "\", " +
        "\"x\": \"" + value + "\", " +
        "\"y\": \"" + value2 + "\"" +
        "}";

    answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
    Console.WriteLine($"The answer when we {operation} the two numbers
is: {answer}.");
}
```



```
        uiWrapper.PressEnter();
    } while (!done);

    // Delete the function created earlier.

    uiWrapper.DisplayTitle("Clean up resources");
    // Detach the IAM policy from the IAM role.
    Console.WriteLine("First detach the IAM policy from the role.");
    success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
    uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached from
the role.");

    Console.WriteLine("Delete the AWS Lambda function.");
    success = await lambdaWrapper.DeleteFunctionAsync(functionName);
    if (success)
    {
        Console.WriteLine($"The {functionName} function was deleted.");
    }
    else
    {
        Console.WriteLine($"Could not remove the function {functionName}");
    }

    // Now delete the IAM role created for use with the functions
    // created by the application.
    Console.WriteLine("Now we can delete the role that we created.");
    success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
    if (success)
    {
        Console.WriteLine("The role has been successfully removed.");
    }
    else
    {
        Console.WriteLine("Couldn't delete the role.");
    }

    Console.WriteLine("The Lambda Scenario is now complete.");
    uiWrapper.PressEnter();

    // Displays a formatted list of existing functions returned by the
    // LambdaMethods.ListFunctions.
    void DisplayFunctionList(List<FunctionConfiguration> functions)
    {
```

```
        functions.ForEach(functionConfig =>
        {
            Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
        });
    }
}

namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>

```

```
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

namespace LambdaScenarioCommon;
```

```
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM)
role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the value
passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <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 einen Lambda-Handler, der eine Zahl inkrementiert.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaIncrement;

public class Function
{
    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}
```

Definieren Sie einen zweiten Lambda-Handler, der arithmetische Operationen ausführt.

```
using Amazon.Lambda.Core;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaCalculator;

public class Function
{
    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
                else

```

```
        result = x / y;
        break;
    default:
        Console.Error.WriteLine($"{action} is not a valid operation.");
        result = 0;
        break;
    }
    return result;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Aufrufen](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von .NET.


```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    }
}
```

```
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
        ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von .NET

```
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))
]
```

```
namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

```
}
```

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von .NET.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }
}
```

```
private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
context)
{
    try
    {
        context.Logger.LogInformation($"Processed record {record.Sns.Message}");

        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von .NET.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            //Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von .NET.

```
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }
    }
}
```

```
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            /* Since we are working with streams, we can return the failed item
immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
            return new StreamsEventResponse
            {
                BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                }
            };
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```



```
public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von .NET.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer),
    namespace sqsSample);

public class Function
{
```

```
public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
ILambdaContext context)
{
    List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
List<SQSBatchResponse.BatchItemFailure>();
    foreach(var message in evnt.Records)
    {
        try
        {
            //process your message
            await ProcessMessageAsync(message, context);
        }
        catch (System.Exception)
        {
            //Add failed message identifier to the batchItemFailures list
            batchItemFailures.Add(new
SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
        }
    }
    return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

MediaConvert -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden MediaConvert.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo MediaConvert

Das folgende Codebeispiel veranschaulicht die ersten Schritte mit AWS Elemental MediaConvert.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.MediaConvert;
using Amazon.MediaConvert.Model;

namespace MediaConvertActions;

public static class HelloMediaConvert
{
    static async Task Main(string[] args)
    {
        // Create the client using the default profile.
        var mediaConvertClient = new AmazonMediaConvertClient();

        Console.WriteLine($"Hello AWS Elemental MediaConvert! Your MediaConvert
Endpoints are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
```

```
// Let's get the MediaConvert endpoints.
var response = await mediaConvertClient.DescribeEndpointsAsync(
    new DescribeEndpointsRequest()
);

foreach (var endPoint in response.Endpoints)
{
    Console.WriteLine($"\\tEndPoint: {endPoint.Url}");
    Console.WriteLine();
}
}
```

- Weitere API-Informationen finden Sie unter [DescribeEndpoints](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)

Aktionen

Erstellen eines Transcodierungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen AWS Elemental MediaConvert Transcodierungsauftrag erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie den Endpunkt ab und richten Sie den Client ein.

```
// MediaConvert role Amazon Resource Name (ARN).
// For information on creating this role, see
```

```
// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
var mediaConvertRole = _configuration["mediaConvertRoleARN"];

// Include the file input and output locations in settings.json or
settings.local.json.
var fileInput = _configuration["fileInput"];
var fileOutput = _configuration["fileOutput"];

// Load the customer endpoint, if it is known.
// When you know what your Region-specific endpoint is, set it here, or set
it in your settings.local.json file.
var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

Console.WriteLine("Welcome to the MediaConvert Create Job example.");
// If you don't have the customer-specific endpoint, request it here.
if (string.IsNullOrEmpty(mediaConvertEndpoint))
{
    Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    AmazonMediaConvertClient client = new AmazonMediaConvertClient();
    DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
    DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
    mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
}
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
Console.WriteLine(new string('-', 80));
// Because you have a service URL for MediaConvert, you don't
// need to set RegionEndpoint. If you do, the ServiceURL will
// be overwritten.
AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
{
    ServiceURL = mediaConvertEndpoint,
};

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

var wrapper = new MediaConvertWrapper(mcClient);
```

```
Console.WriteLine(new string('-', 80));
```

```

    Console.WriteLine($"Creating job for input file {fileInput}.");
    var jobId = await wrapper.CreateJob(mediaConvertRole!, fileInput!,
fileOutput!);
    Console.WriteLine($"Created job with Job ID: {jobId}");
    Console.WriteLine(new string('-', 80));

```

Erstellen Sie den Auftrag mit der Wrapper-Methode und geben Sie die Auftrags-ID zurück.

```

    /// <summary>
    /// Create a job to convert a media file.
    /// </summary>
    /// <param name="mediaConvertRole">The Amazon Resource Name (ARN) of the media
convert role, as specified here:
    /// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-in-
mediaconvert-configured.html</param>
    /// <param name="fileInput">The Amazon Simple Storage Service (Amazon S3)
location of the input media file.</param>
    /// <param name="fileOutput">The Amazon S3 location for the output media file.</
param>
    /// <returns>The ID of the new job.</returns>
    public async Task<string> CreateJob(string mediaConvertRole, string fileInput,
string fileOutput)
    {
        CreateJobRequest createJobRequest = new CreateJobRequest
        {
            Role = mediaConvertRole
        };

        createJobRequest.UserMetadata.Add("Customer", "Amazon");

        JobSettings jobSettings = new JobSettings
        {
            AdAvailOffset = 0,
            TimecodeConfig = new TimecodeConfig
            {
                Source = TimecodeSource.EMBEDDED
            }
        };
        createJobRequest.Settings = jobSettings;

        #region OutputGroup

```

```
OutputGroup ofg = new OutputGroup
{
    Name = "File Group",
    OutputGroupSettings = new OutputGroupSettings
    {
        Type = OutputGroupType.FILE_GROUP_SETTINGS,
        FileGroupSettings = new FileGroupSettings
        {
            Destination = fileOutput
        }
    }
};

Output output = new Output
{
    NameModifier = "_1"
};

#region VideoDescription

VideoDescription vdes = new VideoDescription
{
    ScalingBehavior = ScalingBehavior.DEFAULT,
    TimecodeInsertion = VideoTimecodeInsertion.DISABLED,
    AntiAlias = AntiAlias.ENABLED,
    Sharpness = 50,
    AfdSignaling = AfdSignaling.NONE,
    DropFrameTimecode = DropFrameTimecode.ENABLED,
    RespondToAfd = RespondToAfd.NONE,
    ColorMetadata = ColorMetadata.INSERT,
    CodecSettings = new VideoCodecSettings
    {
        Codec = VideoCodec.H_264
    }
};
output.VideoDescription = vdes;

H264Settings h264 = new H264Settings
{
    InterlaceMode = H264InterlaceMode.PROGRESSIVE,
    NumberReferenceFrames = 3,
    Syntax = H264Syntax.DEFAULT,
    Softness = 0,
```

```
GopClosedCadence = 1,
GopSize = 90,
Slices = 1,
GopBReference = H264GopBReference.DISABLED,
SlowPal = H264SlowPal.DISABLED,
SpatialAdaptiveQuantization = H264SpatialAdaptiveQuantization.ENABLED,
TemporalAdaptiveQuantization = H264TemporalAdaptiveQuantization.ENABLED,
FlickerAdaptiveQuantization = H264FlickerAdaptiveQuantization.DISABLED,
EntropyEncoding = H264EntropyEncoding.CABAC,
Bitrate = 5000000,
FramerateControl = H264FramerateControl.SPECIFIED,
RateControlMode = H264RateControlMode.CBR,
CodecProfile = H264CodecProfile.MAIN,
Telecine = H264Telecine.NONE,
MinIInterval = 0,
AdaptiveQuantization = H264AdaptiveQuantization.HIGH,
CodecLevel = H264CodecLevel.AUTO,
FieldEncoding = H264FieldEncoding.PAFF,
SceneChangeDetect = H264SceneChangeDetect.ENABLED,
QualityTuningLevel = H264QualityTuningLevel.SINGLE_PASS,
FramerateConversionAlgorithm =
    H264FramerateConversionAlgorithm.DUPLICATE_DROP,
UnregisteredSeiTimecode = H264UnregisteredSeiTimecode.DISABLED,
GopSizeUnits = H264GopSizeUnits.FRAMES,
ParControl = H264ParControl.SPECIFIED,
NumberBFramesBetweenReferenceFrames = 2,
RepeatPps = H264RepeatPps.DISABLED,
FramerateNumerator = 30,
FramerateDenominator = 1,
ParNumerator = 1,
ParDenominator = 1
};
output.VideoDescription.CodecSettings.H264Settings = h264;

#endregion VideoDescription

#region AudioDescription

AudioDescription ades = new AudioDescription
{
    LanguageCodeControl = AudioLanguageCodeControl.FOLLOW_INPUT,
    // This name matches one specified in the following Inputs.
    AudioSourceName = "Audio Selector 1",
    CodecSettings = new AudioCodecSettings
```



```
        {
            Codec = AudioCodec.AAC
        }
    };

    AacSettings aac = new AacSettings
    {
        AudioDescriptionBroadcasterMix =
AacAudioDescriptionBroadcasterMix.NORMAL,
        RateControlMode = AacRateControlMode.CBR,
        CodecProfile = AacCodecProfile.LC,
        CodingMode = AacCodingMode.CODING_MODE_2_0,
        RawFormat = AacRawFormat.NONE,
        SampleRate = 48000,
        Specification = AacSpecification.MPEG4,
        Bitrate = 64000
    };
    ades.CodecSettings.AacSettings = aac;
    output.AudioDescriptions.Add(ades);

#endregion AudioDescription

#region Mp4 Container

    output.ContainerSettings = new ContainerSettings
    {
        Container = ContainerType.MP4
    };
    Mp4Settings mp4 = new Mp4Settings
    {
        CslgAtom = Mp4CslgAtom.INCLUDE,
        FreeSpaceBox = Mp4FreeSpaceBox.EXCLUDE,
        MoovPlacement = Mp4MoovPlacement.PROGRESSIVE_DOWNLOAD
    };
    output.ContainerSettings.Mp4Settings = mp4;

#endregion Mp4 Container

    ofg.Outputs.Add(output);
    createJobRequest.Settings.OutputGroups.Add(ofg);

#endregion OutputGroup

#region Input
```

```
Input input = new Input
{
    FilterEnable = InputFilterEnable.AUTO,
    PsiControl = InputPsiControl.USE_PSI,
    FilterStrength = 0,
    DeblockFilter = InputDeblockFilter.DISABLED,
    DenoiseFilter = InputDenoiseFilter.DISABLED,
    TimecodeSource = InputTimecodeSource.EMBEDDED,
    FileInput = fileInput
};

AudioSelector audsel = new AudioSelector
{
    Offset = 0,
    DefaultSelection = AudioDefaultSelection.NOT_DEFAULT,
    ProgramSelection = 1,
    SelectorType = AudioSelectorType.TRACK
};
audsel.Tracks.Add(1);
input.AudioSelectors.Add("Audio Selector 1", audsel);

input.VideoSelector = new VideoSelector
{
    ColorSpace = ColorSpace.FOLLOW
};

createJobRequest.Settings.Inputs.Add(input);

#endregion Input

var jobId = "";
try
{
    CreateJobResponse createJobResponse =
        await _amazonMediaConvert.CreateJobAsync(createJobRequest);

    jobId = createJobResponse.Job.Id;
}
catch (BadRequestException bre)
{
    // If the endpoint was bad.
    if (bre.Message.StartsWith("You must use the customer-"))
    {
```

```
        // The exception contains the correct endpoint; extract it.
        var mediaConvertEndpoint = bre.Message.Split('\')[1];
        Console.WriteLine(
            $"Request failed, please use endpoint {mediaConvertEndpoint}.");
    }
    else
        throw;
}

return jobId;
}
```

- Weitere API-Informationen finden Sie unter [CreateJob](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen eines Transcodierungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen AWS Elemental MediaConvert Transcodierungsauftrag abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie den Endpunkt ab und richten Sie den Client ein.

```
    // MediaConvert role Amazon Resource Name (ARN).
    // For information on creating this role, see
    // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
    var mediaConvertRole = _configuration["mediaConvertRoleARN"];

    // Include the file input and output locations in settings.json or
settings.local.json.
    var fileInput = _configuration["fileInput"];
    var fileOutput = _configuration["fileOutput"];
```

```
// Load the customer endpoint, if it is known.
// When you know what your Region-specific endpoint is, set it here, or set
it in your settings.local.json file.
var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

Console.WriteLine("Welcome to the MediaConvert Create Job example.");
// If you don't have the customer-specific endpoint, request it here.
if (string.IsNullOrEmpty(mediaConvertEndpoint))
{
    Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    AmazonMediaConvertClient client = new AmazonMediaConvertClient();
    DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
    DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
    mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
}
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
Console.WriteLine(new string('-', 80));
// Because you have a service URL for MediaConvert, you don't
// need to set RegionEndpoint. If you do, the ServiceURL will
// be overwritten.
AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
{
    ServiceURL = mediaConvertEndpoint,
};

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

var wrapper = new MediaConvertWrapper(mcClient);
```

Rufen Sie einen Auftrag anhand seiner ID ab.

```
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Getting job information for Job ID {jobId}");
var job = await wrapper.GetJobById(jobId);
Console.WriteLine($"Job {job.Id} created on {job.CreatedAt:d} has status
{job.Status}.");
Console.WriteLine(new string('-', 80));
```

```
/// <summary>
/// Get the job information for a job by its ID.
/// </summary>
/// <param name="jobId">The ID of the job.</param>
/// <returns>The Job object.</returns>
public async Task<Job> GetJobById(string jobId)
{
    var jobResponse = await _amazonMediaConvert.GetJobAsync(
        new GetJobRequest
        {
            Id = jobId
        });

    return jobResponse.Job;
}
```

- Weitere API-Informationen finden Sie unter [GetJob](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Transcodierungsaufträgen

Das folgende Codebeispiel zeigt, wie AWS Elemental MediaConvert Transcodierungsaufträge aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie den Endpunkt ab und richten Sie den Client ein.

```
// MediaConvert role Amazon Resource Name (ARN).
// For information on creating this role, see
// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
var mediaConvertRole = _configuration["mediaConvertRoleARN"];
```

```
// Include the file input and output locations in settings.json or
settings.local.json.
var fileInput = _configuration["fileInput"];
var fileOutput = _configuration["fileOutput"];

// Load the customer endpoint, if it is known.
// When you know what your Region-specific endpoint is, set it here, or set
it in your settings.local.json file.
var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

Console.WriteLine("Welcome to the MediaConvert Create Job example.");
// If you don't have the customer-specific endpoint, request it here.
if (string.IsNullOrEmpty(mediaConvertEndpoint))
{
    Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    AmazonMediaConvertClient client = new AmazonMediaConvertClient();
    DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
    DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
    mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
}
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
Console.WriteLine(new string('-', 80));
// Because you have a service URL for MediaConvert, you don't
// need to set RegionEndpoint. If you do, the ServiceURL will
// be overwritten.
AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
{
    ServiceURL = mediaConvertEndpoint,
};

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

var wrapper = new MediaConvertWrapper(mcClient);
```

Listen Sie die Aufträge mit einem bestimmten Status auf.

```
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Listing all complete jobs.");
var completeJobs = await wrapper.ListAllJobsByStatus(JobStatus.COMPLETE);
```

```

        completeJobs.ForEach(j =>
        {
            Console.WriteLine($"Job {j.Id} created on {j.CreatedAt:d} has status
{j.Status}.");
        });

```

Listen Sie die Aufträge mit einem Paginator auf.

```

/// <summary>
/// List all of the jobs with a particular status using a paginator.
/// </summary>
/// <param name="status">The status to use when listing jobs.</param>
/// <returns>The list of jobs matching the status.</returns>
public async Task<List<Job>> ListAllJobsByStatus(JobStatus? status = null)
{
    var returnedJobs = new List<Job>();

    var paginatedJobs = _amazonMediaConvert.Paginators.ListJobs(
        new ListJobsRequest
        {
            Status = status
        });

    // Get the entire list using the paginator.
    await foreach (var job in paginatedJobs.Jobs)
    {
        returnedJobs.Add(job);
    }

    return returnedJobs;
}

```

- Weitere API-Informationen finden Sie unter [ListJobs](#) in der APIAWS SDK for .NET -Referenz für

Beispiele für Organisationen mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Organizations verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Anfügen einer Richtlinie an ein Ziel

Das folgende Codebeispiel zeigt, wie Sie eine Organizations-Richtlinie an ein Ziel anfügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to attach an AWS Organizations policy to an organization,
/// an organizational unit, or an account.
/// </summary>
public class AttachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then calls the
```



```
/// AttachPolicyAsync method to attach the policy to the root
/// organization.
/// </summary>
public static async Task Main()
{
    IAmazonOrganizations client = new AmazonOrganizationsClient();
    var policyId = "p-00000000";
    var targetId = "r-0000";

    var request = new AttachPolicyRequest
    {
        PolicyId = policyId,
        TargetId = targetId,
    };

    var response = await client.AttachPolicyAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Successfully attached Policy ID {policyId} to
Target ID: {targetId}.");
    }
    else
    {
        Console.WriteLine("Was not successful in attaching the policy.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [AttachPolicy](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Richtlinie

Das folgende Codebeispiel zeigt, wie Sie eine Organizations-Richtlinie erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Creates a new AWS Organizations Policy.
/// </summary>
public class CreatePolicy
{
    /// <summary>
    /// Initializes the AWS Organizations client object, uses it to
    /// create a new Organizations Policy, and then displays information
    /// about the newly created Policy.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var policyContent = "{" +
            "  \"Version\": \"2012-10-17\"," +
            "  \"Statement\" : [{" +
                "    \"Action\" : [\"s3:*\"]," +
                "    \"Effect\" : \"Allow\"," +
                "    \"Resource\" : \"*\" +
            "  }]" +
            "}";

        try
        {
            var response = await client.CreatePolicyAsync(new
CreatePolicyRequest
            {
                Content = policyContent,
```

```
        Description = "Enables admins of attached accounts to delegate  
all Amazon S3 permissions",  
        Name = "AllowAllS3Actions",  
        Type = "SERVICE_CONTROL_POLICY",  
    });  
  
    Policy policy = response.Policy;  
    Console.WriteLine($"{policy.PolicySummary.Name} has the following  
content: {policy.Content}");  
    }  
    catch (Exception ex)  
    {  
        Console.WriteLine(ex.Message);  
    }  
    }  
}
```

- Weitere API-Informationen finden Sie unter [CreatePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines -Kontos

Das folgende Codebeispiel zeigt, wie Sie ein -Organizations-Konto erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Threading.Tasks;  
using Amazon.Organizations;  
using Amazon.Organizations.Model;  
  
/// <summary>  
/// Creates a new AWS Organizations account.  
/// </summary>
```

```
public class CreateAccount
{
    /// <summary>
    /// Initializes an Organizations client object and uses it to create
    /// the new account with the name specified in accountName.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var accountName = "ExampleAccount";
        var email = "someone@example.com";

        var request = new CreateAccountRequest
        {
            AccountName = accountName,
            Email = email,
        };

        var response = await client.CreateAccountAsync(request);
        var status = response.CreateAccountStatus;

        Console.WriteLine($"The status of {status.AccountName} is
{status.State}.");
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateAccount](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Organisation

Das folgende Codebeispiel zeigt, wie Sie eine Organisation von Organizations erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Creates an organization in AWS Organizations.
/// </summary>
public class CreateOrganization
{
    /// <summary>
    /// Creates an Organizations client object and then uses it to create
    /// a new organization with the default user as the administrator, and
    /// then displays information about the new organization.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.CreateOrganizationAsync(new
CreateOrganizationRequest
        {
            FeatureSet = "ALL",
        });

        Organization newOrg = response.Organization;

        Console.WriteLine($"Organization: {newOrg.Id} Main Account:
{newOrg.MasterAccountId}");
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateOrganization](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Organisationseinheit

Das folgende Codebeispiel zeigt, wie Sie eine Organisationseinheit von Organizations erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Creates a new organizational unit in AWS Organizations.
/// </summary>
public class CreateOrganizationalUnit
{
    /// <summary>
    /// Initializes an Organizations client object and then uses it to call
    /// the CreateOrganizationalUnit method. If the call succeeds, it
    /// displays information about the new organizational unit.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitName = "ProductDevelopmentUnit";

        var request = new CreateOrganizationalUnitRequest
        {
            Name = orgUnitName,
            ParentId = "r-0000",
        };

        var response = await client.CreateOrganizationalUnitAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully created organizational unit:
{orgUnitName}.");
        }
    }
}
```


```
        Console.WriteLine($"Organizational unit {orgUnitName} Details");
        Console.WriteLine($"ARN: {response.OrganizationalUnit.Arn} Id:
{response.OrganizationalUnit.Id}");
    }
    else
    {
        Console.WriteLine("Could not create new organizational unit.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [CreateOrganizationalUnit](#) in der APIAWS SDK for .NET -Referenz für .

Löschen Sie eine Richtlinie

Das folgende Codebeispiel zeigt, wie Sie eine Organizations-Richtlinie löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Deletes an existing AWS Organizations policy.
/// </summary>
public class DeletePolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// delete the policy with the specified policyId.
```

```
/// </summary>
public static async Task Main()
{
    // Create the client object using the default account.
    IAmazonOrganizations client = new AmazonOrganizationsClient();

    var policyId = "p-00000000";

    var request = new DeletePolicyRequest
    {
        PolicyId = policyId,
    };

    var response = await client.DeletePolicyAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Successfully deleted Policy: {policyId}.");
    }
    else
    {
        Console.WriteLine($"Could not delete Policy: {policyId}.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [DeletePolicy](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Organisation

Das folgende Codebeispiel zeigt, wie Sie eine Organizations-Organisation löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


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

/// <summary>
/// Shows how to delete an existing organization using the AWS
/// Organizations Service.
/// </summary>
public class DeleteOrganization
{
    /// <summary>
    /// Initializes the Organizations client and then calls
    /// DeleteOrganizationAsync to delete the organization.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.DeleteOrganizationAsync(new
DeleteOrganizationRequest());

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Successfully deleted organization.");
        }
        else
        {
            Console.WriteLine("Could not delete organization.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteOrganization](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Organisationseinheit

Das folgende Codebeispiel zeigt, wie Sie eine Organisationseinheit von Organizations löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to delete an existing AWS Organizations organizational unit.
/// </summary>
public class DeleteOrganizationalUnit
{
    /// <summary>
    /// Initializes the Organizations client object and calls
    /// DeleteOrganizationalUnitAsync to delete the organizational unit
    /// with the selected ID.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitId = "ou-0000-00000000";

        var request = new DeleteOrganizationalUnitRequest
        {
            OrganizationalUnitId = orgUnitId,
        };

        var response = await client.DeleteOrganizationalUnitAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted the organizational unit
with ID: {orgUnitId}.");
        }
    }
}
```

```
        else
        {
            Console.WriteLine($"Could not delete the organizational unit with
ID: {orgUnitId}.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteOrganizationalUnit](#) in der APIAWS SDK for .NET -Referenz für .

Trennen einer Richtlinie von einem Ziel

Das folgende Codebeispiel zeigt, wie Sie eine Organizations-Richtlinie von einem Ziel trennen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to detach a policy from an AWS Organizations organization,
/// organizational unit, or account.
/// </summary>
public class DetachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and uses it to call
    /// DetachPolicyAsync to detach the policy.
    /// </summary>
    public static async Task Main()
    {
```

```
// Create the client object using the default account.
IAmazonOrganizations client = new AmazonOrganizationsClient();

var policyId = "p-00000000";
var targetId = "r-0000";

var request = new DetachPolicyRequest
{
    PolicyId = policyId,
    TargetId = targetId,
};

var response = await client.DetachPolicyAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Successfully detached policy with Policy Id:
{policyId}.");
}
else
{
    Console.WriteLine("Could not detach the policy.");
}
}
```

- Weitere API-Informationen finden Sie unter [DetachPolicy](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Konten

Das folgende Codebeispiel zeigt, wie Konten für eine Organisation aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the AWS Organizations service to list the accounts associated
/// with the default account.
/// </summary>
public class ListAccounts
{
    /// <summary>
    /// Creates the Organizations client and then calls its
    /// ListAccountsAsync method.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var request = new ListAccountsRequest
        {
            MaxResults = 5,
        };

        var response = new ListAccountsResponse();
        try
        {
            do
            {
                response = await client.ListAccountsAsync(request);
                response.Accounts.ForEach(a => DisplayAccounts(a));
                if (response.NextToken is not null)
                {
                    request.NextToken = response.NextToken;
                }
            }
            while (response.NextToken is not null);
        }
        catch (AWSOrganizationsNotInUseException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

```
    }

    /// <summary>
    /// Displays information about an Organizations account.
    /// </summary>
    /// <param name="account">An Organizations account for which to display
    /// information on the console.</param>
    private static void DisplayAccounts(Account account)
    {
        string accountInfo = $"{account.Id} {account.Name}\t{account.Status}";

        Console.WriteLine(accountInfo);
    }
}
```

- Weitere API-Informationen finden Sie unter [ListAccounts](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Organisationseinheiten

Das folgende Codebeispiel zeigt, wie Organisationseinheiten von Organizations aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Lists the AWS Organizations organizational units that belong to an
/// organization.
/// </summary>
public class ListOrganizationalUnitsForParent
```

```
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// call the ListOrganizationalUnitsForParentAsync method to retrieve
    /// the list of organizational units.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var parentId = "r-0000";

        var request = new ListOrganizationalUnitsForParentRequest
        {
            ParentId = parentId,
            MaxResults = 5,
        };

        var response = new ListOrganizationalUnitsForParentResponse();
        try
        {
            do
            {
                response = await
client.ListOrganizationalUnitsForParentAsync(request);
                response.OrganizationalUnits.ForEach(u =>
DisplayOrganizationalUnit(u));
                if (response.NextToken is not null)
                {
                    request.NextToken = response.NextToken;
                }
            }
            while (response.NextToken is not null);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    /// <summary>
    /// Displays information about an Organizations organizational unit.
    /// </summary>
}
```

```
/// <param name="unit">The OrganizationalUnit for which to display
/// information.</param>
public static void DisplayOrganizationalUnit(OrganizationalUnit unit)
{
    string accountInfo = $"{unit.Id} {unit.Name}\t{unit.Arn}";

    Console.WriteLine(accountInfo);
}
}
```

- Weitere API-Informationen finden Sie unter [ListOrganizationalUnitsForParent](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Richtlinien

Das folgende Codebeispiel zeigt, wie Sie Organizations-Richtlinien auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to list the AWS Organizations policies associated with an
/// organization.
/// </summary>
public class ListPolicies
{
    /// <summary>
    /// Initializes an Organizations client object, and then calls its
    /// ListPoliciesAsync method.
    /// </summary>
```



```
public static async Task Main()
{
    // Create the client object using the default account.
    IAmazonOrganizations client = new AmazonOrganizationsClient();

    // The value for the Filter parameter is required and must be
    // one of the following:
    //     AISERVICES_OPT_OUT_POLICY
    //     BACKUP_POLICY
    //     SERVICE_CONTROL_POLICY
    //     TAG_POLICY
    var request = new ListPoliciesRequest
    {
        Filter = "SERVICE_CONTROL_POLICY",
        MaxResults = 5,
    };

    var response = new ListPoliciesResponse();
    try
    {
        do
        {
            response = await client.ListPoliciesAsync(request);
            response.Policies.ForEach(p => DisplayPolicies(p));
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        }
        while (response.NextToken is not null);
    }
    catch (AWSOrganizationsNotInUseException ex)
    {
        Console.WriteLine(ex.Message);
    }
}

/// <summary>
/// Displays information about the Organizations policies associated
/// with an organization.
/// </summary>
/// <param name="policy">An Organizations policy summary to display
/// information on the console.</param>
private static void DisplayPolicies(PolicySummary policy)
```

```
    {  
        string policyInfo = $"{policy.Id} {policy.Name}\t{policy.Description}";  
  
        Console.WriteLine(policyInfo);  
    }  
}
```

- Weitere API-Informationen finden Sie unter [ListPolicies](#) in der APIAWS SDK for .NET -Referenz für .

Amazon Polly-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Polly verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Löschen eines Lexikons

Das folgende Codebeispiel zeigt, wie Sie ein Amazon Polly-Lexikon löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Deletes an existing Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class DeleteLexicon
{
    public static async Task Main()
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();

        var success = await DeletePollyLexiconAsync(client, lexiconName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {lexiconName}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {lexiconName}.");
        }
    }

    /// <summary>
    /// Deletes the named Amazon Polly lexicon.
    /// </summary>
    /// <param name="client">The initialized Amazon Polly client object.</param>
    /// <param name="lexiconName">The name of the Amazon Polly lexicon to
    /// delete.</param>
}
```

```
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> DeletePollyLexiconAsync(
        AmazonPollyClient client,
        string lexiconName)
    {
        var deleteLexiconRequest = new DeleteLexiconRequest()
        {
            Name = lexiconName,
        };

        var response = await client.DeleteLexiconAsync(deleteLexiconRequest);


        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteLexicon](#) in der APIAWS SDK for .NET - Referenz für .

Holen Sie sich ein Lexikon

Das folgende Codebeispiel zeigt, wie Sie ein Amazon Polly-Lexikon abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Retrieves information about a specific Amazon Polly lexicon.
```

```
/// </summary>
public class GetLexicon
{
    public static async Task Main(string[] args)
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();

        await GetPollyLexiconAsync(client, lexiconName);
    }

    public static async Task GetPollyLexiconAsync(AmazonPollyClient client,
string lexiconName)
    {
        var getLexiconRequest = new GetLexiconRequest()
        {
            Name = lexiconName,
        };

        try
        {
            var response = await client.GetLexiconAsync(getLexiconRequest);
            Console.WriteLine($"Lexicon:\n Name: {response.Lexicon.Name}");
            Console.WriteLine($"Content: {response.Lexicon.Content}");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [GetLexicon](#) in der APIAWS SDK for .NET -Referenz für .

Holen Sie sich Stimmen zur Generierung

Das folgende Codebeispiel zeigt, wie Sie Amazon Polly-Stimmen für die Synthetisierung zur Verfügung stellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

public class DescribeVoices
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();

        var allVoicesRequest = new DescribeVoicesRequest();
        var enUsVoicesRequest = new DescribeVoicesRequest()
        {
            LanguageCode = "en-US",
        };

        try
        {
            string nextToken;
            do
            {
                var allVoicesResponse = await
client.DescribeVoicesAsync(allVoicesRequest);
                nextToken = allVoicesResponse.NextToken;
                allVoicesRequest.NextToken = nextToken;

                Console.WriteLine("\nAll voices: ");
                allVoicesResponse.Voices.ForEach(voice =>
                {
                    DisplayVoiceInfo(voice);
                });
            }
            while (nextToken is not null);
        }
    }
}
```

```
        do
        {
            var enUsVoicesResponse = await
client.DescribeVoicesAsync(enUsVoicesRequest);
            nextToken = enUsVoicesResponse.NextToken;
            enUsVoicesRequest.NextToken = nextToken;

            Console.WriteLine("\nen-US voices: ");
            enUsVoicesResponse.Voices.ForEach(voice =>
            {
                DisplayVoiceInfo(voice);
            });
        }
        while (nextToken is not null);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception caught: " + ex.Message);
    }
}

public static void DisplayVoiceInfo(Voice voice)
{
    Console.WriteLine($" Name: {voice.Name}\tGender:
{voice.Gender}\tLanguageName: {voice.LanguageName}");
}
}
```

- Weitere API-Informationen finden Sie unter [DescribeVoices](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Aussprachelexika

Das folgende Codebeispiel zeigt, wie Sie Amazon Polly-Aussprachelexika auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Lists the Amazon Polly lexicons that have been defined. By default,
/// lists the lexicons that are defined in the same AWS Region as the default
/// user. To view Amazon Polly lexicons that are defined in a different AWS
/// Region, supply it as a parameter to the Amazon Polly constructor.
/// </summary>
public class ListLexicons
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        var request = new ListLexiconsRequest();

        try
        {
            Console.WriteLine("All voices: ");

            do
            {
                var response = await client.ListLexiconsAsync(request);
                request.NextToken = response.NextToken;

                response.Lexicons.ForEach(lexicon =>
                {
                    var attributes = lexicon.Attributes;
                    Console.WriteLine($"Name: {lexicon.Name}");
                    Console.WriteLine($"  \tAlphabet: {attributes.Alphabet}");
                    Console.WriteLine($"  \tLanguageCode:
{attributes.LanguageCode}");
```



```
        Console.WriteLine($"\\tLastModified:
{attributes.LastModified}");
        Console.WriteLine($"\\tLexemesCount:
{attributes.LexemesCount}");
        Console.WriteLine($"\\tLexiconArn: {attributes.LexiconArn}");
        Console.WriteLine($"\\tSize: {attributes.Size}");
    });
}
while (request.NextToken is not null);
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
}
}
```

- Weitere API-Informationen finden Sie unter [ListLexicons](#) in der APIAWS SDK for .NET - Referenz für .

Speichern eines Aussprachelexikons

Das folgende Codebeispiel zeigt, wie Sie ein Aussprachelexikon von Amazon Polly speichern.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Creates a new Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
```

```
public class PutLexicon
{
    public static async Task Main()
    {
        string lexiconContent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
            "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/" +
            "pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
            "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-" +
            "lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
            "alphabet=\"ipa\" xml:lang=\"en-US\">" +
            "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>" +
            "</lexicon>";
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();
        var putLexiconRequest = new PutLexiconRequest()
        {
            Name = lexiconName,
            Content = lexiconContent,
        };

        try
        {
            var response = await client.PutLexiconAsync(putLexiconRequest);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully created Lexicon:
{lexiconName}.");
            }
            else
            {
                Console.WriteLine($"Could not create Lexicon: {lexiconName}.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Exception caught: " + ex.Message);
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [PutLexicon](#) in der APIAWS SDK for .NET -Referenz für .

Synthetisieren von Sprache aus Text

Das folgende Codebeispiel zeigt, wie Sie Sprache aus Text mit Amazon Polly synthetisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeech
{
    public static async Task Main()
    {
        string outputFileName = "speech.mp3";
        string text = "Twas brillig, and the slithy toves did gyre and gimbol in
the wabe";

        var client = new AmazonPollyClient();
        var response = await PollySynthesizeSpeech(client, text);

        WriteSpeechToStream(response.AudioStream, outputFileName);
    }

    /// <summary>
    /// Calls the Amazon Polly SynthesizeSpeechAsync method to convert text
    /// to speech.
    /// </summary>
    /// <param name="client">The Amazon Polly client object used to connect
    /// to the Amazon Polly service.</param>
    /// <param name="text">The text to convert to speech.</param>
}
```

```
    /// <returns>A SynthesizeSpeechResponse object that includes an AudioStream
    /// object with the converted text.</returns>
    private static async Task<SynthesizeSpeechResponse>
PollySynthesizeSpeech(IAmazonPolly client, string text)
    {
        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
            OutputFormat = OutputFormat.Mp3,
            VoiceId = VoiceId.Joanna,
            Text = text,
        };

        var synthesizeSpeechResponse =
            await client.SynthesizeSpeechAsync(synthesizeSpeechRequest);

        return synthesizeSpeechResponse;
    }

    /// <summary>
    /// Writes the AudioStream returned from the call to
    /// SynthesizeSpeechAsync to a file in MP3 format.
    /// </summary>
    /// <param name="audioStream">The AudioStream returned from the
    /// call to the SynthesizeSpeechAsync method.</param>
    /// <param name="outputFileName">The full path to the file in which to
    /// save the audio stream.</param>
    private static void WriteSpeechToStream(Stream audioStream, string
outputFileName)
    {
        var outputStream = new FileStream(
            outputFileName,
            FileMode.Create,
            FileAccess.Write);
        byte[] buffer = new byte[2 * 1024];
        int readBytes;

        while ((readBytes = audioStream.Read(buffer, 0, 2 * 1024)) > 0)
        {
            outputStream.Write(buffer, 0, readBytes);
        }

        // Flushes the buffer to avoid losing the last second or so of
        // the synthesized text.
        outputStream.Flush();
    }
}
```

```
        Console.WriteLine($"Saved {outputFileName} to disk.");
    }
}
```

Synthetisieren Sie Sprache aus Text mithilfe von Sprachzeichen mit Amazon Polly mithilfe eines AWS SDK.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeechMarks
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        string outputFileName = "speechMarks.json";

        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
            OutputFormat = OutputFormat.Json,
            SpeechMarkTypes = new List<string>
            {
                SpeechMarkType.Viseme,
                SpeechMarkType.Word,
            },
            VoiceId = VoiceId.Joanna,
            Text = "This is a sample text to be synthesized.",
        };

        try
        {
            using (var outputStream = new FileStream(outputFileName,
                FileMode.Create, FileAccess.Write))
            {
                var synthesizeSpeechResponse = await
                client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
                var buffer = new byte[2 * 1024];
            }
        }
    }
}
```

```
        int readBytes;

        var inputStream = synthesizeSpeechResponse.AudioStream;
        while ((readBytes = inputStream.Read(buffer, 0, 2 * 1024)) > 0)
        {
            outputStream.Write(buffer, 0, readBytes);
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
}
```

- Weitere API-Informationen finden Sie unter [SynthesizeSpeech](#) in der APIAWS SDK for .NET - Referenz für .

Amazon-RDS-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon RDS verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Amazon RDS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon RDS.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

namespace RDSActions;

public static class HelloRds
{
    static async Task Main(string[] args)
    {
        var rdsClient = new AmazonRDSClient();

        Console.WriteLine($"Hello Amazon RDS! Following are some of your DB
instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first twenty DB instances.
        var response = await rdsClient.DescribeDBInstancesAsync(
            new DescribeDBInstancesRequest()
            {
                MaxRecords = 20 // Must be between 20 and 100.
            });

        foreach (var instance in response.DBInstances)
        {
            Console.WriteLine($"DB name: {instance.DBName}");
            Console.WriteLine($"DBArn: {instance.DBInstanceArn}");
            Console.WriteLine($"DBIdentifier: {instance.DBInstanceIdentifier}");
            Console.WriteLine();
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBInstances](#) in der API-Referenz zu AWS SDK for .NET .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen einer DB-Instance.

Das folgende Codebeispiel zeigt, wie Sie eine Amazon RDS-DB-Instance erstellen und warten, bis sie verfügbar ist.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an 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="dbName">Name for the DB instance.</param>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <param name="parameterGroupName">DB parameter group to associate with the
instance.</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>
/// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
```



```
/// <param name="adminName">Admin user name.</param>
/// <param name="adminPassword">Admin user password.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
    string parameterGroupName, string dbEngine, string dbEngineVersion,
    string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
{
    var response = await _amazonRDS.CreateDBInstanceAsync(
        new CreateDBInstanceRequest()
        {
            DBName = dbName,
            DBInstanceIdentifier = dbInstanceIdentifier,
            DBParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            DBInstanceClass = instanceClass,
            AllocatedStorage = allocatedStorage,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword
        });

    return response.DBInstance;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBInstance](#) in der AWS SDK for .NET -API-Referenz.

Erstellen einer DB-Parametergruppe

Das folgende Codebeispiel zeigt, wie Sie eine Amazon RDS-DB-Parametergruppe erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
/// to determine when the DB parameter group is ready to use.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <param name="family">Family of the DB parameter group.</param>
/// <param name="description">Description of the DB parameter group.</param>
/// <returns>The new DB parameter group.</returns>
public async Task<DBParameterGroup> CreateDBParameterGroup(
    string name, string family, string description)
{
    var response = await _amazonRDS.CreateDBParameterGroupAsync(
        new CreateDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
            DBParameterGroupFamily = family,
            Description = description
        });
    return response.DBParameterGroup;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

So erstellen Sie einen Snapshot einer DB-Instance

Das folgende Codebeispiel zeigt, wie Sie einen Snapshot einer Amazon RDS-DB-Instance erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a snapshot of a DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
/// <returns>DB snapshot object.</returns>
public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
{
    var response = await _amazonRDS.CreateDBSnapshotAsync(
        new CreateDBSnapshotRequest()
        {
            DBSnapshotIdentifier = snapshotIdentifier,
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    return response.DBSnapshot;
}
```

- Weitere API-Informationen finden Sie unter [CreateDBSnapshot](#) in der AWS SDK for .NET -API-Referenz.

Löschen einer DB-Instance

Das folgende Codebeispiel zeigt, wie Sie eine Amazon RDS-DB-Instance löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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> DeleteDBInstance(string dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}

```

- Weitere API-Informationen finden Sie unter [DeleteDBInstance](#) in der API-Referenz zu AWS SDK for .NET .

Löschen einer Parametergruppe

Das folgende Codebeispiel zeigt, wie Sie eine Amazon RDS-DB-Parametergruppe löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Delete a DB parameter group. The group cannot be a default DB parameter
group
/// or be associated with any DB instances.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteDBParameterGroup(string name)
{

```

```
var response = await _amazonRDS.DeleteDBParameterGroupAsync(  
    new DeleteDBParameterGroupRequest()  
    {  
        DBParameterGroupName = name,  
    });  
return response.HttpStatusCode == HttpStatusCode.OK;  
}
```

- Weitere API-Informationen finden Sie unter [DeleteDBParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von DB-Instances

Das folgende Codebeispiel zeigt, wie Sie Amazon-RDS-DB-Instances beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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>> DescribeDBInstances(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;
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBInstances](#) in der API-Referenz zu AWS SDK for .NET .

Beschreiben von DB-Parametergruppen

Das folgende Codebeispiel zeigt, wie Sie Amazon-RDS-DB-Parametergruppen beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get descriptions of DB parameter groups.
/// </summary>
/// <param name="name">Optional name of the DB parameter group to describe.</
param>
/// <returns>The list of DB parameter group descriptions.</returns>
public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
= null)
{
    var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
        new DescribeDBParameterGroupsRequest()
        {
            DBParameterGroupName = name
        });
    return response.DBParameterGroups;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBParameterGroups](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben der Datenbank-Engine-Versionen

Das folgende Codebeispiel zeigt, wie Sie Versionen der Amazon-RDS-Datenbank-Engine beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of DB engine versions for a particular DB engine.
/// </summary>
/// <param name="engine">Name of the engine.</param>
/// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
/// <returns>List of DBEngineVersions.</returns>
public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
    string dbParameterGroupFamily = null)
{
    var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
        new DescribeDBEngineVersionsRequest()
        {
            Engine = engine,
            DBParameterGroupFamily = dbParameterGroupFamily
        });
    return response.DBEngineVersions;
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBEngineVersions](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben der Optionen für DB-Instances

Das folgende Codebeispiel zeigt, wie Optionen für Amazon RDS-DB-Instances beschrieben werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) 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>>
DescribeOrderableDBInstanceOptions(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;
}
```



```
}

```

- Weitere API-Informationen finden Sie unter [DescribeOrderableDBInstanceOptions](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von Parametern in einer DB-Parametergruppe

Das folgende Codebeispiel zeigt, wie Parameter in einer Amazon-RDS-DB-Parametergruppe beschrieben werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {

```

```
        results.Add(parameters);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBParameters](#) in der API-Referenz zu AWS SDK for .NET .

Beschreiben von Snapshots von DB-Instances

Das folgende Codebeispiel zeigt, wie Sie Snapshots von Amazon-RDS-DB-Instances beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Return a list of DB snapshots for a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>List of DB snapshots.</returns>
public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
{
    var results = new List<DBSnapshot>();
    var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
        new DescribeDBSnapshotsRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    // Get the entire list using the paginator.
    await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
    {
        results.Add(snapshots);
    }
}
```

```
    }  
    return results;  
}
```

- Weitere API-Informationen finden Sie unter [DescribeDBSnapshots](#) in der API-Referenz zu AWS SDK for .NET .

Aktualisieren von Parametern in einer DB-Parametergruppe

Das folgende Codebeispiel zeigt, wie Parameter in einer Amazon RDS-DB-Parametergruppe aktualisiert werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync  
/// to determine when the DB parameter group is ready to use.  
/// </summary>  
/// <param name="name">Name of the DB parameter group.</param>  
/// <param name="parameters">List of parameters. Maximum of 20 per request.</  
param>  
/// <returns>The updated DB parameter group name.</returns>  
public async Task<string> ModifyDBParameterGroup(  
    string name, List<Parameter> parameters)  
{  
    var response = await _amazonRDS.ModifyDBParameterGroupAsync(  
        new ModifyDBParameterGroupRequest()  
        {  
            DBParameterGroupName = name,  
            Parameters = parameters,  
        });  
    return response.DBParameterGroupName;
```

```
}
```

- Weitere API-Informationen finden Sie unter [ModifyDBParameterGroup](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit DB-Instances

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine benutzerdefinierte DB-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie eine DB-Instance, die zur Verwendung der Parametergruppe konfiguriert ist. Die DB-Instance enthält auch eine Datenbank.
- Erstellen Sie einen Snapshot der Instance.
- Löschen Sie die Instance und die Parametergruppe.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.
```

This .NET example performs the following tasks:

1. Returns a list of the available DB engine families using the `DescribeDBEngineVersionsAsync` method.
 2. Selects an engine family and creates a custom DB parameter group using the `CreateDBParameterGroupAsync` method.
 3. Gets the parameter groups using the `DescribeDBParameterGroupsAsync` method.
 4. Gets parameters in the group using the `DescribeDBParameters` method.
 5. Parses and displays parameters in the group.
 6. Modifies both the `auto_increment_offset` and `auto_increment_increment` parameters using the `ModifyDBParameterGroupAsync` method.
 7. Gets and displays the updated parameters using the `DescribeDBParameters` method with a source of "user".
 8. Gets a list of allowed engine versions using the `DescribeDBEngineVersionsAsync` method.
 9. Displays and selects from a list of micro instance classes available for the selected engine and version.
 10. Creates an RDS DB instance that contains a MySQL database and uses the parameter group using the `CreateDBInstanceAsync` method.
 11. Waits for DB instance to be ready using the `DescribeDBInstancesAsync` method.
 12. Prints out the connection endpoint string for the new DB instance.
 13. Creates a snapshot of the DB instance using the `CreateDBSnapshotAsync` method.
 14. Waits for DB snapshot to be ready using the `DescribeDBSnapshots` method.
 15. Deletes the DB instance using the `DeleteDBInstanceAsync` method.
 16. Waits for DB instance to be deleted using the `DescribeDbInstances` method.
 17. Deletes the parameter group using the `DeleteDBParameterGroupAsync`.
- */

```
private static readonly string sepBar = new('-', 80);
private static RDSWrapper rdsWrapper = null!;
private static ILogger logger = null!;
private static readonly string engine = "mysql";
static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon RDS 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<IAmazonRDS>()
            .AddTransient<RDSWrapper>()
    )
    .Build();

logger = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
}).CreateLogger<RDSInstanceScenario>();

rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

Console.WriteLine(sepBar);
Console.WriteLine(
    "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
Console.WriteLine(sepBar);

try
{
    var parameterGroupFamily = await ChooseParameterGroupFamily();

    var parameterGroup = await CreateDbParameterGroup(parameterGroupFamily);

    var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
        new List<string> { "auto_increment_offset",
"auto_increment_increment" });

    await ModifyParameters(parameterGroup.DBParameterGroupName, parameters);

    await DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

    var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

    var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

    var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

    var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
        instanceChoice.DBInstanceClass, newInstanceIdentifier);
```

```
        if (newInstance != null)
        {
            DisplayConnectionString(newInstance);

            await CreateSnapshot(newInstance);

            await DeleteRdsInstance(newInstance);
        }

        await DeleteParameterGroup(parameterGroup);

        Console.WriteLine("Scenario complete.");
        Console.WriteLine(sepBar);
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
    }
}

/// <summary>
/// Choose the RDS DB parameter group family from a list of available options.
/// </summary>
/// <returns>The selected parameter group family.</returns>
public static async Task<string> ChooseParameterGroupFamily()
{
    Console.WriteLine(sepBar);
    // 1. Get a list of available engines.
    var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

    Console.WriteLine("1. The following is a list of available DB parameter
group families:");
    int i = 1;
    var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
    foreach (var parameterGroupFamily in parameterGroupFamilies)
    {
        // List the available parameter group families.
        Console.WriteLine(
            $"{i}\t{i}. Family: {parameterGroupFamily.Key}");
        i++;
    }

    var choiceNumber = 0;
```

```
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            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<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">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>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"{p.ParameterName}." +
                $"{p.Description}." +
                $"{p.AllowedValues}." +
                $"{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 ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
```

```
        if (p.IsModifiable && p.DataType == "integer")
        {
            int newValue = 0;
            while (newValue == 0)
            {
                Console.WriteLine(
                    $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                var choice = Console.ReadLine();
                Int32.TryParse(choice, out newValue);
            }

            p.ParameterValue = newValue.ToString();
        }
    }

    await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

    Console.WriteLine(sepBar);
}

/// <summary>
/// Describe the user source parameters in the group.
/// </summary>
/// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
/// <returns>Async task.</returns>
public static async Task DescribeUserSourceParameters(string parameterGroupName)
{
    Console.WriteLine(sepBar);
    Console.WriteLine("7. Describe user source parameters in the group.");

    var parameters =
        await rdsWrapper.DescribeDBParameters(parameterGroupName, "user");

    parameters.ForEach(p =>
        Console.WriteLine(
            $"{p.ParameterName}: {p.ParameterName}." +
            $"{p.Description}: {p.Description}." +
            $"{p.AllowedValues}: {p.AllowedValues}." +
            $"{p.ParameterValue}: {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> ChooseDbEngineVersion(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"{i}. Engine: {version.Engine} Version
{version.EngineVersion}.");
            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>
```

```
/// 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 rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
engineVersion);

    Console.WriteLine($"8. Available micro DB instance classes for engine
{engine} and version {engineVersion}:");
    int i = 1;

    // Filter to micro instances for this example.
    allowedInstances = allowedInstances
        .Where(i => i.DBInstanceClass.Contains("micro")).ToList();

    foreach (var instance in allowedInstances)
    {
        Console.WriteLine(
            $"{i}\t{i}. Instance class: {instance.DBInstanceClass} (storage type
{instance.StorageType})");
        i++;
    }

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
    {
        Console.WriteLine("9. Select an available DB instance class by entering
a number from the list above:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }

    var instanceChoice = allowedInstances[choiceNumber - 1];
    Console.WriteLine(sepBar);
    return instanceChoice;
}
```

```
    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB instance.</
param>
    /// <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?> CreateRdsNewInstance(DBParameterGroup
parameterGroup,
        string engineName, string engineVersion, string instanceClass, string
instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"10. Create a new DB instance with identifier
{instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await rdsWrapper.DescribeDBInstances();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {
            Console.WriteLine("Please enter an admin user name:");
            var username = Console.ReadLine();

            Console.WriteLine("Please enter an admin password:");
            var password = Console.ReadLine();

            newInstance = await rdsWrapper.CreateDBInstance(
```

```

        "ExampleInstance",
        instanceIdentifier,
        parameterGroup.DBParameterGroupName,
        engineName,
        engineVersion,
        instanceClass,
        20,
        username,
        password
    );

    // 11. Wait for the DB instance to be ready.

    Console.WriteLine("11. Waiting for DB instance to be ready...");
    while (!isInstanceReady)
    {
        instances = await
rdsWrapper.DescribeDBInstances(instanceIdentifier);
        isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
        newInstance = instances.First();
        Thread.Sleep(30000);
    }
}

Console.WriteLine(sepBar);
return newInstance;
}

/// <summary>
/// Display a connection string for an RDS DB instance.
/// </summary>
/// <param name="instance">The DB instance to use to get a connection string.</
param>
public static void DisplayConnectionString(DBInstance instance)
{
    Console.WriteLine(sepBar);
    // Display the connection string.
    Console.WriteLine("12. New DB instance connection string: ");
    Console.WriteLine(
        $"{Environment.NewLine}{instance.Engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
"
        + $"{Environment.NewLine}-u {instance.MasterUsername} -p [YOUR PASSWORD]{Environment.NewLine}");
}

```

```
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an RDS DB instance.
    /// </summary>
    /// <param name="instance">DB instance to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"13. Creating snapshot from DB instance
{instance.DBInstanceIdentifier}.");
        var snapshot = await
rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            var snapshots = await
rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
```

```
{
    Console.WriteLine(sepBar);
    // Delete the DB instance.
    Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
    await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);

    // Wait for the DB instance to delete.
    Console.WriteLine($"16. Waiting for the DB instance to delete...");
    bool isInstanceDeleted = false;

    while (!isInstanceDeleted)
    {
        var instance = await rdsWrapper.DescribeDBInstances();
        isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
newInstance.DBInstanceIdentifier);
        Thread.Sleep(30000);
    }

    Console.WriteLine("DB instance deleted.");
    Console.WriteLine(sepBar);
}

/// <summary>
/// Delete a DB parameter group.
/// </summary>
/// <param name="parameterGroup">The parameter group to delete.</param>
/// <returns>Async task.</returns>
public static async Task DeleteParameterGroup(DBParameterGroup parameterGroup)
{
    Console.WriteLine(sepBar);
    // Delete the parameter group.
    Console.WriteLine($"17. Delete the DB parameter group
{parameterGroup.DBParameterGroupName}.");
    await
rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

    Console.WriteLine(sepBar);
}
```

Wrapper-Methoden, die vom Szenario für DB-Instance-Aktionen verwendet werden.


```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with DB
/// instance operations.
/// </summary>
public partial class RDSWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public RDSWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <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>>
DescribeOrderableDBInstanceOptions(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>
/// 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>> DescribeDBInstances(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>
/// Create an 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="dbName">Name for the DB instance.</param>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <param name="parameterGroupName">DB parameter group to associate with the
instance.</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>
/// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
/// <param name="adminName">Admin user name.</param>
/// <param name="adminPassword">Admin user password.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
    string parameterGroupName, string dbEngine, string dbEngineVersion,
    string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
{
    var response = await _amazonRDS.CreateDBInstanceAsync(
        new CreateDBInstanceRequest()
        {
            DBName = dbName,
            DBInstanceIdentifier = dbInstanceIdentifier,
            DBParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            DBInstanceClass = instanceClass,
            AllocatedStorage = allocatedStorage,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword
        });

    return response.DBInstance;
}
```

```

/// <summary>
/// Delete a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}

```

Wrapper-Methoden, die vom Szenario für DB-Parametergruppen verwendet werden.

```

/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
/// parameter groups.
/// </summary>
public partial class RDSWrapper
{
    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
= null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()

```

```
        {
            DBParameterGroupName = name
        });
    return response.DBParameterGroups;
}

/// <summary>
/// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
/// to determine when the DB parameter group is ready to use.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <param name="family">Family of the DB parameter group.</param>
/// <param name="description">Description of the DB parameter group.</param>
/// <returns>The new DB parameter group.</returns>
public async Task<DBParameterGroup> CreateDBParameterGroup(
    string name, string family, string description)
{
    var response = await _amazonRDS.CreateDBParameterGroupAsync(
        new CreateDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
            DBParameterGroupFamily = family,
            Description = description
        });
    return response.DBParameterGroup;
}

/// <summary>
/// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
/// to determine when the DB parameter group is ready to use.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
/// <returns>The updated DB parameter group name.</returns>
public async Task<string> ModifyDBParameterGroup(
    string name, List<Parameter> parameters)
{
    var response = await _amazonRDS.ModifyDBParameterGroupAsync(
```

```
        new ModifyDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
            Parameters = parameters,
        });
    return response.DBParameterGroupName;
}

/// <summary>
/// Delete a DB parameter group. The group cannot be a default DB parameter
group
/// or be associated with any DB instances.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteDBParameterGroup(string name)
{
    var response = await _amazonRDS.DeleteDBParameterGroupAsync(
        new DeleteDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get a list of DB parameters from a specific parameter group.
/// </summary>
/// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
/// <param name="source">Optional source for selecting parameters.</param>
/// <returns>List of parameter values.</returns>
public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
{
    var results = new List<Parameter>();
    var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
        new DescribeDBParametersRequest()
        {
            DBParameterGroupName = dbParameterGroupName,
```

```
        Source = source
    });
    // Get the entire list using the paginator.
    await foreach (var parameters in paginateParameters.Parameters)
    {
        results.Add(parameters);
    }
    return results;
}
```

Wrapper-Methoden, die vom Szenario für DB-Snapshot-Aktionen verwendet werden.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
/// snapshots.
/// </summary>
public partial class RDSWrapper
{
    /// <summary>
    /// Create a snapshot of a DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        return response.DBSnapshot;
    }
}
```

```
/// <summary>
/// Return a list of DB snapshots for a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>List of DB snapshots.</returns>
public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
{
    var results = new List<DBSnapshot>();
    var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
        new DescribeDBSnapshotsRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    // Get the entire list using the paginator.
    await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
    {
        results.Add(snapshots);
    }
    return results;
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateDBInstance](#)
 - [CreateDBParameterGroup](#)
 - [CreateDBSnapshot](#)
 - [DeleteDBInstance](#)
 - [DeleteDBParameterGroup](#)
 - [DescribeDBEngineVersions](#)
 - [DescribeDBInstances](#)
 - [DescribeDBParameterGroups](#)
 - [DescribeDBParameters](#)
 - [DescribeDBSnapshots](#)
 - [DescribeOrderableDBInstanceOptions](#)
 - [ModifyDBParameterGroup](#)

Amazon Rekognition-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Rekognition verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Vergleichen Sie Gesichter in einem Bild mit einem Referenzbild

Das folgende Codebeispiel zeigt, wie Gesichter in einem Bild mit einem Referenzbild mit Amazon Rekognition verglichen werden.

Weitere Informationen finden Sie unter [Vergleich von Gesichtern in Bildern](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
/// <summary>
/// Uses the Amazon Rekognition Service to compare faces in two images.
/// </summary>
public class CompareFaces
{
    public static async Task Main()
    {
        float similarityThreshold = 70F;
        string sourceImage = "source.jpg";
        string targetImage = "target.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

        try
        {
            using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
            byte[] data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
            imageSource.Bytes = new MemoryStream(data);
        }
        catch (Exception)
        {
            Console.WriteLine($"Failed to load source image: {sourceImage}");
            return;
        }

        Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

        try
        {
            using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
            byte[] data = new byte[fs.Length];
            data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
            imageTarget.Bytes = new MemoryStream(data);
        }
        catch (Exception ex)
        {
```

```
        Console.WriteLine($"Failed to load target image: {targetImage}");
        Console.WriteLine(ex.Message);
        return;
    }

    var compareFacesRequest = new CompareFacesRequest
    {
        SourceImage = imageSource,
        TargetImage = imageTarget,
        SimilarityThreshold = similarityThreshold,
    };

    // Call operation
    var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

    // Display results
    compareFacesResponse.FaceMatches.ForEach(match =>
    {
        ComparedFace face = match.Face;
        BoundingBox position = face.BoundingBox;
        Console.WriteLine($"Face at {position.Left} {position.Top} matches
with {match.Similarity}% confidence.");
    });

    Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
face(s) that did not match.");
    }
}
```

- Weitere API-Informationen finden Sie unter [CompareFaces](#) in der APIAWS SDK for .NET - Referenz für .

Eine Sammlung erstellen

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Rekognition-Sammlung erstellen.

Weitere Informationen finden Sie unter [Erstellen einer Sammlung](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses Amazon Rekognition to create a collection to which you can add
/// faces using the IndexFaces operation.
/// </summary>
public class CreateCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Creating collection: " + collectionId);

        var createCollectionRequest = new CreateCollectionRequest
        {
            CollectionId = collectionId,
        };

        CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
        Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
        Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
    }
}
```


- Weitere API-Informationen finden Sie unter [CreateCollection](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Sammlung.

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Rekognition-Sammlung löschen.

Weitere Informationen finden Sie unter [Löschen einer Sammlung](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon Rekognition Service to delete an existing collection.
/// </summary>
public class DeleteCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Deleting collection: " + collectionId);

        var deleteCollectionRequest = new DeleteCollectionRequest()
        {
            CollectionId = collectionId,
        };

        var deleteCollectionResponse = await
rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
```

```
        Console.WriteLine($"{collectionId}:  
        {deleteCollectionResponse.StatusCode}");  
    }  
}
```

- Weitere API-Informationen finden Sie unter [DeleteCollection](#) in der APIAWS SDK for .NET - Referenz für .

Löschen von Gesichtern aus einer Sammlung

Das folgende Codebeispiel zeigt, wie Gesichter aus einer Amazon Rekognition-Sammlung gelöscht werden.

Weitere Informationen finden Sie unter [Löschen von Gesichtern aus einer Sammlung](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Amazon.Rekognition;  
using Amazon.Rekognition.Model;  
  
/// <summary>  
/// Uses the Amazon Rekognition Service to delete one or more faces from  
/// a Rekognition collection.  
/// </summary>  
public class DeleteFaces  
{  
    public static async Task Main()  
    {  
        string collectionId = "MyCollection";  
        var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx" };  
    }  
}
```

```
var rekognitionClient = new AmazonRekognitionClient();

var deleteFacesRequest = new DeleteFacesRequest()
{
    CollectionId = collectionId,
    FaceIds = faces,
};

DeleteFacesResponse deleteFacesResponse = await
rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
deleteFacesResponse.DeletedFaces.ForEach(face =>
{
    Console.WriteLine($"FaceID: {face}");
});
}
```

- Weitere API-Informationen finden Sie unter [DeleteFaces](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben einer Sammlung

Das folgende Codebeispiel zeigt, wie Sie eine Amazon Rekognition-Sammlung beschreiben.

Weitere Informationen finden Sie unter [Beschreiben einer Sammlung](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
/// collection.
/// </summary>
public class DescribeCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine($"Describing collection: {collectionId}");

        var describeCollectionRequest = new DescribeCollectionRequest()
        {
            CollectionId = collectionId,
        };

        var describeCollectionResponse = await
rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
        Console.WriteLine($"Collection ARN:
{describeCollectionResponse.CollectionARN}");
        Console.WriteLine($"Face count:
{describeCollectionResponse.FaceCount}");
        Console.WriteLine($"Face model version:
{describeCollectionResponse.FaceModelVersion}");
        Console.WriteLine($"Created:
{describeCollectionResponse.CreationTimestamp}");
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeCollection](#) in der APIAWS SDK for .NET - Referenz für .

Gesichter in einem Bild erkennen

Das folgende Codebeispiel zeigt, wie Sie Gesichter in einem Bild mit Amazon Rekognition erkennen.

Weitere Informationen finden Sie unter [Erkennen von Gesichtern in einem Bild](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectFaces
{
    public static async Task Main()
    {
        string photo = "input.jpg";
        string bucket = "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectFacesRequest = new DetectFacesRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },

            // Attributes can be "ALL" or "DEFAULT".
            // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
            // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
            items/Rekognition/TFaceDetail.html
        };
    }
}
```

```
        Attributes = new List<string>() { "ALL" },
    };

    try
    {
        DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
        bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
        foreach (FaceDetail face in detectFacesResponse.FaceDetails)
        {
            Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
            Console.WriteLine($"Confidence: {face.Confidence}");
            Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
            Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
            Console.WriteLine($"Brightness:
{face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

            if (hasAll)
            {
                Console.WriteLine($"Estimated age is between
{face.AgeRange.Low} and {face.AgeRange.High} years old.");
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Zeigt Informationen zum Begrenzungsrahmen für alle Gesichter in einem Bild an.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;
```

```
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to display the details of the
/// bounding boxes around the faces detected in an image.
/// </summary>
public class ImageOrientationBoundingBox
{
    public static async Task Main()
    {
        string photo = @"D:\Development\AWS-Examples\Rekognition\target.jpg"; //
"photo.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Amazon.Rekognition.Model.Image();
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
            byte[] data = null;
            data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
            image.Bytes = new MemoryStream(data);
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        int height;
        int width;

        // Used to extract original photo width/height
        using (var imageBitmap = new Bitmap(photo))
        {
            height = imageBitmap.Height;
            width = imageBitmap.Width;
        }

        Console.WriteLine("Image Information:");
        Console.WriteLine(photo);
    }
}
```

```
Console.WriteLine("Image Height: " + height);
Console.WriteLine("Image Width: " + width);

try
{
    var detectFacesRequest = new DetectFacesRequest()
    {
        Image = image,
        Attributes = new List<string>() { "ALL" },
    };

    DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
    detectFacesResponse.FaceDetails.ForEach(face =>
    {
        Console.WriteLine("Face:");
        ShowBoundingBoxPositions(
            height,
            width,
            face.BoundingBox,
            detectFacesResponse.OrientationCorrection);

        Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
        Console.WriteLine($"The detected face is estimated to be between
{face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
    });
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

/// <summary>
/// Display the bounding box information for an image.
/// </summary>
/// <param name="imageHeight">The height of the image.</param>
/// <param name="imageWidth">The width of the image.</param>
/// <param name="box">The bounding box for a face found within the image.</
param>
/// <param name="rotation">The rotation of the face's bounding box.</param>
```

```
public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
BoundingBox box, string rotation)
{
    float left;
    float top;

    if (rotation == null)
    {
        Console.WriteLine("No estimated orientation. Check Exif data.");
        return;
    }

    // Calculate face position based on image orientation.
    switch (rotation)
    {
        case "ROTATE_0":
            left = imageWidth * box.Left;
            top = imageHeight * box.Top;
            break;
        case "ROTATE_90":
            left = imageHeight * (1 - (box.Top + box.Height));
            top = imageWidth * box.Left;
            break;
        case "ROTATE_180":
            left = imageWidth - (imageWidth * (box.Left + box.Width));
            top = imageHeight * (1 - (box.Top + box.Height));
            break;
        case "ROTATE_270":
            left = imageHeight * box.Top;
            top = imageWidth * (1 - box.Left - box.Width);
            break;
        default:
            Console.WriteLine("No estimated orientation information. Check
Exif data.");
            return;
    }

    // Display face location information.
    Console.WriteLine($"Left: {left}");
    Console.WriteLine($"Top: {top}");
    Console.WriteLine($"Face Width: {imageWidth * box.Width}");
    Console.WriteLine($"Face Height: {imageHeight * box.Height}");
}
}
```

- Weitere API-Informationen finden Sie unter [DetectFaces](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen von Labels in einem Bild

Das folgende Codebeispiel zeigt, wie Sie Labels in einem Bild mit Amazon Rekognition erkennen.

Weitere Informationen finden Sie unter [Erkennen von Labels in einem Bild](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectLabels
{
    public static async Task Main()
    {
        string photo = "del_river_02092020_01.jpg"; // "input.jpg";
        string bucket = "igsmiths3photos"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectlabelsRequest = new DetectLabelsRequest
        {
            Image = new Image()
```

```
        {
            S3Object = new S3Object()
            {
                Name = photo,
                Bucket = bucket,
            },
        },
        MaxLabels = 10,
        MinConfidence = 75F,
    };

    try
    {
        DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectLabelsRequest);
        Console.WriteLine("Detected labels for " + photo);
        foreach (Label label in detectLabelsResponse.Labels)
        {
            Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Erkennen Sie Labels in einer Bilddatei, die auf Ihrem Computer gespeichert ist.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored locally.
/// </summary>
```

```
public class DetectLabelsLocalFile
{
    public static async Task Main()
    {
        string photo = "input.jpg";

        var image = new Amazon.Rekognition.Model.Image();
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
            byte[] data = null;
            data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
            image.Bytes = new MemoryStream(data);
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        var rekognitionClient = new AmazonRekognitionClient();

        var detectLabelsRequest = new DetectLabelsRequest
        {
            Image = image,
            MaxLabels = 10,
            MinConfidence = 77F,
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectLabelsRequest);
            Console.WriteLine($"Detected labels for {photo}");
            foreach (Label label in detectLabelsResponse.Labels)
            {
                Console.WriteLine($"{label.Name}: {label.Confidence}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```



```
    }  
  }  
}
```

- Weitere API-Informationen finden Sie unter [DetectLabels](#) in der APIAWS SDK for .NET - Referenz für .

Erkennen von Labels zur Moderation in einem Bild

Das folgende Codebeispiel zeigt, wie Sie Moderationslabels in einem Bild mit Amazon Rekognition erkennen. Moderationslabels kennzeichnen Inhalte, die für einige Zielgruppen möglicherweise unangemessen sind.

Weitere Informationen finden Sie unter [Erkennen von unangemessenen Bildern](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Threading.Tasks;  
using Amazon.Rekognition;  
using Amazon.Rekognition.Model;  
  
/// <summary>  
/// Uses the Amazon Rekognition Service to detect unsafe content in a  
/// JPEG or PNG format image.  
/// </summary>  
public class DetectModerationLabels  
{  
    public static async Task Main(string[] args)  
    {  
        string photo = "input.jpg";  
        string bucket = "bucket";  
    }  
}
```

```
var rekognitionClient = new AmazonRekognitionClient();

var detectModerationLabelsRequest = new DetectModerationLabelsRequest()
{
    Image = new Image()
    {
        S3Object = new S3Object()
        {
            Name = photo,
            Bucket = bucket,
        },
    },
    MinConfidence = 60F,
};

try
{
    var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
    Console.WriteLine("Detected labels for " + photo);
    foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
    {
        Console.WriteLine($"Label: {label.Name}");
        Console.WriteLine($"Confidence: {label.Confidence}");
        Console.WriteLine($"Parent: {label.ParentName}");
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
```


- Weitere API-Informationen finden Sie unter [DetectModerationLabels](#) in der APIAWS SDK for .NET -Referenz für .

Erkennen von Text in einem Bild

Das folgende Codebeispiel zeigt, wie Sie Text in einem Bild mit Amazon Rekognition erkennen.

Weitere Informationen finden Sie unter [Erkennen von Text in einem Bild](#).

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon Rekognition Service to detect text in an image. The
/// example was created using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class DetectText
{
    public static async Task Main()
    {
        string photo = "Dad_photographer.jpg"; // "input.jpg";
        string bucket = "igsmiths3photos"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
        };

        try
```

```
    {
        DetectTextResponse detectTextResponse = await
rekognitionClient.DetectTextAsync(detectTextRequest);
        Console.WriteLine($"Detected lines and words for {photo}");
        detectTextResponse.TextDetections.ForEach(text =>
        {
            Console.WriteLine($"Detected: {text.DetectedText}");
            Console.WriteLine($"Confidence: {text.Confidence}");
            Console.WriteLine($"Id : {text.Id}");
            Console.WriteLine($"Parent Id: {text.ParentId}");
            Console.WriteLine($"Type: {text.Type}");
        });
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
}
```

- Weitere API-Informationen finden Sie unter [DetectText](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Informationen über Prominente

Im folgenden Codebeispiel wird gezeigt, wie Informationen über Prominente mithilfe von Amazon Rekognition abgerufen werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
```

```
using Amazon.Rekognition.Model;

/// <summary>
/// Shows how to use Amazon Rekognition to retrieve information about the
/// celebrity identified by the supplied celebrity Id.
/// </summary>
public class CelebrityInfo
{
    public static async Task Main()
    {
        string celebId = "nnnnnnnn";

        var rekognitionClient = new AmazonRekognitionClient();

        var celebrityInfoRequest = new GetCelebrityInfoRequest
        {
            Id = celebId,
        };

        Console.WriteLine($"Getting information for celebrity: {celebId}");

        var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

        // Display celebrity information.
        Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
        Console.WriteLine("Further information (if available):");
        celebrityInfoResponse.Urls.ForEach(url =>
        {
            Console.WriteLine(url);
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [GetCelebrityInfo](#) in der APIAWS SDK for .NET - Referenz für .

Gesichter einer Sammlung zuordnen

Das folgende Codebeispiel zeigt, wie Gesichter in einem Bild indiziert und zu einer Amazon Rekognition-Sammlung hinzugefügt werden.

Weitere Informationen finden Sie unter [Hinzufügen von Gesichtern zu einer Sammlung](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "doc-example-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var indexFacesRequest = new IndexFacesRequest
        {
```

```
        Image = image,
        CollectionId = collectionId,
        ExternalImageId = photo,
        DetectionAttributes = new List<string>() { "ALL" },
    };

    IndexFacesResponse indexFacesResponse = await
rekognitionClient.IndexFacesAsync(indexFacesRequest);

    Console.WriteLine($"{photo} added");
    foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
    {
        Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
    }
}
```

- Weitere API-Informationen finden Sie unter [IndexFaces](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Sammlungen

Das folgende Codebeispiel zeigt, wie Sie Amazon Rekognition-Sammlungen auflisten.

Weitere Informationen finden Sie unter [Sammlungen auflisten](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
/// <summary>
/// Uses Amazon Rekognition to list the collection IDs in the
/// current account.
/// </summary>
public class ListCollections
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        var listCollectionsRequest = new ListCollectionsRequest
        {
            MaxResults = limit,
        };

        var listCollectionsResponse = new ListCollectionsResponse();

        do
        {
            if (listCollectionsResponse is not null)
            {
                listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
            }

            listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

            listCollectionsResponse.CollectionIds.ForEach(id =>
            {
                Console.WriteLine(id);
            });
        }
        while (listCollectionsResponse.NextToken is not null);
    }
}
```


- Weitere API-Informationen finden Sie unter [ListCollections](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Gesichtern in einer Sammlung

Das folgende Codebeispiel zeigt, wie Gesichter in einer Amazon Rekognition-Sammlung aufgelistet werden.

Weitere Informationen finden Sie unter [Gesichter in einer Sammlung auflisten](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon Rekognition Service to retrieve the list of faces
/// stored in a collection.
/// </summary>
public class ListFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";

        var rekognitionClient = new AmazonRekognitionClient();

        var listFacesResponse = new ListFacesResponse();
        Console.WriteLine($"Faces in collection {collectionId}");

        var listFacesRequest = new ListFacesRequest
        {
            CollectionId = collectionId,
            MaxResults = 1,
```

```
};

do
{
    listFacesResponse = await
rekognitionClient.ListFacesAsync(listFacesRequest);
    listFacesResponse.Faces.ForEach(face =>
    {
        Console.WriteLine(face.FaceId);
    });

    listFacesRequest.NextToken = listFacesResponse.NextToken;
}
while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
}
```

- Weitere API-Informationen finden Sie unter [ListFaces](#) in der APIAWS SDK for .NET -Referenz für .

Erkennen von Prominenten in einem Bild

Das folgende Codebeispiel zeigt, wie Prominente in einem Bild mit Amazon Rekognition erkannt werden.

Weitere Informationen finden Sie unter [Erkennen von Prominenten in einem Bild](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
```

```
using Amazon.Rekognition.Model;

/// <summary>
/// Shows how to use Amazon Rekognition to identify celebrities in a photo.
/// </summary>
public class CelebritiesInImage
{
    public static async Task Main(string[] args)
    {
        string photo = "moviestars.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

        var img = new Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
            data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
        }
        catch (Exception)
        {
            Console.WriteLine($"Failed to load file {photo}");
            return;
        }

        img.Bytes = new MemoryStream(data);
        recognizeCelebritiesRequest.Image = img;

        Console.WriteLine($"Looking for celebrities in image {photo}\n");

        var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

        Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
        recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
        {
            Console.WriteLine($"Celebrity recognized: {celeb.Name}");
            Console.WriteLine($"Celebrity ID: {celeb.Id}");
        });
    }
}
```

```
        BoundingBox boundingBox = celeb.Face.BoundingBox;
        Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
        Console.WriteLine("Further information (if available):");
        celeb.Urls.ForEach(url =>
        {
            Console.WriteLine(url);
        });
    });

    Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
}
}
```

- Weitere API-Informationen finden Sie unter [RecognizeCelebrities](#) in der APIAWS SDK for .NET -Referenz für .

Suche nach Gesichtern in einer Sammlung

Das folgende Codebeispiel zeigt, wie Sie in einer Amazon Rekognition-Sammlung nach Gesichtern suchen, die mit einem anderen Gesicht aus der Sammlung übereinstimmen.

Weitere Informationen finden Sie unter [Nach einem Gesicht suchen \(Gesichts-ID\)](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
/// <summary>
/// Uses the Amazon Rekognition Service to find faces in an image that
/// match the face Id provided in the method request.
/// </summary>
public class SearchFacesMatchingId
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

        var rekognitionClient = new AmazonRekognitionClient();

        // Search collection for faces matching the face id.
        var searchFacesRequest = new SearchFacesRequest
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
            MaxFaces = 2,
        };

        SearchFacesResponse searchFacesResponse = await
rekognitionClient.SearchFacesAsync(searchFacesRequest);

        Console.WriteLine("Face matching faceId " + faceId);

        Console.WriteLine("Matche(s): ");
        searchFacesResponse.FaceMatches.ForEach(face =>
        {
            Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
{face.Similarity}");
        });
    }
}
```

- Weitere API-Informationen finden Sie unter [SearchFaces](#) in der APIAWS SDK for .NET - Referenz für .

Suche nach Gesichtern in einer Sammlung im Vergleich zu einem Referenzbild

Das folgende Codebeispiel zeigt, wie Sie in einer Amazon Rekognition-Sammlung im Vergleich zu einem Referenzbild nach Gesichtern suchen.

Weitere Informationen finden Sie unter [Nach einem Gesicht suchen \(Bild\)](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Uses the Amazon Rekognition Service to search for images matching those
/// in a collection.
/// </summary>
public class SearchFacesMatchingImage
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string bucket = "bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        var image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo,
            },
        };
    }
};
```

```
var searchFacesByImageRequest = new SearchFacesByImageRequest()
{
    CollectionId = collectionId,
    Image = image,
    FaceMatchThreshold = 70F,
    MaxFaces = 2,
};

SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

Console.WriteLine("Faces matching largest face in image from " + photo);
searchFacesByImageResponse.FaceMatches.ForEach(face =>
{
    Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
});
}
```

- Weitere API-Informationen finden Sie unter [SearchFacesByImage](#) in der APIAWS SDK for .NET -Referenz für .

Beispiele für die Route-53-Domainregistrierung mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Route-53-Domainregistrierung verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Route-53-Domainregistrierung

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit der Route-53-Domainregistrierung.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
public static class HelloRoute53Domains
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the Amazon Route 53 domain registration 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<IAmazonRoute53Domains>()
            ).Build();

        // Now the client is available for injection.
        var route53Client =
            host.Services.GetRequiredService<IAmazonRoute53Domains>();

        // You can use await and any of the async methods to get a response.
        var response = await route53Client.ListPricesAsync(new ListPricesRequest
        { Tld = "com" });
        Console.WriteLine($"Hello Amazon Route 53 Domains! Following are prices
        for .com domain operations:");
        var comPrices = response.Prices.FirstOrDefault();
        if (comPrices != null)
        {
            Console.WriteLine($"Registration: {comPrices.RegistrationPrice?.Price}
            {comPrices.RegistrationPrice?.Currency}");
            Console.WriteLine($"Renewal: {comPrices.RenewalPrice?.Price}
            {comPrices.RenewalPrice?.Currency}");
        }
    }
}
```



```
    }  
  }  
}
```

- Weitere API-Informationen finden Sie unter [ListPrices](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Überprüfen der Domainverfügbarkeit

Das folgende Codebeispiel zeigt, wie Sie die Verfügbarkeit einer Domain überprüfen.

AWS SDK for .NET

Note

Auf [GitHub](#) gibt es mehr. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Check the availability of a domain name.  
/// </summary>  
/// <param name="domain">The domain to check for availability.</param>  
/// <returns>An availability result string.</returns>  
public async Task<string> CheckDomainAvailability(string domain)  
{  
    var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(  
        new CheckDomainAvailabilityRequest  
        {  
            DomainName = domain  
        }  
    )  
}
```

```
    );  
    return result.Availability.Value;  
}
```

- Weitere API-Informationen finden Sie unter [CheckDomainAvailability](#) in der APIAWS SDK for .NET -Referenz für .

Überprüfen der Domainübertragbarkeit

Das folgende Codebeispiel zeigt, wie Sie die Übertragbarkeit einer Domain überprüfen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Check the transferability of a domain name.  
/// </summary>  
/// <param name="domain">The domain to check for transferability.</param>  
/// <returns>A transferability result string.</returns>  
public async Task<string> CheckDomainTransferability(string domain)  
{  
    var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(  
        new CheckDomainTransferabilityRequest  
        {  
            DomainName = domain  
        }  
    );  
    return result.Transferability.Transferable.Value;  
}
```

- Weitere API-Informationen finden Sie unter [CheckDomainTransferability](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen von Domainedetails

Das folgende Codebeispiel zeigt, wie Sie die Details für eine Domain abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get details for a domain.
/// </summary>
/// <returns>A string with detail information about the domain.</returns>
public async Task<string> GetDomainDetail(string domainName)
{
    try
    {
        var result = await _amazonRoute53Domains.GetDomainDetailAsync(
            new GetDomainDetailRequest()
            {
                DomainName = domainName
            });
        var details = $"{\tDomain {domainName}:\n" +
            $"{\tCreated on {result.CreationDate.ToShortDateString()}.
\n" +
            $"{\tAdmin contact is {result.AdminContact.Email}.\n" +
            $"{\tAuto-renew is {result.AutoRenew}.\n";

        return details;
    }
    catch (InvalidInputException)
    {
        return $"Domain {domainName} was not found in your account.";
    }
}
```

- Weitere API-Informationen finden Sie unter [GetDomainDetail](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Vorgangsdetails

Das folgende Codebeispiel zeigt, wie Sie Details zu einer -Operation abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get details for a domain action operation.
/// </summary>
/// <param name="operationId">The operational Id.</param>
/// <returns>A string describing the operational details.</returns>
public async Task<string> GetOperationDetail(string? operationId)
{
    if (operationId == null)
        return "Unable to get operational details because ID is null.";
    try
    {
        var operationDetails =
            await _amazonRoute53Domains.GetOperationDetailAsync(
                new GetOperationDetailRequest
                {
                    OperationId = operationId
                }
            );

        var details = $"{\tOperation {operationId}:\n" +
            $"{\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}. \n" +
            $"{\tMessage is {operationDetails.Message}. \n" +
            $"{\tStatus is {operationDetails.Status}. \n";

        return details;
    }
}
```

```
    }
    catch (AmazonRoute53DomainsException ex)
    {
        return $"Unable to get operation details. Here's why: {ex.Message}.";
    }
}
```

- Weitere API-Informationen finden Sie unter [GetOperationDetail](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen vorgeschlagener Domainnamen

Das folgende Codebeispiel zeigt, wie Sie Vorschläge für Domainnamen abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get a list of suggestions for a given domain.
/// </summary>
/// <param name="domain">The domain to check for suggestions.</param>
/// <param name="onlyAvailable">If true, only returns available domains.</param>
/// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
/// <returns>A collection of domain suggestions.</returns>
public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
{
    var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
        new GetDomainSuggestionsRequest
        {
            DomainName = domain,
            OnlyAvailable = onlyAvailable,
            SuggestionCount = suggestionCount
        }
    );
}
```

```
    }  
    );  
    return result.SuggestionsList;  
}
```

- Weitere API-Informationen finden Sie unter [GetDomainSuggestions](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Domainpreisen

Das folgende Codebeispiel zeigt, wie Sie Domain-Preise auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// List prices for domain type operations.  
/// </summary>  
/// <param name="domainTypes">Domain types to include in the results.</param>  
/// <returns>The list of domain prices.</returns>  
public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)  
{  
    var results = new List<DomainPrice>();  
    var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new  
ListPricesRequest());  
    // Get the entire list using the paginator.  
    await foreach (var prices in paginatePrices.Prices)  
    {  
        results.Add(prices);  
    }  
    return results.Where(p => domainTypes.Contains(p.Name)).ToList();  
}
```

- Weitere API-Informationen finden Sie unter [ListPrices](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Domains

Das folgende Codebeispiel zeigt, wie Sie die registrierten Domänen auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List the domains for the account.
/// </summary>
/// <returns>A collection of domain summary records.</returns>
public async Task<List<DomainSummary>> ListDomains()
{
    var results = new List<DomainSummary>();
    var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
        new ListDomainsRequest());

    // Get the entire list using the paginator.
    await foreach (var domain in paginateDomains.Domains)
    {
        results.Add(domain);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [ListDomains](#) in der APIAWS SDK for .NET -Referenz für .

Listenoperationen

Das folgende Codebeispiel zeigt, wie Sie -Operationen auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List operations for the account that are submitted after a specified date.
/// </summary>
/// <returns>A collection of operation summary records.</returns>
public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
{
    var results = new List<OperationSummary>();
    var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
        new ListOperationsRequest()
        {
            SubmittedSince = submittedSince
        });

    // Get the entire list using the paginator.
    await foreach (var operations in paginateOperations.Operations)
    {
        results.Add(operations);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [ListOperations](#) in der APIAWS SDK for .NET - Referenz für .

Registrieren einer Domäne

Das folgende Codebeispiel zeigt, wie Sie eine Domain registrieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Initiate a domain registration request.
/// </summary>
/// <param name="contact">Contact details.</param>
/// <param name="domainName">The domain name to register.</param>
/// <param name="autoRenew">True if the domain should automatically renew.</
param>
/// <param name="duration">The duration in years for the domain registration.</
param>
/// <returns>The operation Id.</returns>
public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
duration, ContactDetail contact)
{
    // This example uses the same contact information for admin, registrant, and
    tech contacts.
    try
    {
        var result = await _amazonRoute53Domains.RegisterDomainAsync(
            new RegisterDomainRequest()
            {
                AdminContact = contact,
                RegistrantContact = contact,
                TechContact = contact,
                DomainName = domainName,
                AutoRenew = autoRenew,
                DurationInYears = duration,
                PrivacyProtectAdminContact = false,
                PrivacyProtectRegistrantContact = false,
                PrivacyProtectTechContact = false
            }
        )
    }
}
```

```
        );
        return result.OperationId;
    }
    catch (InvalidInputException)
    {
        _logger.LogInformation($"Unable to request registration for domain
{domainName}");
        return null;
    }
}
```

- Weitere API-Informationen finden Sie unter [RegisterDomain](#) in der APIAWS SDK for .NET - Referenz für .

Anzeigen der Abrechnung

Das folgende Codebeispiel zeigt, wie Sie Fakturierungsdatensätze anzeigen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// View billing records for the account between a start and end date.
/// </summary>
/// <param name="startDate">The start date for billing results.</param>
/// <param name="endDate">The end date for billing results.</param>
/// <returns>A collection of billing records.</returns>
public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
{
    var results = new List<BillingRecord>();
    var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
        new ViewBillingRequest()
        {
            Start = startDate,
```

```
        End = endDate
    });

    // Get the entire list using the paginator.
    await foreach (var billingRecords in paginateBilling.BillingRecords)
    {
        results.Add(billingRecords);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [ViewBilling](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit Domains

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Auflisten der aktuellen Domains und der Vorgänge des letzten Jahres
- Anzeigen der Abrechnung für das vergangene Jahr und der Preise für Domaintypen
- Abrufen von Domainvorschlägen
- Überprüfen der Verfügbarkeit und Übertragbarkeit von Domains
- Optional: Anfordern einer Domainregistrierung
- Abrufen eines Vorgangsdetails
- Optional: Abrufen eines Domaindetails

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
public static class Route53DomainScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. List current domains.
        2. List operations in the past year.
        3. View billing for the account in the past year.
        4. View prices for domain types.
        5. Get domain suggestions.
        6. Check domain availability.
        7. Check domain transferability.
        8. Optionally, request a domain registration.
        9. Get an operation detail.
        10. Optionally, get a domain detail.
    */

    private static Route53Wrapper _route53Wrapper = null!;
    private static IConfiguration _configuration = 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<IAmazonRoute53Domains>()
                    .AddTransient<Route53Wrapper>()
            )
            .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();

var logger = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
}).CreateLogger(typeof(Route53DomainScenario));

_route53Wrapper = host.Services.GetRequiredService<Route53Wrapper>();

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon Route 53 domains example
scenario.");
Console.WriteLine(new string('-', 80));

try
{
    await ListDomains();
    await ListOperations();
    await ListBillingRecords();
    await ListPrices();
    await ListDomainSuggestions();
    await CheckDomainAvailability();
    await CheckDomainTransferability();
    var operationId = await RequestDomainRegistration();
    await GetOperationalDetail(operationId);
    await GetDomainDetails();
}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
}

Console.WriteLine(new string('-', 80));
Console.WriteLine("The Amazon Route 53 domains example scenario is
complete.");
Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List account registered domains.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDomains()
{
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List account domains.");
        var domains = await _route53Wrapper.ListDomains();
        for (int i = 0; i < domains.Count; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {domains[i].DomainName}");
        }

        if (!domains.Any())
        {
            Console.WriteLine("\\tNo domains found in this account.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain operations in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListOperations()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List account domain operations in the past year.");
        var operations = await _route53Wrapper.ListOperations(
            DateTime.Today.AddYears(-1));
        for (int i = 0; i < operations.Count; i++)
        {
            Console.WriteLine($"\\tOperation Id: {operations[i].OperationId}");
            Console.WriteLine($"\\tStatus: {operations[i].Status}");
            Console.WriteLine($"\\tDate: {operations[i].SubmittedDate}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List billing in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListBillingRecords()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. View billing for the account in the past year.");
        var billingRecords = await _route53Wrapper.ViewBilling(
```

```
        DateTime.Today.AddYears(-1),
        DateTime.Today);
    for (int i = 0; i < billingRecords.Count; i++)
    {
        Console.WriteLine($"\\tBill Date:
{billingRecords[i].BillDate.ToShortDateString()}");
        Console.WriteLine($"\\tOperation: {billingRecords[i].Operation}");
        Console.WriteLine($"\\tPrice: {billingRecords[i].Price}");
    }
    if (!billingRecords.Any())
    {
        Console.WriteLine("\\tNo billing records found in this account for the
past year.");
    }
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List prices for a few domain types.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListPrices()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"4. View prices for domain types.");
    var domainTypes = new List<string> { "net", "com", "org", "co" };

    var prices = await _route53Wrapper.ListPrices(domainTypes);
    foreach (var pr in prices)
    {
        Console.WriteLine($"\\tName: {pr.Name}");
        Console.WriteLine($"\\tRegistration: {pr.RegistrationPrice?.Price}
{pr.RegistrationPrice?.Currency}");
        Console.WriteLine($"\\tRenewal: {pr.RenewalPrice?.Price}
{pr.RenewalPrice?.Currency}");
        Console.WriteLine($"\\tTransfer: {pr.TransferPrice?.Price}
{pr.TransferPrice?.Currency}");
        Console.WriteLine($"\\tChange Ownership: {pr.ChangeOwnershipPrice?.Price}
{pr.ChangeOwnershipPrice?.Currency}");
        Console.WriteLine($"\\tRestoration: {pr.RestorationPrice?.Price}
{pr.RestorationPrice?.Currency}");
        Console.WriteLine();
    }
    Console.WriteLine(new string('-', 80));
}
```

```
}

/// <summary>
/// List domain suggestions for a domain name.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDomainSuggestions()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"5. Get domain suggestions.");
    string? domainName = null;
    while (domainName == null || string.IsNullOrWhiteSpace(domainName))
    {
        Console.WriteLine($"Enter a domain name to get available domain
suggestions.");
        domainName = Console.ReadLine();
    }

    var suggestions = await _route53Wrapper.GetDomainSuggestions(domainName,
true, 5);
    foreach (var suggestion in suggestions)
    {
        Console.WriteLine($"    \tSuggestion Name: {suggestion.DomainName}");
        Console.WriteLine($"    \tAvailability: {suggestion.Availability}");
    }
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Check availability for a domain name.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CheckDomainAvailability()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"6. Check domain availability.");
    string? domainName = null;
    while (domainName == null || string.IsNullOrWhiteSpace(domainName))
    {
        Console.WriteLine($"Enter a domain name to check domain availability.");
        domainName = Console.ReadLine();
    }
}
```



```
        var availability = await
_route53Wrapper.CheckDomainAvailability(domainName);
        Console.WriteLine($"\\tAvailability: {availability}");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainTransferability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Check domain transferability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain
transferability.");
            domainName = Console.ReadLine();
        }

        var transferability = await
_route53Wrapper.CheckDomainTransferability(domainName);
        Console.WriteLine($"\\tTransferability: {transferability}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> RequestDomainRegistration()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Optionally, request a domain registration.");

        Console.WriteLine($"\\tNote: This example uses domain request settings in
settings.json.");
        Console.WriteLine($"\\tTo change the domain registration settings, set the
values in that file.");
        Console.WriteLine($"\\tRemember, registering an actual domain will incur an
account billing cost.");
    }
}
```

```
        Console.WriteLine($"\\tWould you like to begin a domain registration? (y/n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase))
        {
            string domainName = _configuration["DomainName"];
            ContactDetail contact = new ContactDetail();
            contact.CountryCode =
CountryCode.FindValue(_configuration["Contact:CountryCode"]);
            contact.ContactType =
ContactType.FindValue(_configuration["Contact:ContactType"]);

            _configuration.GetSection("Contact").Bind(contact);

            var operationId = await _route53Wrapper.RegisterDomain(
                domainName,
                Convert.ToBoolean(_configuration["AutoRenew"]),
                Convert.ToInt32(_configuration["DurationInYears"]),
                contact);
            if (operationId != null)
            {
                Console.WriteLine(
                    $"\\tRegistration requested. Operation Id: {operationId}");
            }

            return operationId;
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }

    /// <summary>
    /// Get details for an operation.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetOperationalDetail(string? operationId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Get an operation detail.");

        var operationDetails =
            await _route53Wrapper.GetOperationDetail(operationId);
    }
}
```

```
        Console.WriteLine(operationDetails);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Optionally, get details for a registered domain.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> GetDomainDetails()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"!0. Get details on a domain.");

        Console.WriteLine($"\\tNote: you must have a registered domain to get
details.");
        Console.WriteLine($"\\tWould you like to get domain details? (y/n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase))
        {
            string? domainName = null;
            while (domainName == null)
            {
                Console.WriteLine($"\\tEnter a domain name to get details.");
                domainName = Console.ReadLine();
            }

            var domainDetails = await _route53Wrapper.GetDomainDetail(domainName);
            Console.WriteLine(domainDetails);
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }
}
```

Wrapper-Methoden, die vom Szenario für Route-53-Domainregistrierungsaktionen verwendet werden.

```
public class Route53Wrapper
{
    private readonly IAmazonRoute53Domains _amazonRoute53Domains;
    private readonly ILogger<Route53Wrapper> _logger;
    public Route53Wrapper(IAmazonRoute53Domains amazonRoute53Domains,
        ILogger<Route53Wrapper> logger)
    {
        _amazonRoute53Domains = amazonRoute53Domains;
        _logger = logger;
    }

    /// <summary>
    /// List prices for domain type operations.
    /// </summary>
    /// <param name="domainTypes">Domain types to include in the results.</param>
    /// <returns>The list of domain prices.</returns>
    public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
    {
        var results = new List<DomainPrice>();
        var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
ListPricesRequest());
        // Get the entire list using the paginator.
        await foreach (var prices in paginatePrices.Prices)
        {
            results.Add(prices);
        }
        return results.Where(p => domainTypes.Contains(p.Name)).ToList();
    }

    /// <summary>
    /// Check the availability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for availability.</param>
    /// <returns>An availability result string.</returns>
    public async Task<string> CheckDomainAvailability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
            new CheckDomainAvailabilityRequest
            {
                DomainName = domain
            }
        );
    }
}
```

```
        return result.Availability.Value;
    }

    /// <summary>
    /// Check the transferability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for transferability.</param>
    /// <returns>A transferability result string.</returns>
    public async Task<string> CheckDomainTransferability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
            new CheckDomainTransferabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Transferability.Transferable.Value;
    }

    /// <summary>
    /// Get a list of suggestions for a given domain.
    /// </summary>
    /// <param name="domain">The domain to check for suggestions.</param>
    /// <param name="onlyAvailable">If true, only returns available domains.</param>
    /// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
    /// <returns>A collection of domain suggestions.</returns>
    public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
    {
        var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
            new GetDomainSuggestionsRequest
            {
                DomainName = domain,
                OnlyAvailable = onlyAvailable,
                SuggestionCount = suggestionCount
            }
        );
        return result.SuggestionsList;
    }
}
```

```
/// <summary>
/// Get details for a domain action operation.
/// </summary>
/// <param name="operationId">The operational Id.</param>
/// <returns>A string describing the operational details.</returns>
public async Task<string> GetOperationDetail(string? operationId)
{
    if (operationId == null)
        return "Unable to get operational details because ID is null.";
    try
    {
        var operationDetails =
            await _amazonRoute53Domains.GetOperationDetailAsync(
                new GetOperationDetailRequest
                {
                    OperationId = operationId
                }
            );

        var details = $"{\tOperation {operationId}:\n" +
            $"{\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}\n" +
            $"{\tMessage is {operationDetails.Message}.\n" +
            $"{\tStatus is {operationDetails.Status}.\n";

        return details;
    }
    catch (AmazonRoute53DomainsException ex)
    {
        return $"Unable to get operation details. Here's why: {ex.Message}.";
    }
}

/// <summary>
/// Initiate a domain registration request.
/// </summary>
/// <param name="contact">Contact details.</param>
/// <param name="domainName">The domain name to register.</param>
/// <param name="autoRenew">True if the domain should automatically renew.</
param>
/// <param name="duration">The duration in years for the domain registration.</
param>
/// <returns>The operation Id.</returns>
```

```
public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
duration, ContactDetail contact)
{
    // This example uses the same contact information for admin, registrant, and
tech contacts.
    try
    {
        var result = await _amazonRoute53Domains.RegisterDomainAsync(
            new RegisterDomainRequest()
            {
                AdminContact = contact,
                RegistrantContact = contact,
                TechContact = contact,
                DomainName = domainName,
                AutoRenew = autoRenew,
                DurationInYears = duration,
                PrivacyProtectAdminContact = false,
                PrivacyProtectRegistrantContact = false,
                PrivacyProtectTechContact = false
            }
        );
        return result.OperationId;
    }
    catch (InvalidInputException)
    {
        _logger.LogInformation($"Unable to request registration for domain
{domainName}");
        return null;
    }
}

/// <summary>
/// View billing records for the account between a start and end date.
/// </summary>
/// <param name="startDate">The start date for billing results.</param>
/// <param name="endDate">The end date for billing results.</param>
/// <returns>A collection of billing records.</returns>
public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
{
    var results = new List<BillingRecord>();
    var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
        new ViewBillingRequest()
```

```
        {
            Start = startDate,
            End = endDate
        });

    // Get the entire list using the paginator.
    await foreach (var billingRecords in paginateBilling.BillingRecords)
    {
        results.Add(billingRecords);
    }
    return results;
}

/// <summary>
/// List the domains for the account.
/// </summary>
/// <returns>A collection of domain summary records.</returns>
public async Task<List<DomainSummary>> ListDomains()
{
    var results = new List<DomainSummary>();
    var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
        new ListDomainsRequest());

    // Get the entire list using the paginator.
    await foreach (var domain in paginateDomains.Domains)
    {
        results.Add(domain);
    }
    return results;
}

/// <summary>
/// List operations for the account that are submitted after a specified date.
/// </summary>
/// <returns>A collection of operation summary records.</returns>
public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
{
    var results = new List<OperationSummary>();
    var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
        new ListOperationsRequest()
    {
```



```

        SubmittedSince = submittedSince
    });

    // Get the entire list using the paginator.
    await foreach (var operations in paginateOperations.Operations)
    {
        results.Add(operations);
    }
    return results;
}

/// <summary>
/// Get details for a domain.
/// </summary>
/// <returns>A string with detail information about the domain.</returns>
public async Task<string> GetDomainDetail(string domainName)
{
    try
    {
        var result = await _amazonRoute53Domains.GetDomainDetailAsync(
            new GetDomainDetailRequest()
            {
                DomainName = domainName
            });
        var details = $"\\tDomain {domainName}:\\n" +
            $"\\tCreated on {result.CreationDate.ToShortDateString()}.\\n" +
            $"\\tAdmin contact is {result.AdminContact.Email}.\\n" +
            $"\\tAuto-renew is {result.AutoRenew}.\\n";

        return details;
    }
    catch (InvalidInputException)
    {
        return $"Domain {domainName} was not found in your account.";
    }
}
}

```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CheckDomainAvailability](#)

- [CheckDomainTransferability](#)
- [GetDomainDetail](#)
- [GetDomainSuggestions](#)
- [GetOperationDetail](#)
- [ListDomains](#)
- [ListOperations](#)
- [ListPrices](#)
- [RegisterDomain](#)
- [ViewBilling](#)

Amazon S3-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon S3 verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

CORS-Regeln einem Bucket hinzufügen

Das folgende Codebeispiel zeigt, wie CORS-Regeln (Cross-Origin Resource Sharing) zu einem S3-Bucket hinzugefügt werden.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add CORS configuration to the Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to apply the CORS configuration to an Amazon S3 bucket.</param>
/// <param name="configuration">The CORS configuration to apply.</param>
private static async Task PutCORSConfigurationAsync(AmazonS3Client client,
CORSConfiguration configuration)
{
    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest()
    {
        BucketName = BucketName,
        Configuration = configuration,
    };


    _ = await client.PutCORSConfigurationAsync(request);
}
```

- Weitere API-Informationen finden Sie unter [PutBucketCors](#) in der APIAWS SDK for .NET - Referenz für .

Einem Bucket eine Lebenszyklus-Konfiguration hinzufügen

Das folgende Codebeispiel zeigt, wie Sie einem S3-Bucket eine Lebenszykluskonfiguration hinzufügen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Adds lifecycle configuration information to the S3 bucket named in
/// the bucketName parameter.
/// </summary>
/// <param name="client">The S3 client used to call the
/// PutLifecycleConfigurationAsync method.</param>
/// <param name="bucketName">A string representing the S3 bucket to
/// which configuration information will be added.</param>
/// <param name="configuration">A LifecycleConfiguration object that
/// will be applied to the S3 bucket.</param>
public static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
string bucketName, LifecycleConfiguration configuration)
{
    var request = new PutLifecycleConfigurationRequest()
    {
        BucketName = bucketName,
        Configuration = configuration,
    };
    var response = await client.PutLifecycleConfigurationAsync(request);
}
```

- Weitere API-Informationen finden Sie unter [PutBucketLifecycleConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Abbrechen mehrteiliger Uploads

Das folgende Codebeispiel veranschaulicht, wie Sie eine mehrteilige Uploads abbrechen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to use the Amazon Simple Storage Service
/// (Amazon S3) to stop a multi-part upload process using the Amazon S3
/// TransferUtility.
/// </summary>
public class AbortMPU
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();

        await AbortMPUAsync(client, bucketName);
    }

    /// <summary>
    /// Cancels the multi-part copy process.
    /// </summary>
    /// <param name="client">The initialized client object used to create
    /// the TransferUtility object.</param>
    /// <param name="bucketName">The name of the S3 bucket where the
    /// multi-part copy operation is in progress.</param>
    public static async Task AbortMPUAsync(IAmazonS3 client, string bucketName)
    {
```

```
        try
        {
            var transferUtility = new TransferUtility(client);

            // Cancel all in-progress uploads initiated before the specified
date.
            await transferUtility.AbortMultipartUploadsAsync(
                bucketName, DateTime.Now.AddDays(-7));
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine($"Error: {e.Message}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [AbortMultipartUploads](#) in der APIAWS SDK for .NET -Referenz für .

Ein Objekt von einem Bucket in einen anderen Bucket kopieren

Im folgenden Codebeispiel wird demonstriert, wie Sie ein S3-Objekt von einem Bucket in einen anderen kopieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

public class CopyObject
{
```

```

public static async Task Main()
{
    // Specify the AWS Region where your buckets are located if it is
    // different from the AWS Region of the default user.
    IAmazonS3 s3Client = new AmazonS3Client();

    // Remember to change these values to refer to your Amazon S3 objects.
    string sourceBucketName = "doc-example-bucket1";
    string destinationBucketName = "doc-example-bucket2";
    string sourceObjectKey = "testfile.txt";
    string destinationObjectKey = "testfilecopy.txt";

    Console.WriteLine($"Copying {sourceObjectKey} from {sourceBucketName} to
");
    Console.WriteLine($"{{destinationBucketName}} as {{destinationObjectKey}}");

    var response = await CopyingObjectAsync(
        s3Client,
        sourceObjectKey,
        destinationObjectKey,
        sourceBucketName,
        destinationBucketName);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine("\nCopy complete.");
    }
}

/// <summary>
/// This method calls the AWS SDK for .NET to copy an
/// object from one Amazon S3 bucket to another.
/// </summary>
/// <param name="client">The Amazon S3 client object.</param>
/// <param name="sourceKey">The name of the object to be copied.</param>
/// <param name="destinationKey">The name under which to save the copy.</
param>
/// <param name="sourceBucketName">The name of the Amazon S3 bucket
/// where the file is located now.</param>
/// <param name="destinationBucketName">The name of the Amazon S3
/// bucket where the copy should be saved.</param>
/// <returns>Returns a CopyObjectResponse object with the results from
/// the async call.</returns>
public static async Task<CopyObjectResponse> CopyingObjectAsync(

```

```
    IAmazonS3 client,  
    string sourceKey,  
    string destinationKey,  
    string sourceBucketName,  
    string destinationBucketName)  
{  
    var response = new CopyObjectResponse();  
    try  
    {  
        var request = new CopyObjectRequest  
        {  
            SourceBucket = sourceBucketName,  
            SourceKey = sourceKey,  
            DestinationBucket = destinationBucketName,  
            DestinationKey = destinationKey,  
        };  
        response = await client.CopyObjectAsync(request);  
    }  
    catch (AmazonS3Exception ex)  
    {  
        Console.WriteLine($"Error copying object: '{ex.Message}'");  
    }  
  
    return response;  
}  
}
```

- Weitere API-Informationen finden Sie unter [CopyObject](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines -Buckets

Das folgende Codebeispiel zeigen, wie Sie einen S3 Bucket erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Shows how to create a new Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <returns>A boolean value representing the success or failure of
/// the bucket creation process.</returns>
public static async Task<bool> CreateBucketAsync(IAmazonS3 client, string
bucketName)
{
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
        };

        var response = await client.PutBucketAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error creating bucket: '{ex.Message}'");
        return false;
    }
}
```

Erstellen Sie einen Bucket mit aktivierter Objektsperre.

```
/// <summary>
```

```
/// Create a new Amazon S3 bucket with object lock actions.
/// </summary>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
enableObjectLock)
{
    Console.WriteLine($"\\tCreating bucket {bucketName} with object lock
{enableObjectLock}.");
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
            ObjectLockEnabledForBucket = enableObjectLock,
        };

        var response = await _amazonS3.PutBucketAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error creating bucket: '{ex.Message}'");
        return false;
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateBucket](#) in der APIAWS SDK for .NET - Referenz für .

CORS-Regeln aus einem Bucket löschen

Das folgende Codebeispiel zeigt, wie CORS-Regeln aus einem S3-Bucket gelöscht werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Deletes a CORS configuration from an Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to delete the CORS configuration from the bucket.</param>
private static async Task DeleteCORSConfigurationAsync(AmazonS3Client
client)
{
    DeleteCORSConfigurationRequest request = new
DeleteCORSConfigurationRequest()
    {
        BucketName = BucketName,
    };
    await client.DeleteCORSConfigurationAsync(request);
}
```

- Weitere API-Informationen finden Sie unter [DeleteBucketCors](#) in der APIAWS SDK for .NET - Referenz für .

Einen leeren Bucket löschen

Im folgenden Codebeispiel wird demonstriert, wie Sie einen leeren S3 Bucket löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client object.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</
param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public static async Task<bool> DeleteBucketAsync(IAmazonS3 client, string
bucketName)
    {
        var request = new DeleteBucketRequest
        {
            BucketName = bucketName,
        };

        var response = await client.DeleteBucketAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteBucket](#) in der APIAWS SDK for .NET - Referenz für .

Ein Objekt löschen

Im folgenden Codebeispiel wird demonstriert, wie Sie ein S3-Objekt löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie ein Objekt in einem nicht versionierten S3-Bucket.

```
using System;
```

```
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete an object from a non-versioned Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteObject
{
    /// <summary>
    /// The Main method initializes the necessary variables and then calls
    /// the DeleteObjectNonVersionedBucketAsync method to delete the object
    /// named by the keyName parameter.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";
        const string keyName = "testfile.txt";

        // If the Amazon S3 bucket is located in an AWS Region other than the
        // Region of the default account, define the AWS Region for the
        // Amazon S3 bucket in your call to the AmazonS3Client constructor.
        // For example RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
        await DeleteObjectNonVersionedBucketAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// The DeleteObjectNonVersionedBucketAsync takes care of deleting the
    /// desired object from the named bucket.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client used to delete
    /// an object from an Amazon S3 bucket.</param>
    /// <param name="bucketName">The name of the bucket from which the
    /// object will be deleted.</param>
    /// <param name="keyName">The name of the object to delete.</param>
    public static async Task DeleteObjectNonVersionedBucketAsync(IAmazonS3
client, string bucketName, string keyName)
    {
        try
        {
            var deleteObjectRequest = new DeleteObjectRequest
            {
```

```

        BucketName = bucketName,
        Key = keyName,
    };

    Console.WriteLine($"Deleting object: {keyName}");
    await client.DeleteObjectAsync(deleteObjectRequest);
    Console.WriteLine($"Object: {keyName} deleted from {bucketName}.");
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error encountered on server.
Message: '{ex.Message}' when deleting an object.");
}
}
}

```

Löschen Sie ein Objekt in einem versionierten S3-Bucket.

```

using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example creates an object in an Amazon Simple Storage Service
/// (Amazon S3) bucket and then deletes the object version that was
/// created.
/// </summary>
public class DeleteObjectVersion
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "verstioned-object.txt";

        // If the AWS Region of the default user is different from the AWS
        // Region of the Amazon S3 bucket, pass the AWS Region of the
        // bucket region to the Amazon S3 client object's constructor.
        // Define it like this:
        //     RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        IAmazonS3 client = new AmazonS3Client();
    }
}

```

```
        await CreateAndDeleteObjectVersionAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// This method creates and then deletes a versioned object.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// create and delete the object.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object will be created and deleted.</param>
    /// <param name="keyName">The key name of the object to create.</param>
    public static async Task CreateAndDeleteObjectVersionAsync(IAmazonS3 client,
string bucketName, string keyName)
    {
        try
        {
            // Add a sample object.
            string versionID = await PutAnObject(client, bucketName, keyName);

            // Delete the object by specifying an object key and a version ID.
            DeleteObjectRequest request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = keyName,
                VersionId = versionID,
            };

            Console.WriteLine("Deleting an object");
            await client.DeleteObjectAsync(request);
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }

    /// <summary>
    /// This method is used to create the temporary Amazon S3 object.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 object which will be used
    /// to create the temporary Amazon S3 object.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
object
```

```
    /// will be created.</param>
    /// <param name="objectKey">The name of the Amazon S3 object to create.</
param>
    /// <returns>The Version ID of the created object.</returns>
    public static async Task<string> PutAnObject(IAmazonS3 client, string
bucketName, string objectKey)
    {
        PutObjectRequest request = new PutObjectRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = await client.PutObjectAsync(request);
        return response.VersionId;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteObject](#) in der APIAWS SDK for .NET - Referenz für .

Mehrere Objekte löschen

Im folgenden Codebeispiel wird demonstriert, wie Sie mehrere Objekte aus einem S3 Bucket löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie alle Objekte aus einem S3 Bucket.

```
    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
```



```
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket from which the
/// contents will be deleted.</param>
/// <returns>A boolean value that represents the success or failure of
/// deleting all of the objects in the bucket.</returns>
public static async Task<bool> DeleteBucketContentsAsync(IAmazonS3 client,
string bucketName)
{
    // Iterate over the contents of the bucket and delete all objects.
    var request = new ListObjectsV2Request
    {
        BucketName = bucketName,
    };

    try
    {
        ListObjectsV2Response response;

        do
        {
            response = await client.ListObjectsV2Async(request);
            response.S3Objects
                .ForEach(async obj => await
client.DeleteObjectAsync(bucketName, obj.Key));

            // If the response is truncated, set the request
ContinuationToken
            // from the NextContinuationToken property of the response.
            request.ContinuationToken = response.NextContinuationToken;
        }
        while (response.IsTruncated);

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting objects: {ex.Message}");
        return false;
    }
}
```

Löschen Sie mehrere Objekte in einem nicht versionierten S3-Bucket.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete multiple objects from an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    /// <summary>
    /// The Main method initializes the Amazon S3 client and the name of
    /// the bucket and then passes those values to MultiObjectDeleteAsync.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";

        // If the Amazon S3 bucket from which you wish to delete objects is not
        // located in the same AWS Region as the default user, define the
        // AWS Region for the Amazon S3 bucket as a parameter to the client
        // constructor.
        IAmazonS3 s3Client = new AmazonS3Client();

        await MultiObjectDeleteAsync(s3Client, bucketName);
    }

    /// <summary>
    /// This method uses the passed Amazon S3 client to first create and then
    /// delete three files from the named bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// Amazon S3 methods.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where objects
    /// will be created and then deleted.</param>
    public static async Task MultiObjectDeleteAsync(IAmazonS3 client, string
bucketName)
    {
        // Create three sample objects which we will then delete.
    }
}
```

```
var keysAndVersions = await PutObjectsAsync(client, 3, bucketName);

// Now perform the multi-object delete, passing the key names and
// version IDs. Since we are working with a non-versioned bucket,
// the object keys collection includes null version IDs.
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
{
    BucketName = bucketName,
    Objects = keysAndVersions,
};

// You can add a specific object key to the delete request using the
// AddKey method of the multiObjectDeleteRequest.
try
{
    DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
    Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
}
catch (DeleteObjectsException e)
{
    PrintDeletionErrorStatus(e);
}
}

/// <summary>
/// Prints the list of errors raised by the call to DeleteObjectsAsync.
/// </summary>
/// <param name="ex">A collection of exceptions returned by the call to
/// DeleteObjectsAsync.</param>
public static void PrintDeletionErrorStatus(DeleteObjectsException ex)
{
    DeleteObjectsResponse errorResponse = ex.Response;
    Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

    Console.WriteLine($"Successfully deleted
{errorResponse.DeletedObjects.Count}.");
    Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");

    Console.WriteLine("Printing error data...");
    foreach (DeleteError deleteError in errorResponse.DeleteErrors)
    {
```

```
        Console.WriteLine($"Object Key:
{deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
    }
}

/// <summary>
/// This method creates simple text file objects that can be used in
/// the delete method.
/// </summary>
/// <param name="client">The Amazon S3 client used to call PutObjectAsync.</
param>
/// <param name="number">The number of objects to create.</param>
/// <param name="bucketName">The name of the bucket where the objects
/// will be created.</param>
/// <returns>A list of keys (object keys) and versions that the calling
/// method will use to delete the newly created files.</returns>
public static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3 client,
int number, string bucketName)
{
    List<KeyVersion> keys = new List<KeyVersion>();
    for (int i = 0; i < number; i++)
    {
        string key = "ExampleObject-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = await client.PutObjectAsync(request);

        // For non-versioned bucket operations, we only need the
        // object key.
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
        };
        keys.Add(keyVersion);
    }

    return keys;
}
}
```

Löschen Sie mehrere Objekte in einem versionierten S3-Bucket.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete objects in a version-enabled Amazon
/// Simple StorageService (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region for your Amazon S3 bucket is different from
        // the AWS Region of the default user, define the AWS Region for
        // the Amazon S3 bucket and pass it to the client constructor
        // like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        IAmazonS3 s3Client;

        s3Client = new AmazonS3Client();
        await DeleteMultipleObjectsFromVersionedBucketAsync(s3Client,
bucketName);
    }

    /// <summary>
    /// This method removes multiple versions and objects from a
    /// version-enabled Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
}
```

```

    public static async Task
DeleteMultipleObjectsFromVersionedBucketAsync(IAmazonS3 client, string bucketName)
    {
        // Delete objects (specifying object version in the request).
        await DeleteObjectVersionsAsync(client, bucketName);

        // Delete objects (without specifying object version in the request).
        var deletedObjects = await DeleteObjectsAsync(client, bucketName);

        // Additional exercise - remove the delete markers Amazon S3 returned
from
        // the preceding response. This results in the objects reappearing
        // in the bucket (you can verify the appearance/disappearance of
        // objects in the console).
        await RemoveDeleteMarkersAsync(client, bucketName, deletedObjects);
    }

    /// <summary>
    /// Creates and then deletes non-versioned Amazon S3 objects and then
deletes
    /// them again. The method returns a list of the Amazon S3 objects deleted.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// PutObjectsAsync and NonVersionedDeleteAsync.</param>
    /// <param name="bucketName">The name of the bucket where the objects
    /// will be created and then deleted.</param>
    /// <returns>A list of DeletedObjects.</returns>
    public static async Task<List<DeletedObject>> DeleteObjectsAsync(IAmazonS3
client, string bucketName)
    {
        // Upload the sample objects.
        var keysAndVersions2 = await PutObjectsAsync(client, bucketName, 3);

        // Delete objects using only keys. Amazon S3 creates a delete marker and
        // returns its version ID in the response.
        List<DeletedObject> deletedObjects = await
NonVersionedDeleteAsync(client, bucketName, keysAndVersions2);
        return deletedObjects;
    }

    /// <summary>
    /// This method creates several temporary objects and then deletes them.
    /// </summary>

```

```

    /// <param name="client">The S3 client.</param>
    /// <param name="bucketName">Name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteObjectVersionsAsync(IAmazonS3 client, string
bucketName)
    {
        // Upload the sample objects.
        var keysAndVersions1 = await PutObjectsAsync(client, bucketName, 3);

        // Delete the specific object versions.
        await VersionedDeleteAsync(client, bucketName, keysAndVersions1);
    }

    /// <summary>
    /// Displays the list of information about deleted files to the console.
    /// </summary>
    /// <param name="e">Error information from the delete process.</param>
    private static void DisplayDeletionErrors(DeleteObjectsException e)
    {
        var errorResponse = e.Response;
        Console.WriteLine($"No. of objects successfully deleted =
{errorResponse.DeletedObjects.Count}");
        Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");
        Console.WriteLine("Printing error data...");
        foreach (var deleteError in errorResponse.DeleteErrors)
        {
            Console.WriteLine($"Object Key:
{deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
        }
    }

    /// <summary>
    /// Delete multiple objects from a version-enabled bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="keys">A list of key names for the objects to delete.</
param>

```

```

    private static async Task VersionedDeleteAsync(IAmazonS3 client, string
bucketName, List<KeyVersion> keys)
    {
        var multiObjectDeleteRequest = new DeleteObjectsRequest
        {
            BucketName = bucketName,
            Objects = keys, // This includes the object keys and specific
version IDs.
        };

        try
        {
            Console.WriteLine("Executing VersionedDelete...");
            DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine($"Successfully deleted all the
{response.DeletedObjects.Count} items");
        }
        catch (DeleteObjectsException ex)
        {
            DisplayDeletionErrors(ex);
        }
    }

    /// <summary>
    /// Deletes multiple objects from a non-versioned Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="keys">A list of key names for the objects to delete.</
param>
    /// <returns>A list of the deleted objects.</returns>
    private static async Task<List<DeletedObject>>
NonVersionedDeleteAsync(IAmazonS3 client, string bucketName, List<KeyVersion> keys)
    {
        // Create a request that includes only the object key names.
        DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest();
        multiObjectDeleteRequest.BucketName = bucketName;

```



```

        foreach (var key in keys)
        {
            multiObjectDeleteRequest.AddKey(key.Key);
        }

        // Execute DeleteObjectsAsync.
        // The DeleteObjectsAsync method adds a delete marker for each
        // object deleted. You can verify that the objects were removed
        // using the Amazon S3 console.
        DeleteObjectsResponse response;
        try
        {
            Console.WriteLine("Executing NonVersionedDelete...");
            response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
        }
        catch (DeleteObjectsException ex)
        {
            DisplayDeletionErrors(ex);
            throw; // Some deletions failed. Investigate before continuing.
        }

        // This response contains the DeletedObjects list which we use to delete
the delete markers.
        return response.DeletedObjects;
    }

    /// <summary>
    /// Deletes the markers left after deleting the temporary objects.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="deletedObjects">A list of the objects that were deleted.</
param>
    private static async Task RemoveDeleteMarkersAsync(IAmazonS3 client, string
bucketName, List<DeletedObject> deletedObjects)
    {
        var keyVersionList = new List<KeyVersion>();

```

```
foreach (var deletedObject in deletedObjects)
{
    KeyVersion keyVersion = new KeyVersion
    {
        Key = deletedObject.Key,
        VersionId = deletedObject.DeleteMarkerVersionId,
    };
    keyVersionList.Add(keyVersion);
}

// Create another request to delete the delete markers.
var multiObjectDeleteRequest = new DeleteObjectsRequest
{
    BucketName = bucketName,
    Objects = keyVersionList,
};

// Now, delete the delete marker to bring your objects back to the
bucket.
try
{
    Console.WriteLine("Removing the delete markers .....");
    var deleteObjectResponse = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
    Console.WriteLine($"Successfully deleted the
{deleteObjectResponse.DeletedObjects.Count} delete markers");
}
catch (DeleteObjectsException ex)
{
    DisplayDeletionErrors(ex);
}
}

/// <summary>
/// Create temporary Amazon S3 objects to show how object deletion works in
an
/// Amazon S3 bucket with versioning enabled.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// PutObjectAsync to create temporary objects for the example.</param>
/// <param name="bucketName">A string representing the name of the S3
/// bucket where we will create the temporary objects.</param>
```

```
    /// <param name="number">The number of temporary objects to create.</param>
    /// <returns>A list of the KeyVersion objects.</returns>
    private static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3
client, string bucketName, int number)
    {
        var keys = new List<KeyVersion>();

        for (var i = 0; i < number; i++)
        {
            string key = "ObjectToDelete-" + new System.Random().Next();
            PutObjectRequest request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = "This is the content body!",
            };

            var response = await client.PutObjectAsync(request);
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
                VersionId = response.VersionId,
            };

            keys.Add(keyVersion);
        }

        return keys;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteObjects](#) in der APIAWS SDK for .NET - Referenz für .

Die Lebenszyklus-Konfiguration eines Buckets löschen

Das folgende Beispiel veranschaulicht, wie Sie die Lebenszyklus-Konfiguration eines S3 Buckets löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// This method removes the Lifecycle configuration from the named
/// S3 bucket.
/// </summary>
/// <param name="client">The S3 client object used to call
/// the RemoveLifecycleConfigAsync method.</param>
/// <param name="bucketName">A string representing the name of the
/// S3 bucket from which the configuration will be removed.</param>
public static async Task RemoveLifecycleConfigAsync(IAmazonS3 client, string
bucketName)
{
    var request = new DeleteLifecycleConfigurationRequest()
    {
        BucketName = bucketName,
    };
    await client.DeleteLifecycleConfigurationAsync(request);
}
```

- Weitere API-Informationen finden Sie unter [DeleteBucketLifecycle](#) in der APIAWS SDK for .NET -Referenz für .

Enable logging (Protokollierung aktivieren)

Das folgende Beispiel zeigt, wie Sie die Protokollierung in einem S3-Bucket aktivieren.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

/// <summary>
/// This example shows how to enable logging on an Amazon Simple Storage
/// Service (Amazon S3) bucket. You need to have two Amazon S3 buckets for
/// this example. The first is the bucket for which you wish to enable
/// logging, and the second is the location where you want to store the
/// logs.
/// </summary>
public class ServerAccessLogging
{
    private static IConfiguration _configuration = null!;

    public static async Task Main()
    {
        LoadConfig();

        string bucketName = _configuration["BucketName"];
        string logBucketName = _configuration["LogBucketName"];
        string logObjectKeyPrefix = _configuration["LogObjectKeyPrefix"];
        string accountId = _configuration["AccountId"];

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        try
```

```
    {
        // Update bucket policy for target bucket to allow delivery of logs
to it.
        await SetBucketPolicyToAllowLogDelivery(
            client,
            bucketName,
            logBucketName,
            logObjectKeyPrefix,
            accountId);

        // Enable logging on the source bucket.
        await EnableLoggingAsync(
            client,
            bucketName,
            logBucketName,
            logObjectKeyPrefix);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine($"Error: {e.Message}");
    }
}

/// <summary>
/// This method grants appropriate permissions for logging to the
/// Amazon S3 bucket where the logs will be stored.
/// </summary>
/// <param name="client">The initialized Amazon S3 client which will be used
/// to apply the bucket policy.</param>
/// <param name="sourceBucketName">The name of the source bucket.</param>
/// <param name="logBucketName">The name of the bucket where logging
/// information will be stored.</param>
/// <param name="logPrefix">The logging prefix where the logs should be
delivered.</param>
/// <param name="accountId">The account id of the account where the source
bucket exists.</param>
/// <returns>Async task.</returns>
public static async Task SetBucketPolicyToAllowLogDelivery(
    IAmazonS3 client,
    string sourceBucketName,
    string logBucketName,
    string logPrefix,
    string accountId)
{
```

```

var resourceArn = @"arn:aws:s3:::" + logBucketName + "/" + logPrefix +
@"*";

var newPolicy = @"{
    ""Statement"": [{
        ""Sid"": ""S3ServerAccessLogsPolicy"",
        ""Effect"": ""Allow"",
        ""Principal"": { ""Service"":
""logging.s3.amazonaws.com"" },
        ""Action"": [""s3:PutObject""],
        ""Resource"": ["" + resourceArn + @""],
        ""Condition"": {
            ""ArnLike"": { ""aws:SourceArn"": ""arn:aws:s3:::" +
sourceBucketName + @"" },
            ""StringEquals"": { ""aws:SourceAccount"": """" +
accountId + @"" }
        }
    }]
}";

Console.WriteLine($"The policy to apply to bucket {logBucketName} to
enable logging:");
Console.WriteLine(newPolicy);

PutBucketPolicyRequest putRequest = new PutBucketPolicyRequest
{
    BucketName = logBucketName,
    Policy = newPolicy,
};
await client.PutBucketPolicyAsync(putRequest);
Console.WriteLine("Policy applied.");
}

/// <summary>
/// This method enables logging for an Amazon S3 bucket. Logs will be stored
/// in the bucket you selected for logging. Selected prefix
/// will be prepended to each log object.
/// </summary>
/// <param name="client">The initialized Amazon S3 client which will be used
/// to configure and apply logging to the selected Amazon S3 bucket.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket for which you
/// wish to enable logging.</param>
/// <param name="logBucketName">The name of the Amazon S3 bucket where
logging
/// information will be stored.</param>

```

```
/// <param name="logObjectKeyPrefix">The prefix to prepend to each
/// object key.</param>
/// <returns>Async task.</returns>
public static async Task EnableLoggingAsync(
    IAmazonS3 client,
    string bucketName,
    string logBucketName,
    string logObjectKeyPrefix)
{
    Console.WriteLine($"Enabling logging for bucket {bucketName}.");
    var loggingConfig = new S3BucketLoggingConfig
    {
        TargetBucketName = logBucketName,
        TargetPrefix = logObjectKeyPrefix,
    };

    var putBucketLoggingRequest = new PutBucketLoggingRequest
    {
        BucketName = bucketName,
        LoggingConfig = loggingConfig,
    };
    await client.PutBucketLoggingAsync(putBucketLoggingRequest);
    Console.WriteLine($"Logging enabled.");
}

/// <summary>
/// Loads configuration from settings files.
/// </summary>
public static void LoadConfig()
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json", true) // Optionally, load local
settings.
        .Build();
}
}
```

- Weitere API-Informationen finden Sie unter [PutBucketLogging](#) in der APIAWS SDK for .NET - Referenz für .

Aktivieren von Benachrichtigungen

Das folgende Codebeispiel zeigt, wie Sie Benachrichtigungen für einen S3-Bucket aktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to enable notifications for an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class EnableNotifications
{
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket1";
        const string snsTopic = "arn:aws:sns:us-east-2:0123456789ab:bucket-
notify";
        const string sqsQueue = "arn:aws:sqs:us-
east-2:0123456789ab:Example_Queue";

        IAmazonS3 client = new AmazonS3Client(Amazon.RegionEndpoint.USEast2);
        await EnableNotificationAsync(client, bucketName, snsTopic, sqsQueue);
    }

    /// <summary>
    /// This method makes the call to the PutBucketNotificationAsync method.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client used to call
    /// the PutBucketNotificationAsync method.</param>
    /// <param name="bucketName">The name of the bucket for which
    /// notifications will be turned on.</param>
}
```

```
/// <param name="snsTopic">The ARN for the Amazon Simple Notification
/// Service (Amazon SNS) topic associated with the S3 bucket.</param>
/// <param name="sqsQueue">The ARN of the Amazon Simple Queue Service
/// (Amazon SQS) queue to which notifications will be pushed.</param>
public static async Task EnableNotificationAsync(
    IAmazonS3 client,
    string bucketName,
    string snsTopic,
    string sqsQueue)
{
    try
    {
        // The bucket for which we are setting up notifications.
        var request = new PutBucketNotificationRequest()
        {
            BucketName = bucketName,
        };

        // Defines the topic to use when sending a notification.
        var topicConfig = new TopicConfiguration()
        {
            Events = new List<EventType> { EventType.ObjectCreatedCopy },
            Topic = snsTopic,
        };
        request.TopicConfigurations = new List<TopicConfiguration>
        {
            topicConfig,
        };
        request.QueueConfigurations = new List<QueueConfiguration>
        {
            new QueueConfiguration()
            {
                Events = new List<EventType> { EventType.ObjectCreatedPut },
                Queue = sqsQueue,
            },
        };

        // Now apply the notification settings to the bucket.
        PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}
```

```
    }  
  }  
}
```

- Weitere API-Informationen finden Sie unter [PutBucketNotificationConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Aktivieren von Transfer Acceleration

Das folgende Beispiel zeigt, wie Sie Transfer Acceleration für einen S3-Bucket aktivieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Threading.Tasks;  
using Amazon.S3;  
using Amazon.S3.Model;  
  
/// <summary>  
/// Amazon Simple Storage Service (Amazon S3) Transfer Acceleration is a  
/// bucket-level feature that enables you to perform faster data transfers  
/// to Amazon S3. This example shows how to configure Transfer  
/// Acceleration.  
/// </summary>  
public class TransferAcceleration  
{  
    /// <summary>  
    /// The main method initializes the client object and sets the  
    /// Amazon Simple Storage Service (Amazon S3) bucket name before  
    /// calling EnableAccelerationAsync.  
    /// </summary>  
    public static async Task Main()  
    {  
        var s3Client = new AmazonS3Client();
```

```
        const string bucketName = "doc-example-bucket";

        await EnableAccelerationAsync(s3Client, bucketName);
    }

    /// <summary>
    /// This method sets the configuration to enable transfer acceleration
    /// for the bucket referred to in the bucketName parameter.
    /// </summary>
    /// <param name="client">An Amazon S3 client used to enable the
    /// acceleration on an Amazon S3 bucket.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket for which the
    /// method will be enabling acceleration.</param>
    private static async Task EnableAccelerationAsync(AmazonS3Client client,
string bucketName)
    {
        try
        {
            var putRequest = new PutBucketAccelerateConfigurationRequest
            {
                BucketName = bucketName,
                AccelerateConfiguration = new AccelerateConfiguration
                {
                    Status = BucketAccelerateStatus.Enabled,
                },
            };
            await client.PutBucketAccelerateConfigurationAsync(putRequest);

            var getRequest = new GetBucketAccelerateConfigurationRequest
            {
                BucketName = bucketName,
            };
            var response = await
client.GetBucketAccelerateConfigurationAsync(getRequest);

            Console.WriteLine($"Acceleration state = '{response.Status}' ");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error occurred. Message: '{ex.Message}' when
setting transfer acceleration");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [PutBucketAccelerateConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

CORS-Regeln für einen Bucket abrufen

Das folgende Codebeispiel zeigt, wie CORS-Regeln (Cross-Origin Resource Sharing) für einen S3-Bucket abgerufen werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve the CORS configuration applied to the Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to retrieve the CORS configuration.</param>
/// <returns>The created CORS configuration object.</returns>
private static async Task<CORSConfiguration>
RetrieveCORSConfigurationAsync(AmazonS3Client client)
{
    GetCORSConfigurationRequest request = new GetCORSConfigurationRequest()
    {
        BucketName = BucketName,
    };
    var response = await client.GetCORSConfigurationAsync(request);
    var configuration = response.Configuration;
    PrintCORSRules(configuration);
    return configuration;
}
```

- Weitere API-Informationen finden Sie unter [GetBucketCors](#) in der APIAWS SDK for .NET - Referenz für .

Ein Objekt aus einem Bucket abrufen

Im folgenden Codebeispiel wird veranschaulicht, wie Sie Daten aus einem Objekt in einem S3 Bucket lesen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public static async Task<bool> DownloadObjectFromBucketAsync(
    IAmazonS3 client,
    string bucketName,
    string objectName,
    string filePath)
{
    // Create a GetObject request
    var request = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
    };
};
```

```
// Issue request and remember to dispose of the response
using GetObjectResponse response = await client.GetObjectAsync(request);


try
{
    // Save object to local file
    await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error saving {objectName}: {ex.Message}");
    return false;
}
}
```

- Weitere API-Informationen finden Sie unter [GetObject](#) in der APIAWS SDK for .NET -Referenz für .

Die ACL eines Buckets abrufen

Das folgende Codebeispiel zeigt, wie Sie die Zugriffssteuerungsliste (ACL) eines S3-Buckets abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the access control list (ACL) for the new bucket.
/// </summary>
/// <param name="client">The initialized client object used to get the
/// access control list (ACL) of the bucket.</param>
```

```

    /// <param name="newBucketName">The name of the newly created bucket.</
param>
    /// <returns>An S3AccessControlList.</returns>
    public static async Task<S3AccessControlList> GetACLForBucketAsync(IAmazonS3
client, string newBucketName)
    {
        // Retrieve bucket ACL to show that the ACL was properly applied to
        // the new bucket.
        GetACLResponse getACLResponse = await client.GetACLAsync(new
GetACLRequest
    {
        BucketName = newBucketName,
    });

        return getACLResponse.AccessControlList;
    }

```

- Weitere API-Informationen finden Sie unter [GetBucketAcl](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen der Konfiguration für die rechtliche Aufbewahrungsfrist eines Objekts

Das folgende Codebeispiel zeigt, wie Sie die Konfiguration für die gesetzliche Aufbewahrungsfrist eines S3-Buckets abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>

```



```
public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };


        var response = await _amazonS3.GetObjectLegalHoldAsync(request);
        Console.WriteLine($"\\tObject legal hold for {objectKey} in {bucketName}:
" +
            $"\\n\\tStatus: {response.LegalHold.Status}");
        return response.LegalHold;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}
```

- Weitere API-Informationen finden Sie unter [GetObjectLegalHold](#) in der APIAWS SDK for .NET - Referenz für .

Die Lebenszyklus-Konfiguration eines Buckets abrufen

Das folgende Codebeispiel zeigt, wie Sie die Lebenszyklus-Konfiguration eines S3 Buckets abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Returns a configuration object for the supplied bucket name.
    /// </summary>
    /// <param name="client">The S3 client object used to call
    /// the GetLifecycleConfigurationAsync method.</param>
    /// <param name="bucketName">The name of the S3 bucket for which a
    /// configuration will be created.</param>
    /// <returns>Returns a new LifecycleConfiguration object.</returns>
    public static async Task<LifecycleConfiguration>
RetrieveLifecycleConfigAsync(IAmazonS3 client, string bucketName)
    {
        var request = new GetLifecycleConfigurationRequest()
        {
            BucketName = bucketName,
        };
        var response = await client.GetLifecycleConfigurationAsync(request);
        var configuration = response.Configuration;
        return configuration;
    }

```

- Weitere API-Informationen finden Sie unter [GetBucketLifecycleConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen der Objektsperrekonfiguration eines Buckets

Das folgende Codebeispiel zeigt, wie Sie die Objektsperrekonfiguration eines S3-Buckets abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>

```

```
/// <returns>The bucket's object lock configuration details.</returns>
public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
{
    try
    {
        var request = new GetObjectLockConfigurationRequest()
        {
            BucketName = bucketName
        };

        var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
        Console.WriteLine($"\\tBucket object lock config for {bucketName} in
{bucketName}: " +
                        $"\\n\\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                        $"\\n\\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

        return response.ObjectLockConfiguration;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to fetch object lock config:
'{ex.Message}'");
        return new ObjectLockConfiguration();
    }
}
```

- Weitere API-Informationen finden Sie unter [GetObjectLockConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Abrufen der Aufbewahrungskonfiguration eines Objekts

Das folgende Codebeispiel zeigt, wie Sie die Aufbewahrungskonfiguration eines S3-Objekts abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the retention period for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object retention details.</returns>
public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectRetentionAsync(request);
        Console.WriteLine($"  \tObject retention for {objectKey} in {bucketName}:
" +
            $"\n\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
        return response.Retention;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"  \tUnable to fetch object lock retention:
'{ex.Message}'");
        return new ObjectLockRetention();
    }
}
```

- Weitere API-Informationen finden Sie unter [GetObjectRetention](#) in der APIAWS SDK for .NET - Referenz für .

Die Website-Konfiguration für einen Bucket abrufen

Das folgende Codebeispiel zeigt, wie Sie die Website-Konfiguration für einen S3-Bucket abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get the website configuration.
GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
{
    BucketName = bucketName,
};
GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
Console.WriteLine($"Index document:
{getResponse.WebsiteConfiguration.IndexDocumentSuffix}");
Console.WriteLine($"Error document:
{getResponse.WebsiteConfiguration.ErrorDocument}");
```

- Weitere API-Informationen finden Sie unter [GetBucketWebsite](#) in der APIAWS SDK for .NET - Referenz für .

Buckets auflisten

Das folgende Codebeispiel zeigt, wie Sie S3-Buckets auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace ListBucketsExample
{
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example uses the AWS SDK for .NET to list the Amazon Simple Storage
    /// Service (Amazon S3) buckets belonging to the default account.
    /// </summary>
    public class ListBuckets
    {
        private static IAmazonS3 _s3Client;

        /// <summary>
        /// Get a list of the buckets owned by the default user.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client object.</param>
        /// <returns>The response from the ListingBuckets call that contains a
        /// list of the buckets owned by the default user.</returns>
        public static async Task<ListBucketsResponse> GetBuckets(IAmazonS3 client)
        {
            return await client.ListBucketsAsync();
        }

        /// <summary>
        /// This method lists the name and creation date for the buckets in
        /// the passed List of S3 buckets.
        /// </summary>
        /// <param name="bucketList">A List of S3 bucket objects.</param>
        public static void DisplayBucketList(List<S3Bucket> bucketList)
        {
```

```
        bucketList
            .ForEach(b => Console.WriteLine($"Bucket name: {b.BucketName},
created on: {b.CreationDate}"));
    }

    public static async Task Main()
    {
        // The client uses the AWS Region of the default user.
        // If the Region where the buckets were created is different,
        // pass the Region to the client constructor. For example:
        // _s3Client = new AmazonS3Client(RegionEndpoint.USEast1);
        _s3Client = new AmazonS3Client();
        var response = await GetBuckets(_s3Client);
        DisplayBucketList(response.Buckets);
    }
}
}
```

- Weitere API-Informationen finden Sie unter [ListBuckets](#) in der APIAWS SDK for .NET -Referenz für .

Objektversionen in einem Bucket auflisten

Das folgende Codebeispiel zeigt, wie Sie Objektversionen in einem S3 Bucket auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example lists the versions of the objects in a version enabled
```

```
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class ListObjectVersions
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region where your bucket is defined is different from
        // the AWS Region where the Amazon S3 bucket is defined, pass the
constant
        // for the AWS Region to the client constructor like this:
        //     var client = new AmazonS3Client(RegionEndpoint.USWest2);
        IAmazonS3 client = new AmazonS3Client();
        await GetObjectListWithAllVersionsAsync(client, bucketName);
    }

    /// <summary>
    /// This method lists all versions of the objects within an Amazon S3
    /// version enabled bucket.
    /// </summary>
    /// <param name="client">The initialized client object used to call
    /// ListVersionsAsync.</param>
    /// <param name="bucketName">The name of the version enabled Amazon S3
bucket
    /// for which you want to list the versions of the contained objects.</
param>
    public static async Task GetObjectListWithAllVersionsAsync(IAmazonS3 client,
string bucketName)
    {
        try
        {
            // When you instantiate the ListVersionRequest, you can
            // optionally specify a key name prefix in the request
            // if you want a list of object versions of a specific object.

            // For this example we set a small limit in MaxKeys to return
            // a small list of versions.
            ListVersionsRequest request = new ListVersionsRequest()
            {
                BucketName = bucketName,
                MaxKeys = 2,
            };

```



```
        do
        {
            ListVersionsResponse response = await
client.ListVersionsAsync(request);

            // Process response.
            foreach (S3ObjectVersion entry in response.Versions)
            {
                Console.WriteLine($"key: {entry.Key} size: {entry.Size}");
            }

            // If response is truncated, set the marker to get the next
            // set of keys.
            if (response.IsTruncated)
            {
                request.KeyMarker = response.NextKeyMarker;
                request.VersionIdMarker = response.NextVersionIdMarker;
            }
            else
            {
                request = null;
            }
        }
        while (request != null);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: '{ex.Message}'");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [ListObjectVersions](#) in der APIAWS SDK for .NET - Referenz für .

Objekte in einem Bucket auflisten

Im folgenden Codebeispiel wird veranschaulicht, wie Sie Objekte in einem S3 Bucket auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Shows how to list the objects in an Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket for which to list
/// the contents.</param>
/// <returns>A boolean value indicating the success or failure of the
/// copy operation.</returns>
public static async Task<bool> ListBucketContentsAsync(IAmazonS3 client,
string bucketName)
{
    try
    {
        var request = new ListObjectsV2Request
        {
            BucketName = bucketName,
            MaxKeys = 5,
        };

        Console.WriteLine("-----");
        Console.WriteLine($"Listing the contents of {bucketName}:");
        Console.WriteLine("-----");

        ListObjectsV2Response response;

        do
        {
            response = await client.ListObjectsV2Async(request);

            response.S3Objects
                .ForEach(obj => Console.WriteLine($"{obj.Key, -35}
{obj.LastModified.ToShortDateString(), 10}{obj.Size, 10}"));
        }
    }
}
```

```

        // If the response is truncated, set the request
ContinuationToken
        // from the NextContinuationToken property of the response.
        request.ContinuationToken = response.NextContinuationToken;
    }
    while (response.IsTruncated);

    return true;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error encountered on server.
Message: '{ex.Message}' getting list of objects.");
    return false;
}
}

```

Listen Sie Objekte mit einem Paginator auf.

```

using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// The following example lists objects in an Amazon Simple Storage
/// Service (Amazon S3) bucket.
/// </summary>
public class ListObjectsPaginator
{
    private const string BucketName = "doc-example-bucket";

    public static async Task Main()
    {
        IAmazonS3 s3Client = new AmazonS3Client();

        Console.WriteLine($"Listing the objects contained in {BucketName}:\n");
        await ListingObjectsAsync(s3Client, BucketName);
    }

    /// <summary>

```

```
/// This method uses a paginator to retrieve the list of objects in an
/// an Amazon S3 bucket.
/// </summary>
/// <param name="client">An Amazon S3 client object.</param>
/// <param name="bucketName">The name of the S3 bucket whose objects
/// you want to list.</param>
public static async Task ListingObjectsAsync(IAmazonS3 client, string
bucketName)
{
    var listObjectsV2Paginator = client.Paginators.ListObjectsV2(new
ListObjectsV2Request
    {
        BucketName = bucketName,
    });

    await foreach (var response in listObjectsV2Paginator.Responses)
    {
        Console.WriteLine($"HttpStatusCode: {response.HttpStatusCode}");
        Console.WriteLine($"Number of Keys: {response.KeyCount}");
        foreach (var entry in response.S3Objects)
        {
            Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListObjectsV2](#) in der APIAWS SDK for .NET - Referenz für .

Eine archivierte Kopie eines Objekts wiederherstellen

Das folgende Codebeispiel zeigt, wie Sie eine archivierte Kopie eines Objekts in einem S3 Bucket wiederherstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to restore an archived object in an Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class RestoreArchivedObject
{
    public static void Main()
    {
        string bucketName = "doc-example-bucket";
        string objectKey = "archived-object.txt";

        // Specify your bucket region (an example region is shown).
        RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

        IAmazonS3 client = new AmazonS3Client(bucketRegion);
        RestoreObjectAsync(client, bucketName, objectKey).Wait();
    }

    /// <summary>
    /// This method restores an archived object from an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// RestoreObjectAsync.</param>
    /// <param name="bucketName">A string representing the name of the
    /// bucket where the object was located before it was archived.</param>
    /// <param name="objectKey">A string representing the name of the
    /// archived object to restore.</param>
```

```
public static async Task RestoreObjectAsync(IAmazonS3 client, string
bucketName, string objectKey)
{
    try
    {
        var restoreRequest = new RestoreObjectRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Days = 2,
        };
        RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

        // Check the status of the restoration.
        await CheckRestorationStatusAsync(client, bucketName, objectKey);
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        Console.WriteLine($"Error: {amazonS3Exception.Message}");
    }
}

/// <summary>
/// This method retrieves the status of the object's restoration.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// GetObjectMetadataAsync.</param>
/// <param name="bucketName">A string representing the name of the Amazon
/// S3 bucket which contains the archived object.</param>
/// <param name="objectKey">A string representing the name of the
/// archived object you want to restore.</param>
public static async Task CheckRestorationStatusAsync(IAmazonS3 client,
string bucketName, string objectKey)
{
    GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest()
    {
        BucketName = bucketName,
        Key = objectKey,
    };
};
```

```

        GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);

        var restStatus = response.RestoreInProgress ? "in-progress" : "finished
or failed";
        Console.WriteLine($"Restoration status: {restStatus}");
    }
}

```

- Weitere API-Informationen finden Sie unter [RestoreObject](#) in der APIAWS SDK for .NET - Referenz für .

Eine neue ACL für einen Bucket festlegen

Das folgende Codebeispiel zeigt, wie Sie eine neue Zugriffssteuerungsliste (ACL) für einen S3-Bucket festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// Creates an Amazon S3 bucket with an ACL to control access to the
    /// bucket and the objects stored in it.
    /// </summary>
    /// <param name="client">The initialized client object used to create
    /// an Amazon S3 bucket, with an ACL applied to the bucket.
    /// </param>
    /// <param name="region">The AWS Region where the bucket will be created.</
param>
    /// <param name="newBucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value indicating success or failure.</returns>
    public static async Task<bool> CreateBucketUseCannedACLAsync(IAmazonS3
client, S3Region region, string newBucketName)

```

```
{
    try
    {
        // Create a new Amazon S3 bucket with Canned ACL.
        var putBucketRequest = new PutBucketRequest()
        {
            BucketName = newBucketName,
            BucketRegion = region,
            CannedACL = S3CannedACL.LogDeliveryWrite,
        };

        PutBucketResponse putBucketResponse = await
client.PutBucketAsync(putBucketRequest);

        return putBucketResponse.HttpStatusCode ==
System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Amazon S3 error: {ex.Message}");
    }

    return false;
}
```

- Weitere API-Informationen finden Sie unter [PutBucketAc](#) in der APIAWS SDK for .NET - Referenz für .

Festlegen des Standardaufbewahrungszeitraums eines Buckets

Das folgende Codebeispiel zeigt, wie der Standardaufbewahrungszeitraum eines S3-Buckets festgelegt wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```

/// <summary>
/// Set or modify a retention period on an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket to modify.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date for retention until.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
{
    var enabledString = enableObjectLock ? "Enabled" : "Disabled";
    var timeDifference = retainUntilDate.Subtract(DateTime.Now);
    try
    {
        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,
                Status = VersionStatus.Enabled
            }
        });

        var request = new PutObjectLockConfigurationRequest()
        {
            BucketName = bucketName,
            ObjectLockConfiguration = new ObjectLockConfiguration()
            {
                ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                Rule = new ObjectLockRule()
                {
                    DefaultRetention = new DefaultRetention()
                    {
                        Mode = retention,
                        Days = timeDifference.Days // Can be specified in days
or years but not both.
                    }
                }
            }
        };
    }
};

```

```

        var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
        Console.WriteLine($"\\tAdded a default retention to bucket
{bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying object lock: '{ex.Message}');
        return false;
    }
}

```

- Weitere API-Informationen finden Sie unter [PutObjectLockConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Festlegen der Konfiguration für die rechtliche Aufbewahrungsfrist eines Objekts

Das folgende Codebeispiel zeigt, wie Sie die Konfiguration für rechtliche Aufbewahrungsfristen eines S3-Objekts festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Set or modify a legal hold on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="holdStatus">The On or Off status for the legal hold.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectLegalHold(string bucketName,
    string objectKey, ObjectLockLegalHoldStatus holdStatus)
{
    try

```

```
{
    var request = new PutObjectLegalHoldRequest()
    {
        BucketName = bucketName,
        Key = objectKey,
        LegalHold = new ObjectLockLegalHold()
        {
            Status = holdStatus
        }
    };

    var response = await _amazonS3.PutObjectLegalHoldAsync(request);
    Console.WriteLine($"\\tModified legal hold for {objectKey} in
{bucketName}.");
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"\\tError modifying legal hold: '{ex.Message}'");
    return false;
}
}
```

- Weitere API-Informationen finden Sie unter [PutObjectLegalHold](#) in der APIAWS SDK for .NET - Referenz für .

Festlegen der Objektsperrekonfiguration eines -Buckets

Das folgende Codebeispiel zeigt, wie Sie die Objektsperrekonfiguration eines vorhandenen S3-Buckets festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Enable object lock on an existing bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket to modify.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableObjectLockOnBucket(string bucketName)
{
    try
    {
        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,
                Status = VersionStatus.Enabled
            }
        });

        var request = new PutObjectLockConfigurationRequest()
        {
            BucketName = bucketName,
            ObjectLockConfiguration = new ObjectLockConfiguration()
            {
                ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
            },
        };

        var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
        Console.WriteLine($"\\tAdded an object lock policy to bucket
{bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error modifying object lock: '{ex.Message}');
        return false;
    }
}
```

- Weitere API-Informationen finden Sie unter [PutObjectLockConfiguration](#) in der APIAWS SDK for .NET -Referenz für .

Festlegen des Aufbewahrungszeitraums eines Objekts

Das folgende Codebeispiel zeigt, wie der Aufbewahrungszeitraum eines S3-Objekts festgelegt wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Set or modify a retention period on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date retention expires.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
    string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
{
    try
    {
        var request = new PutObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            Retention = new ObjectLockRetention()
            {
                Mode = retention,
                RetainUntilDate = retainUntilDate
            }
        };

        var response = await _amazonS3.PutObjectRetentionAsync(request);
```

```
        Console.WriteLine($"\\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying retention period:
'{ex.Message}'");
        return false;
    }
}
```

- Weitere API-Informationen finden Sie unter [PutObjectRetention](#) in der APIAWS SDK for .NET - Referenz für .

Die Website-Konfiguration für einen Bucket festlegen

Das folgende Codebeispiel zeigt, wie Sie die Website-Konfiguration für einen S3-Bucket festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Put the website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
{
    BucketName = bucketName,
    WebsiteConfiguration = new WebsiteConfiguration()
    {
        IndexDocumentSuffix = indexDocumentSuffix,
        ErrorDocument = errorDocument,
    },
};
PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- Weitere API-Informationen finden Sie unter [PutBucketWebsite](#) in der APIAWS SDK for .NET - Referenz für .

Ein Objekt in einen Bucket hochladen

Im folgenden Codebeispiel wird veranschaulicht, wie Sie ein Objekt in einen S3 Bucket hochladen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>
public static async Task<bool> UploadFileAsync(
    IAmazonS3 client,
    string bucketName,
    string objectName,
    string filePath)
{
    var request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
        FilePath = filePath,
```

```
};

var response = await client.PutObjectAsync(request);
if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Successfully uploaded {objectName} to
{bucketName}.");
    return true;
}
else
{
    Console.WriteLine($"Could not upload {objectName} to
{bucketName}.");
    return false;
}
}
```

Laden Sie ein Objekt mit serverseitiger Verschlüsselung hoch.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to upload an object to an Amazon Simple Storage
/// Service (Amazon S3) bucket with server-side encryption enabled.
/// </summary>
public class ServerSideEncryption
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "samplefile.txt";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
```



```
        await WritingAnObjectAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// Upload a sample object include a setting for encryption.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// to upload a file and apply server-side encryption.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// encrypted object will reside.</param>
    /// <param name="keyName">The name for the object that you want to
    /// create in the supplied bucket.</param>
    public static async Task WritingAnObjectAsync(IAmazonS3 client, string
bucketName, string keyName)
    {
        try
        {
            var putRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
                ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256,
            };

            var putResponse = await client.PutObjectAsync(putRequest);

            // Determine the encryption state of an object.
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest
            {
                BucketName = bucketName,
                Key = keyName,
            };
            GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
            ServerSideEncryptionMethod objectEncryption =
response.ServerSideEncryptionMethod;

            Console.WriteLine($"Encryption method used: {0}",
objectEncryption.ToString());
        }
        catch (AmazonS3Exception ex)
        {
```

```
        Console.WriteLine($"Error: '{ex.Message}' when writing an object");  
    }  
}  
}
```

- Weitere API-Informationen finden Sie unter [PutObject](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Eine vorsignierte URL erstellen

Das folgende Codebeispiel zeigt, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Generieren Sie eine vorsignierte URL, die für einen begrenzten Zeitraum eine Amazon-S3-Aktion ausführen kann.

```
using System;  
using Amazon;  
using Amazon.S3;  
using Amazon.S3.Model;  
  
public class GenPresignedUrl  
{  
    public static void Main()  
    {  
        const string bucketName = "doc-example-bucket";  
        const string objectKey = "sample.txt";  
  
        // Specify how long the presigned URL lasts, in hours
```

```

        const double timeoutDuration = 12;

        // Specify the AWS Region of your Amazon S3 bucket. If it is
        // different from the Region defined for the default user,
        // pass the Region to the constructor for the client. For
        // example: new AmazonS3Client(RegionEndpoint.USEast1);

        // If using the Region us-east-1, and server-side encryption with AWS
        KMS, you must specify Signature Version 4.
        // Region us-east-1 defaults to Signature Version 2 unless explicitly
        set to Version 4 as shown below.
        // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
        userguide/UsingAWSSDK.html#specify-signature-version
        // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
        TAWSConfigsS3.html
        AWSConfigsS3.UseSignatureVersion4 = true;
        IAmazonS3 s3Client = new AmazonS3Client(RegionEndpoint.USEast1);

        string urlString = GeneratePresignedURL(s3Client, bucketName, objectKey,
        timeoutDuration);
        Console.WriteLine($"The generated URL is: {urlString}.");
    }

    /// <summary>
    /// Generate a presigned URL that can be used to access the file named
    /// in the objectKey parameter for the amount of time specified in the
    /// duration parameter.
    /// </summary>
    /// <param name="client">An initialized S3 client object used to call
    /// the GetPresignedUrl method.</param>
    /// <param name="bucketName">The name of the S3 bucket containing the
    /// object for which to create the presigned URL.</param>
    /// <param name="objectKey">The name of the object to access with the
    /// presigned URL.</param>
    /// <param name="duration">The length of time for which the presigned
    /// URL will be valid.</param>
    /// <returns>A string representing the generated presigned URL.</returns>
    public static string GeneratePresignedURL(IAmazonS3 client, string
    bucketName, string objectKey, double duration)
    {
        string urlString = string.Empty;
        try
        {
            var request = new GetPreSignedUrlRequest()

```

```
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = DateTime.UtcNow.AddHours(duration),
        };
        urlString = client.GetPreSignedURL(request);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: '{ex.Message}'");
    }

    return urlString;
}
}
```

Generieren Sie eine vorsignierte URL und führen Sie ein Upload mit dieser URL durch.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to upload an object to an Amazon Simple Storage
/// Service (Amazon S3) bucket using a presigned URL. The code first
/// creates a presigned URL and then uses it to upload an object to an
/// Amazon S3 bucket using that URL.
/// </summary>
public class UploadUsingPresignedURL
{
    private static HttpClient httpClient = new HttpClient();

    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "samplefile.txt";
        string filePath = $"source\\{keyName}";
```

```
// Specify how long the signed URL will be valid in hours.
double timeoutDuration = 12;

// Specify the AWS Region of your Amazon S3 bucket. If it is
// different from the Region defined for the default user,
// pass the Region to the constructor for the client. For
// example: new AmazonS3Client(RegionEndpoint.USEast1);

// If using the Region us-east-1, and server-side encryption with AWS
KMS, you must specify Signature Version 4.
// Region us-east-1 defaults to Signature Version 2 unless explicitly
set to Version 4 as shown below.
// For more details, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingAWSSDK.html#specify-signature-version
// and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/TAWSConfigsS3.html
AWSConfigsS3.UseSignatureVersion4 = true;
IAmazonS3 client = new AmazonS3Client(RegionEndpoint.USEast1);

var url = GeneratePreSignedURL(client, bucketName, keyName,
timeoutDuration);
var success = await UploadObject(filePath, url);

if (success)
{
    Console.WriteLine("Upload succeeded.");
}
else
{
    Console.WriteLine("Upload failed.");
}
}

/// <summary>
/// Uploads an object to an Amazon S3 bucket using the presigned URL passed
in
/// the url parameter.
/// </summary>
/// <param name="filePath">The path (including file name) to the local
/// file you want to upload.</param>
/// <param name="url">The presigned URL that will be used to upload the
/// file to the Amazon S3 bucket.</param>
/// <returns>A Boolean value indicating the success or failure of the
```

```
/// operation, based on the HttpResponseMessage.</returns>
public static async Task<bool> UploadObject(string filePath, string url)
{
    using var streamContent = new StreamContent(
        new FileStream(filePath, FileMode.Open, FileAccess.Read));

    var response = await httpClient.PutAsync(url, streamContent);
    return response.IsSuccessStatusCode;
}

/// <summary>
/// Generates a presigned URL which will be used to upload an object to
/// an Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// GetPreSignedURL.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket to which the
/// presigned URL will point.</param>
param>
/// <param name="objectKey">The name of the file that will be uploaded.</
/// <param name="duration">How long (in hours) the presigned URL will
/// be valid.</param>
/// <returns>The generated URL.</returns>
public static string GeneratePreSignedURL(
    IAmazonS3 client,
    string bucketName,
    string objectKey,
    double duration)
{
    var request = new GetPreSignedUrlRequest
    {
        BucketName = bucketName,
        Key = objectKey,
        Verb = HttpVerb.PUT,
        Expires = DateTime.UtcNow.AddHours(duration),
    };

    string url = client.GetPreSignedURL(request);
    return url;
}
}
```

Erste Schritte mit Buckets und Objekten

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Bucket und laden Sie eine Datei in ihn hoch.
- Laden Sie ein Objekt aus einem Bucket herunter.
- Kopieren Sie ein Objekt in einen Unterordner eines Buckets.
- Listen Sie die Objekte in einem Bucket auf.
- Löschen Sie die Bucket-Objekte und den Bucket.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
public class S3_Basics
{
    public static async Task Main()
    {
        // Create an Amazon S3 client object. The constructor uses the
        // default user installed on the system. To work with Amazon S3
        // features in a different AWS Region, pass the AWS Region as a
        // parameter to the client constructor.
        IAmazonS3 client = new AmazonS3Client();
        string bucketName = string.Empty;
        string filePath = string.Empty;
        string keyName = string.Empty;

        var sepBar = new string('-', Console.WindowWidth);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
    }
}
```

```
Console.WriteLine("\n\t2. Upload an object to the new bucket");
Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
Console.WriteLine("\n\t4. List the items in the new bucket");
Console.WriteLine("\n\t5. Delete all the items in the bucket");
Console.WriteLine("\n\t6. Delete the bucket");
Console.WriteLine(sepBar);

// Create a bucket.
Console.WriteLine($"{sepBar}");
Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
Console.WriteLine(sepBar);

Console.Write("Please enter a name for the new bucket: ");
bucketName = Console.ReadLine();

var success = await S3Bucket.CreateBucketAsync(client, bucketName);
if (success)
{
    Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
}
else
{
    Console.WriteLine($"Could not create bucket: {bucketName}.\n");
}

Console.WriteLine(sepBar);
Console.WriteLine("Upload a file to the new bucket.");
Console.WriteLine(sepBar);

// Get the local path and filename for the file to upload.
while (string.IsNullOrEmpty(filePath))
{
    Console.Write("Please enter the path and filename of the file to
upload: ");
    filePath = Console.ReadLine();

    // Confirm that the file exists on the local computer.
    if (!File.Exists(filePath))
    {
        Console.WriteLine($"Couldn't find {filePath}. Try again.\n");
        filePath = string.Empty;
    }
}
```



```
// Get the file name from the full path.
keyName = Path.GetFileName(filePath);

success = await S3Bucket.UploadFileAsync(client, bucketName, keyName,
filePath);

if (success)
{
    Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
}
else
{
    Console.WriteLine($"Could not upload {keyName}.\n");
}

// Set the file path to an empty string to avoid overwriting the
// file we just uploaded to the bucket.
filePath = string.Empty;

// Now get a new location where we can save the file.
while (string.IsNullOrEmpty(filePath))
{
    // First get the path to which the file will be downloaded.
    Console.Write("Please enter the path where the file will be
downloaded: ");
    filePath = Console.ReadLine();

    // Confirm that the file exists on the local computer.
    if (File.Exists($"{filePath}\\{keyName}"))
    {
        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
        filePath = string.Empty;
    }
}

// Download an object from a bucket.
success = await S3Bucket.DownloadObjectFromBucketAsync(client,
bucketName, keyName, filePath);

if (success)
{
```

```
        Console.WriteLine($"Successfully downloaded {keyName}.\n");
    }
    else
    {
        Console.WriteLine($"Sorry, could not download {keyName}.\n");
    }

    // Copy the object to a different folder in the bucket.
    string folderName = string.Empty;

    while (string.IsNullOrEmpty(folderName))
    {
        Console.Write("Please enter the name of the folder to copy your
object to: ");
        folderName = Console.ReadLine();
    }

    while (string.IsNullOrEmpty(keyName))
    {
        // Get the name to give to the object once uploaded.
        Console.Write("Enter the name of the object to copy: ");
        keyName = Console.ReadLine();
    }

    await S3Bucket.CopyObjectInBucketAsync(client, bucketName, keyName,
folderName);

    // List the objects in the bucket.
    await S3Bucket.ListBucketContentsAsync(client, bucketName);

    // Delete the contents of the bucket.
    await S3Bucket.DeleteBucketContentsAsync(client, bucketName);

    // Deleting the bucket too quickly after deleting its contents will
    // cause an error that the bucket isn't empty. So...
    Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
    _ = Console.ReadLine();

    // Delete the bucket.
    await S3Bucket.DeleteBucketAsync(client, bucketName);
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

Erste Schritte mit der Verschlüsselung

Das folgende Codebeispiel veranschaulicht die ersten Schritte mit der Verschlüsselung von Amazon-S3-Objekten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to apply client encryption to an object in an
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class SSEClientEncryption
{
    public static async Task Main()
```

```
{
    string bucketName = "doc-example-bucket";
    string keyName = "exampleobject.txt";
    string copyTargetKeyName = "examplecopy.txt";

    // If the AWS Region defined for your default user is different
    // from the Region where your Amazon S3 bucket is located,
    // pass the Region name to the Amazon S3 client object's constructor.
    // For example: RegionEndpoint.USWest2.
    IAmazonS3 client = new AmazonS3Client();

    try
    {
        // Create an encryption key.
        Aes aesEncryption = Aes.Create();
        aesEncryption.KeySize = 256;
        aesEncryption.GenerateKey();
        string base64Key = Convert.ToBase64String(aesEncryption.Key);

        // Upload the object.
        PutObjectRequest putObjectRequest = await UploadObjectAsync(client,
bucketName, keyName, base64Key);

        // Download the object and verify that its contents match what you
uploaded.
        await DownloadObjectAsync(client, bucketName, keyName, base64Key,
putObjectRequest);

        // Get object metadata and verify that the object uses AES-256
encryption.
        await GetObjectMetadataAsync(client, bucketName, keyName,
base64Key);

        // Copy both the source and target objects using server-side
encryption with
        // an encryption key.
        await CopyObjectAsync(client, bucketName, keyName,
copyTargetKeyName, aesEncryption, base64Key);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}
```

```

    /// <summary>
    /// Uploads an object to an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// PutObjectAsync.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket to which the
    /// object will be uploaded.</param>
    /// <param name="keyName">The name of the object to upload to the Amazon S3
    /// bucket.</param>
    /// <param name="base64Key">The encryption key.</param>
    /// <returns>The PutObjectRequest object for use by DownloadObjectAsync.</
returns>
    public static async Task<PutObjectRequest> UploadObjectAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string base64Key)
    {
        PutObjectRequest putObjectRequest = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = keyName,
            ContentBody = "sample text",
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key,
        };
        PutObjectResponse putObjectResponse = await
client.PutObjectAsync(putObjectRequest);
        return putObjectRequest;
    }

    /// <summary>
    /// Downloads an encrypted object from an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// GetObjectAsync.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
object
    /// is located.</param>

```

```
    /// <param name="keyName">The name of the Amazon S3 object to download.</  
param>  
    /// <param name="base64Key">The encryption key used to encrypt the  
    /// object.</param>  
    /// <param name="putObjectRequest">The PutObjectRequest used to upload  
    /// the object.</param>  
    public static async Task DownloadObjectAsync(  
        IAmazonS3 client,  
        string bucketName,  
        string keyName,  
        string base64Key,  
        PutObjectRequest putObjectRequest)  
    {  
        GetObjectRequest getObjectRequest = new GetObjectRequest  
        {  
            BucketName = bucketName,  
            Key = keyName,  
  
            // Provide encryption information for the object stored in Amazon  
S3.  
            ServerSideEncryptionCustomerMethod =  
ServerSideEncryptionCustomerMethod.AES256,  
            ServerSideEncryptionCustomerProvidedKey = base64Key,  
        };  
  
        using (GetObjectResponse getResponse = await  
client.GetObjectAsync(getObjectRequest))  
            using (StreamReader reader = new  
StreamReader(getResponse.ResponseStream))  
            {  
                string content = reader.ReadToEnd();  
                if (string.Compare(putObjectRequest.ContentBody, content) == 0)  
                {  
                    Console.WriteLine("Object content is same as we uploaded");  
                }  
                else  
                {  
                    Console.WriteLine("Error...Object content is not same.");  
                }  
  
                if (getResponse.ServerSideEncryptionCustomerMethod ==  
ServerSideEncryptionCustomerMethod.AES256)  
                {
```

```
        Console.WriteLine("Object encryption method is AES256, same as
we set");
    }
    else
    {
        Console.WriteLine("Error...Object encryption method is not the
same as AES256 we set");
    }
}

/// <summary>
/// Retrieves the metadata associated with an Amazon S3 object.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to call GetObjectMetadataAsync.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket containing the
/// object for which we want to retrieve metadata.</param>
/// <param name="keyName">The name of the object for which we wish to
/// retrieve the metadata.</param>
/// <param name="base64Key">The encryption key associated with the
/// object.</param>
public static async Task GetObjectMetadataAsync(
    IAmazonS3 client,
    string bucketName,
    string keyName,
    string base64Key)
{
    GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
    {
        BucketName = bucketName,
        Key = keyName,

        // The object stored in Amazon S3 is encrypted, so provide the
necessary encryption information.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key,
    };

    GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
```

```
        Console.WriteLine("The object metadata show encryption method used is:
{0}", getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
    }

    /// <summary>
    /// Copies an encrypted object from one Amazon S3 bucket to another.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// CopyObjectAsync.</param>
    /// <param name="bucketName">The Amazon S3 bucket containing the object
    /// to copy.</param>
    /// <param name="keyName">The name of the object to copy.</param>
    /// <param name="copyTargetKeyName">The Amazon S3 bucket to which the object
    /// will be copied.</param>
    /// <param name="aesEncryption">The encryption type to use.</param>
    /// <param name="base64Key">The encryption key to use.</param>
    public static async Task CopyObjectAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string copyTargetKeyName,
        Aes aesEncryption,
        string base64Key)
    {
        aesEncryption.GenerateKey();
        string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

        CopyObjectRequest copyRequest = new CopyObjectRequest
        {
            SourceBucket = bucketName,
            SourceKey = keyName,
            DestinationBucket = bucketName,
            DestinationKey = copyTargetKeyName,

            // Information about the source object's encryption.
            CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,

            // Information about the target object's encryption.
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = copyBase64Key,
```



```
        };  
        await client.CopyObjectAsync(copyRequest);  
    }  
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CopyObject](#)
 - [GetObject](#)
 - [GetObjectMetadata](#)

Erste Schritte mit Tags

Das folgende Codebeispiel zeigt die ersten Schritte mit Tags für Amazon-S3-Objekte.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Amazon;  
using Amazon.S3;  
using Amazon.S3.Model;  
  
/// <summary>  
/// This example shows how to work with tags in Amazon Simple Storage  
/// Service (Amazon S3) objects.  
/// </summary>  
public class ObjectTag  
{  
    public static async Task Main()  
    {  
        string bucketName = "doc-example-bucket";
```

```
string keyName = "newobject.txt";
string filePath = @"*** file path ***";

// Specify your bucket region (an example region is shown).
RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

var client = new AmazonS3Client(bucketRegion);
await PutObjectsWithTagsAsync(client, bucketName, keyName, filePath);
}

/// <summary>
/// This method uploads an object with tags. It then shows the tag
/// values, changes the tags, and shows the new tags.
/// </summary>
/// <param name="client">The Initialized Amazon S3 client object used
/// to call the methods to create and change an objects tags.</param>
/// <param name="bucketName">A string representing the name of the
/// bucket where the object will be stored.</param>
/// <param name="keyName">A string representing the key name of the
/// object to be tagged.</param>
/// <param name="filePath">The directory location and file name of the
/// object to be uploaded to the Amazon S3 bucket.</param>
public static async Task PutObjectsWithTagsAsync(IAmazonS3 client, string
bucketName, string keyName, string filePath)
{
    try
    {
        // Create an object with tags.
        var putRequest = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = keyName,
            FilePath = filePath,
            TagSet = new List<Tag>
            {
                new Tag { Key = "Keyx1", Value = "Value1" },
                new Tag { Key = "Keyx2", Value = "Value2" },
            },
        };

        PutObjectResponse response = await
client.PutObjectAsync(putRequest);

        // Now retrieve the new object's tags.
```

```
        GetObjectTaggingRequest getTagsRequest = new
GetObjectTaggingRequest()
        {
            BucketName = bucketName,
            Key = keyName,
        };

        GetObjectTaggingResponse objectTags = await
client.GetObjectTaggingAsync(getTagsRequest);

        // Display the tag values.
        objectTags.Tagging
            .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));

        Tagging newTagSet = new Tagging()
        {
            TagSet = new List<Tag>
            {
                new Tag { Key = "Key3", Value = "Value3" },
                new Tag { Key = "Key4", Value = "Value4" },
            },
        };

        PutObjectTaggingRequest putObjTagsRequest = new
PutObjectTaggingRequest()
        {
            BucketName = bucketName,
            Key = keyName,
            Tagging = newTagSet,
        };

        PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

        // Retrieve the tags again and show the values.
        GetObjectTaggingRequest getTagsRequest2 = new
GetObjectTaggingRequest()
        {
            BucketName = bucketName,
            Key = keyName,
        };

        GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);
```

```
        objectTags2.Tagging
            .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine(
            $"Error: '{ex.Message}'");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [GetObjectTagging](#) in der APIAWS SDK for .NET - Referenz für .

Sperrungen von Amazon S3-Objekten

Das folgende Codebeispiel zeigt, wie Sie mit S3-Objektsperreffunktionen arbeiten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, das die Features der Amazon S3-Objektsperre demonstriert.

```
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
```

```
namespace S3ObjectLockScenario;

public static class S3ObjectLockWorkflow
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. Create test Amazon Simple Storage Service (S3) buckets with different
        lock policies.
        2. Upload sample objects to each bucket.
        3. Set some Legal Hold and Retention Periods on objects and buckets.
        4. Investigate lock policies by viewing settings or attempting to delete or
        overwrite objects.
        5. Clean up objects and buckets.
    */

    public static S3ActionsWrapper _s3ActionsWrapper = null!;
    public static IConfiguration _configuration = null!;
    private static string _resourcePrefix = null!;
    private static string noLockBucketName = null!;
    private static string lockEnabledBucketName = null!;
    private static string retentionAfterCreationBucketName = null!;
    private static List<string> bucketNames = new List<string>();
    private static List<string> fileNames = new List<string>();

    public 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<IAmazonS3>()
                    .AddTransient<S3ActionsWrapper>()
                )
            .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();

ConfigurationSetup();

ServicesSetup(host);

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Amazon Simple Storage Service (S3)
Object Locking Workflow Scenario.");
    Console.WriteLine(new string('-', 80));
    await Setup(true);

    await DemoActionChoices();

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Cleaning up resources.");
    Console.WriteLine(new string('-', 80));
    await Cleanup(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Amazon S3 Object Locking Workflow is complete.");
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"There was a problem: {ex.Message}");
    await Cleanup(true);
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
```

```
        _s3ActionsWrapper = host.Services.GetRequiredService<S3ActionsWrapper>();
    }

    /// <summary>
    /// Any setup operations needed.
    /// </summary>
    public static void ConfigurationSetup()
    {
        _resourcePrefix = _configuration["resourcePrefix"] ?? "dotnet-example";

        noLockBucketName = _resourcePrefix + "-no-lock";
        lockEnabledBucketName = _resourcePrefix + "-lock-enabled";
        retentionAfterCreationBucketName = _resourcePrefix + "-retention-after-
creation";

        bucketNames.Add(noLockBucketName);
        bucketNames.Add(lockEnabledBucketName);
        bucketNames.Add(retentionAfterCreationBucketName);
    }

    // <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Setup(bool interactive)
    {
        Console.WriteLine(
            S3\n" +
            "\nFor this workflow, we will use the AWS SDK for .NET to create several
            "buckets and files to demonstrate working with S3 locking features.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you are ready to start.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nS3 buckets can be created either with or without object
lock enabled.");
        await _s3ActionsWrapper.CreateBucketWithObjectLock(noLockBucketName, false);
        await _s3ActionsWrapper.CreateBucketWithObjectLock(lockEnabledBucketName,
true);
    }
}
```

```
        await
        _s3ActionsWrapper.CreateBucketWithObjectLock(retentionAfterCreationBucketName,
        false);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nA bucket can be configured to use object locking with a
        default retention period.");
        await
        _s3ActionsWrapper.ModifyBucketDefaultRetention(retentionAfterCreationBucketName,
        true,
            ObjectLockRetentionMode.Governance, DateTime.UtcNow.AddDays(1));

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nObject lock policies can also be added to existing
        buckets.");
        await _s3ActionsWrapper.EnableObjectLockOnBucket(lockEnabledBucketName);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        // Upload some files to the buckets.
        Console.WriteLine("\nNow let's add some test files:");
        var fileName = _configuration["exampleFileName"] ?? "exampleFile.txt";
        int fileCount = 2;
        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for uploading to a bucket.");
        }

        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileCount; i++)
            {
```



```
        var numberedFileName = Path.GetFileNameWithoutExtension(fileName) +
i + Path.GetExtension(fileName);
        fileNames.Add(numberedFileName);
        await _s3ActionsWrapper.UploadFileAsync(bucketName,
numberedFileName, fileName);
    }
}
Console.WriteLine("Press Enter to continue.");
if (interactive)
    Console.ReadLine();

if (!interactive)
    return true;
Console.WriteLine("\nNow we can set some object lock policies on individual
files:");
foreach (var bucketName in bucketNames)
{
    for (int i = 0; i < fileNames.Count; i++)
    {
        // No modifications to the objects in the first bucket.
        if (bucketName != bucketNames[0])
        {
            var exampleFileName = fileNames[i];
            switch (i)
            {
                case 0:
                {
                    var question =
                        $"{"\nWould you like to add a legal hold to
{exampleFileName} in {bucketName}? (y/n)";
                    if (GetYesNoResponse(question))
                    {
                        // Set a legal hold.
                        await
_s3ActionsWrapper.ModifyObjectLegalHold(bucketName, exampleFileName,
ObjectLockLegalHoldStatus.0n);

                    }
                    break;
                }
                case 1:
                {
                    var question =
```

```

        $"Would you like to add a 1 day Governance
retention period to {exampleFileName} in {bucketName}? (y/n)" +
        "\nReminder: Only a user with the
s3:BypassGovernanceRetention permission will be able to delete this file or its
bucket until the retention period has expired.";
        if (GetYesNoResponse(question))
        {
            // Set a Governance mode retention period for 1
day.
            await
_s3ActionsWrapper.ModifyObjectRetentionPeriod(
                bucketName, exampleFileName,
                ObjectLockRetentionMode.Governance,
                DateTime.UtcNow.AddDays(1));
        }
        break;
    }
}
}
}
}
Console.WriteLine(new string('-', 80));
return true;
}

// <summary>
/// List all of the current buckets and objects.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>The list of buckets and objects.</returns>
public static async Task<List<S3ObjectVersion>> ListBucketsAndObjects(bool
interactive)
{
    var allObjects = new List<S3ObjectVersion>();
    foreach (var bucketName in bucketNames)
    {
        var objectsInBucket = await
_s3ActionsWrapper.ListBucketObjectsAndVersions(bucketName);
        foreach (var objectKey in objectsInBucket.Versions)
        {
            allObjects.Add(objectKey);
        }
    }
}
}

```

```
        if (interactive)
        {
            Console.WriteLine("\nCurrent buckets and objects:\n");
            int i = 0;
            foreach (var bucketObject in allObjects)
            {
                i++;
                Console.WriteLine(
                    $"{i}: {bucketObject.Key} \n\tBucket:
{bucketObject.BucketName}\n\tVersion: {bucketObject.VersionId}");
            }
        }

        return allObjects;
    }

    /// <summary>
    /// Present the user with the demo action choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<bool> DemoActionChoices()
    {
        var choices = new string[]{
            "List all files in buckets.",
            "Attempt to delete a file.",
            "Attempt to delete a file with retention period bypass.",
            "Attempt to overwrite a file.",
            "View the object and bucket retention settings for a file.",
            "View the legal hold settings for a file.",
            "Finish the workflow."};

        var choice = 0;
        // Keep asking the user until they choose to move on.
        while (choice != 6)
        {
            Console.WriteLine(new string('-', 80));
            choice = GetChoiceResponse(
                "\nExplore the S3 locking features by selecting one of the following
choices:"
                , choices);
            Console.WriteLine(new string('-', 80));
            switch (choice)
            {
                case 0:
```

```
        {
            await ListBucketsAndObjects(true);
            break;
        }
    case 1:
        {
            Console.WriteLine("\nEnter the number of the object to
delete:");

            var allFiles = await ListBucketsAndObjects(true);
            var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());

            await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, false, allFiles[fileChoice].VersionId);
            break;
        }
    case 2:
        {
            Console.WriteLine("\nEnter the number of the object to
delete:");

            var allFiles = await ListBucketsAndObjects(true);
            var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());

            await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, true, allFiles[fileChoice].VersionId);
            break;
        }
    case 3:
        {
            var allFiles = await ListBucketsAndObjects(true);
            Console.WriteLine("\nEnter the number of the object to
overwrite:");

            var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());

            // Create the file if it does not already exist.
            if (!File.Exists(allFiles[fileChoice].Key))
            {
                await using StreamWriter sw =
File.CreateText(allFiles[fileChoice].Key);
                await sw.WriteLineAsync(
                    "This is a sample file for uploading to a bucket.");
            }
        }
    }
```

```
        await
        _s3ActionsWrapper.UploadFileAsync(allFiles[fileChoice].BucketName,
        allFiles[fileChoice].Key, allFiles[fileChoice].Key);
        break;
    }
    case 4:
    {
        var allFiles = await ListBucketsAndObjects(true);
        Console.WriteLine("\nEnter the number of the object and
bucket to view:");
        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
        await
        _s3ActionsWrapper.GetObjectRetention(allFiles[fileChoice].BucketName,
        allFiles[fileChoice].Key);
        await
        _s3ActionsWrapper.GetBucketObjectLockConfiguration(allFiles[fileChoice].BucketName);
        break;
    }
    case 5:
    {
        var allFiles = await ListBucketsAndObjects(true);
        Console.WriteLine("\nEnter the number of the object to
view:");
        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
        await
        _s3ActionsWrapper.GetObjectLegalHold(allFiles[fileChoice].BucketName,
        allFiles[fileChoice].Key);
        break;
    }
    }
    }
    return true;
}

// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Cleanup(bool interactive)
{
    Console.WriteLine(new string('-', 80));
}
```

```
        if (!interactive || GetYesNoResponse("Do you want to clean up all files and
buckets? (y/n) "))
        {
            // Remove all locks and delete all buckets and objects.
            var allFiles = await ListBucketsAndObjects(false);
            foreach (var fileInfo in allFiles)
            {
                // Check for a legal hold.
                var legalHold = await
                _s3ActionsWrapper.GetObjectLegalHold(fileInfo.BucketName, fileInfo.Key);
                if (legalHold?.Status?.Value == ObjectLockLegalHoldStatus.On)
                {
                    await
                _s3ActionsWrapper.ModifyObjectLegalHold(fileInfo.BucketName, fileInfo.Key,
                ObjectLockLegalHoldStatus.Off);
                }

                // Check for a retention period.
                var retention = await
                _s3ActionsWrapper.GetObjectRetention(fileInfo.BucketName, fileInfo.Key);
                var hasRetentionPeriod = retention?.Mode ==
                ObjectLockRetentionMode.Governance && retention.RetainUntilDate >
                DateTime.UtcNow.Date;
                await _s3ActionsWrapper.DeleteObjectFromBucket(fileInfo.BucketName,
                fileInfo.Key, hasRetentionPeriod, fileInfo.VersionId);
            }

            foreach (var bucketName in bucketNames)
            {
                await _s3ActionsWrapper.DeleteBucketByName(bucketName);
            }
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
    }
}
```

```
        return true;
    }

    /// <summary>
    /// Helper method to 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;
    }

    /// <summary>
    /// Helper method to get a choice response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="choices">The choices to print on the console.</param>
    /// <returns>The index of the selected choice</returns>
    private static int GetChoiceResponse(string? question, string[] choices)
    {
        if (question != null)
        {
            Console.WriteLine(question);

            for (int i = 0; i < choices.Length; i++)
            {
                Console.WriteLine($"{i + 1}. {choices[i]}");
            }
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > choices.Length)
        {
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        return choiceNumber - 1;
    }
}
```

```
}
```

Eine Wrapper-Klasse für S3-Funktionen.

```
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

namespace S3ObjectLockScenario;

/// <summary>
/// Encapsulate the Amazon S3 operations.
/// </summary>
public class S3ActionsWrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Constructor for the S3ActionsWrapper.
    /// </summary>
    /// <param name="amazonS3">The injected S3 client.</param>
    public S3ActionsWrapper(IAmazonS3 amazonS3, IConfiguration configuration)
    {
        _amazonS3 = amazonS3;
    }

    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
enableObjectLock)
    {
        Console.WriteLine($"\\tCreating bucket {bucketName} with object lock
{enableObjectLock}.");
        try
        {
```



```
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
            ObjectLockEnabledForBucket = enableObjectLock,
        };

        var response = await _amazonS3.PutBucketAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error creating bucket: '{ex.Message}'");
        return false;
    }
}

/// <summary>
/// Enable object lock on an existing bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket to modify.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableObjectLockOnBucket(string bucketName)
{
    try
    {
        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,
                Status = VersionStatus.Enabled
            }
        });

        var request = new PutObjectLockConfigurationRequest()
        {
            BucketName = bucketName,
            ObjectLockConfiguration = new ObjectLockConfiguration()
            {
```

```

        ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
    },
};

var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
Console.WriteLine($"\\tAdded an object lock policy to bucket
{bucketName}.");
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
    return false;
}
}

/// <summary>
/// Set or modify a retention period on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date retention expires.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
    string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
{
    try
    {
        var request = new PutObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            Retention = new ObjectLockRetention()
            {
                Mode = retention,
                RetainUntilDate = retainUntilDate
            }
        };
    };

    var response = await _amazonS3.PutObjectRetentionAsync(request);
    Console.WriteLine($"\\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
}
}

```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying retention period:
'{ex.Message}');
        return false;
    }
}

/// <summary>
/// Set or modify a retention period on an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket to modify.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date for retention until.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
{
    var enabledString = enableObjectLock ? "Enabled" : "Disabled";
    var timeDifference = retainUntilDate.Subtract(DateTime.Now);
    try
    {
        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,
                Status = VersionStatus.Enabled
            }
        });

        var request = new PutObjectLockConfigurationRequest()
        {
            BucketName = bucketName,
            ObjectLockConfiguration = new ObjectLockConfiguration()
            {
                ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                Rule = new ObjectLockRule()
            }
        }
    }
}
```

```

        DefaultRetention = new DefaultRetention()
        {
            Mode = retention,
            Days = timeDifference.Days // Can be specified in days
or years but not both.
        }
    }
};

var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
Console.WriteLine($"\\tAdded a default retention to bucket
{bucketName}.");
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"\\tError modifying object lock: '{ex.Message}'");
    return false;
}
}

/// <summary>
/// Get the retention period for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object retention details.</returns>
public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectRetentionAsync(request);
        Console.WriteLine($"\\tObject retention for {objectKey} in {bucketName}:
" +
            $"\\n\\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");

```

```
        return response.Retention;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to fetch object lock retention:
'{ex.Message}'");
        return new ObjectLockRetention();
    }
}

/// <summary>
/// Set or modify a legal hold on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="holdStatus">The On or Off status for the legal hold.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectLegalHold(string bucketName,
    string objectKey, ObjectLockLegalHoldStatus holdStatus)
{
    try
    {
        var request = new PutObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            LegalHold = new ObjectLockLegalHold()
            {
                Status = holdStatus
            }
        };

        var response = await _amazonS3.PutObjectLegalHoldAsync(request);
        Console.WriteLine($"\\tModified legal hold for {objectKey} in
{bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying legal hold: '{ex.Message}'");
        return false;
    }
}
```

```
/// <summary>
/// Get the legal hold details for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object legal hold details.</returns>
public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectLegalHoldAsync(request);
        Console.WriteLine($"{\tObject legal hold for {objectKey} in {bucketName}:
" +
            $"\n\tStatus: {response.LegalHold.Status}");
        return response.LegalHold;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"{\tUnable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}

/// <summary>
/// Get the object lock configuration details for an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket to get details.</param>
/// <returns>The bucket's object lock configuration details.</returns>
public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
{
    try
    {
        var request = new GetObjectLockConfigurationRequest()
        {
            BucketName = bucketName
        };
    }
}
```

```
        var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
        Console.WriteLine($"{\tBucket object lock config for {bucketName} in
{bucketName}: " +
            $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
            $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

        return response.ObjectLockConfiguration;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"{\tUnable to fetch object lock config:
'{ex.Message}'");
        return new ObjectLockConfiguration();
    }
}

/// <summary>
/// Upload a file from the local computer to an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to use.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object to
upload.</param>
/// <returns>True if success.</returns>
public async Task<bool> UploadFileAsync(string bucketName, string objectName,
string filePath)
{
    var request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
        FilePath = filePath,
        ChecksumAlgorithm = ChecksumAlgorithm.SHA256
    };

    var response = await _amazonS3.PutObjectAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"{\tSuccessfully uploaded {objectName} to
{bucketName}.");
        return true;
    }
}
```

```
    }
    else
    {
        Console.WriteLine($"\\tCould not upload {objectName} to {bucketName}.");
        return false;
    }
}

/// <summary>
/// List bucket objects and versions.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to use.</param>
/// <returns>The list of objects and versions.</returns>
public async Task<ListVersionsResponse> ListBucketObjectsAndVersions(string
bucketName)
{
    var request = new ListVersionsRequest()
    {
        BucketName = bucketName
    };

    var response = await _amazonS3.ListVersionsAsync(request);
    return response;
}

/// <summary>
/// Delete an object from a specific bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to use.</param>
/// <param name="objectKey">The key of the object to delete.</param>
/// <param name="hasRetention">True if the object has retention settings.</
param>
/// <param name="versionId">Optional versionId.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteObjectFromBucket(string bucketName, string
objectKey, bool hasRetention, string? versionId = null)
{
    try
    {
        var request = new DeleteObjectRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            VersionId = versionId,
```



```
};
if (hasRetention)
{
    // Set the BypassGovernanceRetention header
    // if the file has retention settings.
    request.BypassGovernanceRetention = true;
}
await _amazonS3.DeleteObjectAsync(request);
Console.WriteLine(
    $"Deleted {objectKey} in {bucketName}.");
return true;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Unable to delete object {objectKey} in bucket
{bucketName}: " + ex.Message);
    return false;
}
}

/// <summary>
/// Delete a specific bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to use.</param>
/// <param name="objectKey">The key of the object to delete.</param>
/// <param name="versionId">Optional versionId.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteBucketByName(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest() { BucketName = bucketName, };
        var response = await _amazonS3.DeleteBucketAsync(request);
        Console.WriteLine($"Delete for {bucketName} complete.");
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Unable to delete bucket {bucketName}: " +
ex.Message);
        return false;
    }
}
}
```

```
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetention](#)
 - [PutObjectLegalHold](#)
 - [PutObjectLockConfiguration](#)
 - [PutObjectRetention](#)

Verwalten von Zugriffssteuerungslisten (ACL)

Das folgende Codebeispiel zeigt, wie Sie eine neue Zugriffssteuerungsliste (ACL) für Amazon-S3-Buckets verwalten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to manage Amazon Simple Storage Service
/// (Amazon S3) access control lists (ACLs) to control Amazon S3 bucket
/// access.
/// </summary>
public class ManageACLs
{
    public static async Task Main()
```

```
{
    string bucketName = "doc-example-bucket1";
    string newBucketName = "doc-example-bucket2";
    string keyName = "sample-object.txt";
    string emailAddress = "someone@example.com";

    // If the AWS Region where your bucket is located is different from
    // the Region defined for the default user, pass the Amazon S3 bucket's
    // name to the client constructor. It should look like this:
    // RegionEndpoint bucketRegion = RegionEndpoint.USEast1;
    IAmazonS3 client = new AmazonS3Client();

    await TestBucketObjectACLsAsync(client, bucketName, newBucketName,
keyName, emailAddress);
}

/// <summary>
/// Creates a new Amazon S3 bucket with a canned ACL, then retrieves the ACL
/// information and then adds a new ACL to one of the objects in the
/// Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// methods to create a bucket, get an ACL, and add a different ACL to
/// one of the objects.</param>
/// <param name="bucketName">A string representing the original Amazon S3
/// bucket name.</param>
/// <param name="newBucketName">A string representing the name of the
/// new bucket that will be created.</param>
/// <param name="keyName">A string representing the key name of an Amazon S3
/// object for which we will change the ACL.</param>
/// <param name="emailAddress">A string representing the email address
/// belonging to the person to whom access to the Amazon S3 bucket will be
/// granted.</param>
public static async Task TestBucketObjectACLsAsync(
    IAmazonS3 client,
    string bucketName,
    string newBucketName,
    string keyName,
    string emailAddress)
{
    try
    {
        // Create a new Amazon S3 bucket and specify canned ACL.
```

```
        var success = await CreateBucketWithCannedACLAsync(client,
newBucketName);

        // Get the ACL on a bucket.
        await GetBucketACLAsync(client, bucketName);

        // Add (replace) the ACL on an object in a bucket.
        await AddACLToExistingObjectAsync(client, bucketName, keyName,
emailAddress);
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        Console.WriteLine($"Exception: {amazonS3Exception.Message}");
    }
}

/// <summary>
/// Creates a new Amazon S3 bucket with a canned ACL attached.
/// </summary>
/// <param name="client">The initialized client object used to call
/// PutBucketAsync.</param>
/// <param name="newBucketName">A string representing the name of the
/// new Amazon S3 bucket.</param>
/// <returns>Returns a boolean value indicating success or failure.</
returns>
public static async Task<bool> CreateBucketWithCannedACLAsync(IAmazonS3
client, string newBucketName)
{
    var request = new PutBucketRequest()
    {
        BucketName = newBucketName,
        BucketRegion = S3Region.EUWest1,

        // Add a canned ACL.
        CannedACL = S3CannedACL.LogDeliveryWrite,
    };

    var response = await client.PutBucketAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Retrieves the ACL associated with the Amazon S3 bucket name in the
```

```

    /// bucketName parameter.
    /// </summary>
    /// <param name="client">The initialized client object used to call
    /// PutBucketAsync.</param>
    /// <param name="bucketName">The Amazon S3 bucket for which we want to get
the
    /// ACL list.</param>
    /// <returns>Returns an S3AccessControlList returned from the call to
    /// GetACLAsync.</returns>
    public static async Task<S3AccessControlList> GetBucketACLAsync(IAmazonS3
client, string bucketName)
    {
        GetACLResponse response = await client.GetACLAsync(new GetACLRequest
        {
            BucketName = bucketName,
        });

        return response.AccessControlList;
    }

    /// <summary>
    /// Adds a new ACL to an existing object in the Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized client object used to call
    /// PutBucketAsync.</param>
    /// <param name="bucketName">A string representing the name of the Amazon S3
    /// bucket containing the object to which we want to apply a new ACL.</
param>
    /// <param name="keyName">A string representing the name of the object
    /// to which we want to apply the new ACL.</param>
    /// <param name="emailAddress">The email address of the person to whom
    /// we will be applying to whom access will be granted.</param>
    public static async Task AddACLToExistingObjectAsync(IAmazonS3 client,
string bucketName, string keyName, string emailAddress)
    {
        // Retrieve the ACL for an object.
        GetACLResponse aclResponse = await client.GetACLAsync(new GetACLRequest
        {
            BucketName = bucketName,
            Key = keyName,
        });

```

```
S3AccessControllist acl = aclResponse.AccessControllist;

// Retrieve the owner.
Owner owner = acl.Owner;

// Clear existing grants.
acl.Grants.Clear();

// Add a grant to reset the owner's full permission
// (the previous clear statement removed all permissions).
var fullControlGrant = new S3Grant
{
    Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id },
};
acl.AddGrant(fullControlGrant.Grantee, S3Permission.FULL_CONTROL);

// Specify email to identify grantee for granting permissions.
var grantUsingEmail = new S3Grant
{
    Grantee = new S3Grantee { EmailAddress = emailAddress },
    Permission = S3Permission.WRITE_ACP,
};

// Specify log delivery group as grantee.
var grantLogDeliveryGroup = new S3Grant
{
    Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" },
    Permission = S3Permission.WRITE,
};

// Create a new ACL.
var newAcl = new S3AccessControllist
{
    Grants = new List<S3Grant> { grantUsingEmail,
grantLogDeliveryGroup },
    Owner = owner,
};

// Set the new ACL. We're throwing away the response here.
_ = await client.PutACLAsync(new PutACLRequest
{
    BucketName = bucketName,
    Key = keyName,
```

```
        AccessControllList = newAcl,  
    });  
}  
  
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [GetBucketAcl](#)
 - [GetObjectAcl](#)
 - [PutBucketAcl](#)
 - [PutObjectAcl](#)

Erstellen einer mehrteiligen Kopie

Das folgende Codebeispiel zeigt, wie Sie eine mehrteilige Kopie eines Amazon-S3-Objekts erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Amazon.S3;  
using Amazon.S3.Model;  
  
/// <summary>  
/// This example shows how to perform a multi-part copy from one Amazon  
/// Simple Storage Service (Amazon S3) bucket to another.  
/// </summary>  
public class MPUapiCopyObj  
{  
    private const string SourceBucket = "doc-example-bucket1";  
    private const string TargetBucket = "doc-example-bucket2";
```

```
private const string SourceObjectKey = "example.mov";
private const string TargetObjectKey = "copied_video_file.mov";

/// <summary>
/// This method starts the multi-part upload.
/// </summary>
public static async Task Main()
{
    var s3Client = new AmazonS3Client();
    Console.WriteLine("Copying object...");
    await MPUCopyObjectAsync(s3Client);
}

/// <summary>
/// This method uses the passed client object to perform a multipart
/// copy operation.
/// </summary>
/// <param name="client">An Amazon S3 client object that will be used
/// to perform the copy.</param>
public static async Task MPUCopyObjectAsync(AmazonS3Client client)
{
    // Create a list to store the copy part responses.
    var copyResponses = new List<CopyPartResponse>();

    // Setup information required to initiate the multipart upload.
    var initiateRequest = new InitiateMultipartUploadRequest
    {
        BucketName = TargetBucket,
        Key = TargetObjectKey,
    };

    // Initiate the upload.
    InitiateMultipartUploadResponse initResponse =
        await client.InitiateMultipartUploadAsync(initiateRequest);

    // Save the upload ID.
    string uploadId = initResponse.UploadId;

    try
    {
        // Get the size of the object.
        var metadataRequest = new GetObjectMetadataRequest
        {
            BucketName = SourceBucket,
```



```
        Key = SourceObjectKey,
    };

    GetObjectMetadataResponse metadataResponse =
        await client.GetObjectMetadataAsync(metadataRequest);
    var objectSize = metadataResponse.ContentLength; // Length in bytes.

    // Copy the parts.
    var partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        var copyRequest = new CopyPartRequest
        {
            DestinationBucket = TargetBucket,
            DestinationKey = TargetObjectKey,
            SourceBucket = SourceBucket,
            SourceKey = SourceObjectKey,
            UploadId = uploadId,
            FirstByte = bytePosition,
            LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
            PartNumber = i,
        };

        copyResponses.Add(await client.CopyPartAsync(copyRequest));

        bytePosition += partSize;
    }

    // Set up to complete the copy.
    var completeRequest = new CompleteMultipartUploadRequest
    {
        BucketName = TargetBucket,
        Key = TargetObjectKey,
        UploadId = initResponse.UploadId,
    };
    completeRequest.AddPartETags(copyResponses);

    // Complete the copy.
    CompleteMultipartUploadResponse completeUploadResponse =
        await client.CompleteMultipartUploadAsync(completeRequest);
    }
```

```
        catch (AmazonS3Exception e)
        {
            Console.WriteLine($"Error encountered on server.
Message: '{e.Message}' when writing an object");
        }
        catch (Exception e)
        {
            Console.WriteLine($"Unknown encountered on server.
Message: '{e.Message}' when writing an object");
        }
    }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CompleteMultipartUpload](#)
 - [CreateMultipartUpload](#)
 - [GetObjectMetadata](#)
 - [UploadPartCopy](#)

Hoch- oder Herunterladen großer Dateien

Das folgende Codebeispiel zeigt, wie Sie große Dateien zu und von Amazon S3 hoch- oder herunterladen.

Weitere Informationen finden Sie unter [Hochladen eines Objekts mit Multipart-Upload](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie Funktionen auf, die Dateien mithilfe von Amazon S3 TransferUtility3 in und aus einem S3-Bucket übertragen.

```
global using System.Text;
```

```
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.S3.Transfer;
global using TransferUtilityBasics;

// This Amazon S3 client uses the default user credentials
// defined for this computer.
using Microsoft.Extensions.Configuration;

IAmazonS3 client = new AmazonS3Client();
var transferUtil = new TransferUtility(client);
IConfiguration _configuration;

_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();

// Edit the values in settings.json to use an S3 bucket and files that
// exist on your AWS account and on the local computer where you
// run this scenario.
var bucketName = _configuration["BucketName"];
var localPath =
    $"{Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)}\
\TransferFolder";

DisplayInstructions();

PressEnter();

Console.WriteLine();

// Upload a single file to an S3 bucket.
DisplayTitle("Upload a single file");

var fileToUpload = _configuration["FileToUpload"];
Console.WriteLine($"Uploading {fileToUpload} to the S3 bucket, {bucketName}.");

var success = await TransferMethods.UploadSingleFileAsync(transferUtil, bucketName,
    fileToUpload, localPath);
```

```
if (success)
{
    Console.WriteLine($"Successfully uploaded the file, {fileToUpload} to
{bucketName}.");
}

PressEnter();

// Upload a local directory to an S3 bucket.
DisplayTitle("Upload all files from a local directory");
Console.WriteLine("Upload all the files in a local folder to an S3 bucket.");
const string keyPrefix = "UploadFolder";
var uploadPath = $"{localPath}\\UploadFolder";

Console.WriteLine($"Uploading the files in {uploadPath} to {bucketName}");
DisplayTitle($"{uploadPath} files");
DisplayLocalFiles(uploadPath);
Console.WriteLine();

PressEnter();

success = await TransferMethods.UploadFullDirectoryAsync(transferUtil, bucketName,
    keyPrefix, uploadPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the files in {uploadPath} to
{bucketName}.");
    Console.WriteLine($"{bucketName} currently contains the following files:");
    await DisplayBucketFiles(client, bucketName, keyPrefix);
    Console.WriteLine();
}

PressEnter();

// Download a single file from an S3 bucket.
DisplayTitle("Download a single file");
Console.WriteLine("Now we will download a single file from an S3 bucket.");

var keyName = _configuration["FileToDownload"];

Console.WriteLine($"Downloading {keyName} from {bucketName}.");

success = await TransferMethods.DownloadSingleFileAsync(transferUtil, bucketName,
    keyName, localPath);
```

```
if (success)
{
    Console.WriteLine($"Successfully downloaded the file, {keyName} from
{bucketName}.");
}

PressEnter();

// Download the contents of a directory from an S3 bucket.
DisplayTitle("Download the contents of an S3 bucket");
var s3Path = _configuration["S3Path"];
var downloadPath = $"{localPath}\\{s3Path}";

Console.WriteLine($"Downloading the contents of {bucketName}\\{s3Path}");
Console.WriteLine($"{bucketName}\\{s3Path} contains the following files:");
await DisplayBucketFiles(client, bucketName, s3Path);
Console.WriteLine();

success = await TransferMethods.DownloadS3DirectoryAsync(transferUtil, bucketName,
s3Path, downloadPath);
if (success)
{
    Console.WriteLine($"Downloaded the files in {bucketName} to {downloadPath}.");
    Console.WriteLine($"{downloadPath} now contains the following files:");
    DisplayLocalFiles(downloadPath);
}

Console.WriteLine("\nThe TransferUtility Basics application has completed.");
PressEnter();

// Displays the title for a section of the scenario.
static void DisplayTitle(string titleText)
{
    var sepBar = new string('-', Console.WindowWidth);

    Console.WriteLine(sepBar);
    Console.WriteLine(CenterText(titleText));
    Console.WriteLine(sepBar);
}

// Displays a description of the actions to be performed by the scenario.
static void DisplayInstructions()
{
    var sepBar = new string('-', Console.WindowWidth);
```

```
    DisplayTitle("Amazon S3 Transfer Utility Basics");
    Console.WriteLine("This program shows how to use the Amazon S3 Transfer
Utility.");
    Console.WriteLine("It performs the following actions:");
    Console.WriteLine("\t1. Upload a single object to an S3 bucket.");
    Console.WriteLine("\t2. Upload an entire directory from the local computer to an
\n\t S3 bucket.");
    Console.WriteLine("\t3. Download a single object from an S3 bucket.");
    Console.WriteLine("\t4. Download the objects in an S3 bucket to a local
directory.");
    Console.WriteLine($" \n{sepBar}");
}

// Pauses the scenario.
static void PressEnter()
{
    Console.WriteLine("Press <Enter> to continue.");
    _ = Console.ReadLine();
    Console.WriteLine("\n");
}

// Returns the string textToCenter, padded on the left with spaces
// that center the text on the console display.
static string CenterText(string textToCenter)
{
    var centeredText = new StringBuilder();
    var screenWidth = Console.WindowWidth;
    centeredText.Append(new string(' ', (int)(screenWidth - textToCenter.Length) /
2));
    centeredText.Append(textToCenter);
    return centeredText.ToString();
}

// Displays a list of file names included in the specified path.
static void DisplayLocalFiles(string localPath)
{
    var fileList = Directory.GetFiles(localPath);
    if (fileList.Length > 0)
    {
        foreach (var fileName in fileList)
        {
            Console.WriteLine(fileName);
        }
    }
}
```

```
    }  
  }  
  
  // Displays a list of the files in the specified S3 bucket and prefix.  
  static async Task DisplayBucketFiles(IAmazonS3 client, string bucketName, string  
    s3Path)  
  {  
    ListObjectsV2Request request = new()  
    {  
      BucketName = bucketName,  
      Prefix = s3Path,  
      MaxKeys = 5,  
    };  
  
    var response = new ListObjectsV2Response();  
  
    do  
    {  
      response = await client.ListObjectsV2Async(request);  
  
      response.S3Objects  
        .ForEach(obj => Console.WriteLine($"{obj.Key}"));  
  
      // If the response is truncated, set the request ContinuationToken  
      // from the NextContinuationToken property of the response.  
      request.ContinuationToken = response.NextContinuationToken;  
    } while (response.IsTruncated);  
  }  
}
```

Laden Sie eine einzelne Datei hoch.

```
/// <summary>  
/// Uploads a single file from the local computer to an S3 bucket.  
/// </summary>  
/// <param name="transferUtil">The transfer initialized TransferUtility  
/// object.</param>  
/// <param name="bucketName">The name of the S3 bucket where the file  
/// will be stored.</param>  
/// <param name="fileName">The name of the file to upload.</param>  
/// <param name="localPath">The local path where the file is stored.</param>
```

```
/// <returns>A boolean value indicating the success of the action.</returns>
public static async Task<bool> UploadSingleFileAsync(
    TransferUtility transferUtil,
    string bucketName,
    string fileName,
    string localPath)
{
    if (File.Exists($"{localPath}\\{fileName}"))
    {
        try
        {
            await transferUtil.UploadAsync(new TransferUtilityUploadRequest
            {
                BucketName = bucketName,
                Key = fileName,
                FilePath = $"{localPath}\\{fileName}",
            });

            return true;
        }
        catch (AmazonS3Exception s3Ex)
        {
            Console.WriteLine($"Could not upload {fileName} from {localPath}
because:");
            Console.WriteLine(s3Ex.Message);
            return false;
        }
    }
    else
    {
        Console.WriteLine($"{fileName} does not exist in {localPath}");
        return false;
    }
}
```

Laden Sie ein ganzes lokales Verzeichnis hoch.

```
/// <summary>
/// Uploads all the files in a local directory to a directory in an S3
/// bucket.
/// </summary>
```



```
    /// <param name="transferUtil">The transfer initialized TransferUtility
    /// object.</param>
    /// <param name="bucketName">The name of the S3 bucket where the files
    /// will be stored.</param>
    /// <param name="keyPrefix">The key prefix is the S3 directory where
    /// the files will be stored.</param>
    /// <param name="localPath">The local directory that contains the files
    /// to be uploaded.</param>
    /// <returns>A Boolean value representing the success of the action.</
returns>
    public static async Task<bool> UploadFullDirectoryAsync(
        TransferUtility transferUtil,
        string bucketName,
        string keyPrefix,
        string localPath)
    {
        if (Directory.Exists(localPath))
        {
            try
            {
                await transferUtil.UploadDirectoryAsync(new
TransferUtilityUploadDirectoryRequest
                {
                    BucketName = bucketName,
                    KeyPrefix = keyPrefix,
                    Directory = localPath,
                });

                return true;
            }
            catch (AmazonS3Exception s3Ex)
            {
                Console.WriteLine($"Can't upload the contents of {localPath}
because:");
                Console.WriteLine(s3Ex?.Message);
                return false;
            }
        }
        else
        {
            Console.WriteLine($"The directory {localPath} does not exist.");
            return false;
        }
    }
}
```

Laden Sie eine einzelne Datei herunter.

```
/// <summary>
/// Download a single file from an S3 bucket to the local computer.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
/// object.</param>
/// <param name="bucketName">The name of the S3 bucket containing the
/// file to download.</param>
/// <param name="keyName">The name of the file to download.</param>
/// <param name="localPath">The path on the local computer where the
/// downloaded file will be saved.</param>
/// <returns>A Boolean value indicating the results of the action.</returns>
public static async Task<bool> DownloadSingleFileAsync(
    TransferUtility transferUtil,
    string bucketName,
    string keyName,
    string localPath)
{
    await transferUtil.DownloadAsync(new TransferUtilityDownloadRequest
    {
        BucketName = bucketName,
        Key = keyName,
        FilePath = $"{localPath}\\{keyName}",
    });

    return (File.Exists($"{localPath}\\{keyName}"));
}
```

Laden Sie den Inhalt eines S3 Buckets herunter.

```
/// <summary>
/// Downloads the contents of a directory in an S3 bucket to a
/// directory on the local computer.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
```

```
    /// object.</param>
    /// <param name="bucketName">The bucket containing the files to download.</
param>
    /// <param name="s3Path">The S3 directory where the files are located.</
param>
    /// <param name="localPath">The local path to which the files will be
    /// saved.</param>
    /// <returns>A Boolean value representing the success of the action.</
returns>
    public static async Task<bool> DownloadS3DirectoryAsync(
        TransferUtility transferUtil,
        string bucketName,
        string s3Path,
        string localPath)
    {
        int fileCount = 0;

        // If the directory doesn't exist, it will be created.
        if (Directory.Exists(s3Path))
        {
            var files = Directory.GetFiles(localPath);
            fileCount = files.Length;
        }

        await transferUtil.DownloadDirectoryAsync(new
TransferUtilityDownloadDirectoryRequest
        {
            BucketName = bucketName,
            LocalDirectory = localPath,
            S3Directory = s3Path,
        });

        if (Directory.Exists(localPath))
        {
            var files = Directory.GetFiles(localPath);
            if (files.Length > fileCount)
            {
                return true;
            }

            // No change in the number of files. Assume
            // the download failed.
            return false;
        }
    }
}
```

```
    // The local directory doesn't exist. No files
    // were downloaded.
    return false;
}
```

Verfolgen Sie den Fortschritt eines Uploads mithilfe der TransferUtility.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to track the progress of a multipart upload
/// using the Amazon Simple Storage Service (Amazon S3) TransferUtility to
/// upload to an Amazon S3 bucket.
/// </summary>
public class TrackMPUUsingHighLevelAPI
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "sample_pic.png";
        string path = "filepath/directory/";
        string filePath = $"{path}{keyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        await TrackMPUAsync(client, bucketName, filePath, keyName);
    }

    /// <summary>
    /// Starts an Amazon S3 multipart upload and assigns an event handler to
    /// track the progress of the upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
```

```
/// perform the multipart upload.</param>
/// <param name="bucketName">The name of the bucket to which to upload
/// the file.</param>
/// <param name="filePath">The path, including the file name of the
/// file to be uploaded to the Amazon S3 bucket.</param>
/// <param name="keyName">The file name to be used in the
/// destination Amazon S3 bucket.</param>
public static async Task TrackMPUAsync(
    IAmazonS3 client,
    string bucketName,
    string filePath,
    string keyName)
{
    try
    {
        var fileTransferUtility = new TransferUtility(client);

        // Use TransferUtilityUploadRequest to configure options.
        // In this example we subscribe to an event.
        var uploadRequest =
            new TransferUtilityUploadRequest
            {
                BucketName = bucketName,
                FilePath = filePath,
                Key = keyName,
            };

        uploadRequest.UploadProgressEvent +=
            new EventHandler<UploadProgressArgs>(
                UploadRequest_UploadPartProgressEvent);

        await fileTransferUtility.UploadAsync(uploadRequest);
        Console.WriteLine("Upload completed");
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error:: {ex.Message}");
    }
}

/// <summary>
/// Event handler to check the progress of the multipart upload.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
```

```
    /// <param name="e">The object that contains multipart upload
    /// information.</param>
    public static void UploadRequest_UploadPartProgressEvent(object sender,
UploadProgressArgs e)
    {
        // Process event.
        Console.WriteLine($"{e.TransferredBytes}/{e.TotalBytes}");
    }
}
```

Laden Sie ein Objekt mit Verschlüsselung hoch.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Uses the Amazon Simple Storage Service (Amazon S3) low level API to
/// perform a multipart upload to an Amazon S3 bucket.
/// </summary>
public class SSECLowLevelMPUcopyObject
{
    public static async Task Main()
    {
        string existingBucketName = "doc-example-bucket";
        string sourceKeyName = "sample_file.txt";
        string targetKeyName = "sample_file_copy.txt";
        string filePath = $"sample\\{targetKeyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USEast1.
        IAmazonS3 client = new AmazonS3Client();

        // Create the encryption key.
        var base64Key = CreateEncryptionKey();
    }
}
```

```
        await CreateSampleObjUsingClientEncryptionKeyAsync(
            client,
            existingBucketName,
            sourceKeyName,
            filePath,
            base64Key);
    }

    /// <summary>
    /// Creates the encryption key to use with the multipart upload.
    /// </summary>
    /// <returns>A string containing the base64-encoded key for encrypting
    /// the multipart upload.</returns>
    public static string CreateEncryptionKey()
    {
        Aes aesEncryption = Aes.Create();
        aesEncryption.KeySize = 256;
        aesEncryption.GenerateKey();
        string base64Key = Convert.ToBase64String(aesEncryption.Key);
        return base64Key;
    }

    /// <summary>
    /// Creates and uploads an object using a multipart upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 object used to
    /// initialize and perform the multipart upload.</param>
    /// <param name="existingBucketName">The name of the bucket to which
    /// the object will be uploaded.</param>
    /// <param name="sourceKeyName">The source object name.</param>
    /// <param name="filePath">The location of the source object.</param>
    /// <param name="base64Key">The encryption key to use with the upload.</
param>
    public static async Task CreateSampleObjUsingClientEncryptionKeyAsync(
        IAmazonS3 client,
        string existingBucketName,
        string sourceKeyName,
        string filePath,
        string base64Key)
    {
        List<UploadPartResponse> uploadResponses = new
List<UploadPartResponse>();
```

```
        InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
    {
        BucketName = existingBucketName,
        Key = sourceKeyName,
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key,
    };

    InitiateMultipartUploadResponse initResponse =
        await client.InitiateMultipartUploadAsync(initiateRequest);

    long contentLength = new FileInfo(filePath).Length;
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

    try
    {
        long filePosition = 0;
        for (int i = 1; filePosition < contentLength; i++)
        {
            UploadPartRequest uploadRequest = new UploadPartRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                UploadId = initResponse.UploadId,
                PartNumber = i,
                PartSize = partSize,
                FilePosition = filePosition,
                FilePath = filePath,
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            // Upload part and add response to our list.
            uploadResponses.Add(await
client.UploadPartAsync(uploadRequest));

            filePosition += partSize;
        }

        CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
```



```
        {
            BucketName = existingBucketName,
            Key = sourceKeyName,
            UploadId = initResponse.UploadId,
        };
        completeRequest.AddPartETags(uploadResponses);

        CompleteMultipartUploadResponse completeUploadResponse =
            await client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (Exception exception)
    {
        Console.WriteLine($"Exception occurred: {exception.Message}");

        // If there was an error, abort the multipart upload.
        AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
    {
        BucketName = existingBucketName,
        Key = sourceKeyName,
        UploadId = initResponse.UploadId,
    };

        await client.AbortMultipartUploadAsync(abortMPURequest);
    }
}
}
```

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von .NET

```
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                }
            }
        }
    }
}
```

```
        return string.Empty;
    }

    var bucket = evt.Records[0].S3.Bucket.Name;
    var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

    context.Logger.LogLine($"Request is for {bucket} and {key}");

    var objectResult = await _s3Client.GetObjectAsync(bucket, key);

    context.Logger.LogLine($"Returning {objectResult.Key}");

    return objectResult.Key;
}
catch (Exception e)
{
    context.Logger.LogLine($"Error processing request - {e.Message}");

    return string.Empty;
}
}
}
```

S3-Glacier-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit S3 Glacier verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Amazon S3 Glacier

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon S3 Glacier.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.Glacier;
using Amazon.Glacier.Model;

namespace GlacierActions;

public static class HelloGlacier
{
    static async Task Main()
    {
        var glacierService = new AmazonGlacierClient();

        Console.WriteLine("Hello Amazon Glacier!");
        Console.WriteLine("Let's list your Glacier vaults:");

        // You can use await and any of the async methods to get a response.
        // Let's get the vaults using a paginator.
        var glacierVaultPaginator = glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            Console.WriteLine($"{vault.CreationDate}:{vault.VaultName}, ARN:
{vault.VaultARN}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListVaults](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)

Aktionen

Tags hinzufügen

Im folgenden Codebeispiel wird veranschaulicht, wie Sie einem Amazon-S3-Glacier-Tresor Tags hinzufügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add tags to the items in an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to add tags to.</param>
/// <param name="key">The name of the object to tag.</param>
/// <param name="value">The tag value to add.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AddTagsToVaultAsync(string vaultName, string key, string
value)
{
    var request = new AddTagsToVaultRequest
    {
        Tags = new Dictionary<string, string>
        {
            { key, value },
        },
        AccountId = "-",
        VaultName = vaultName,
    };
};
```

```
var response = await _glacierService.AddTagsToVaultAsync(request);
return response.HttpStatusCode == HttpStatusCode.NoContent;
}
```

- Weitere API-Informationen finden Sie unter [AddTagsToVault](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen eines Tresors

Das folgende Codebeispiel zeigt, wie Sie einen Amazon S3-Glacier-Tresor erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to create.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateVaultAsync(string vaultName)
{
    var request = new CreateVaultRequest
    {
        // Setting the AccountId to "-" means that
        // the account associated with the current
        // account will be used.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.CreateVaultAsync(request);

    Console.WriteLine($"Created {vaultName} at: {response.Location}");
}
```

```
        return response.HttpStatusCode == HttpStatusCode.Created;
    }
```

- Weitere API-Informationen finden Sie unter [CreateVault](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben eines Tresors

Das folgende Code-Beispiel zeigt, wie Sie einen Amazon-S3-Glacier-Tresor beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Describe an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to describe.</param>
/// <returns>The Amazon Resource Name (ARN) of the vault.</returns>
public async Task<string> DescribeVaultAsync(string vaultName)
{
    var request = new DescribeVaultRequest
    {
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.DescribeVaultAsync(request);

    // Display the information about the vault.
    Console.WriteLine($"{response.VaultName}\tARN: {response.VaultARN}");
    Console.WriteLine($"Created on: {response.CreationDate}\tNumber of Archives:
{response.NumberOfArchives}\tSize (in bytes): {response.SizeInBytes}");
    if (response.LastInventoryDate != DateTime.MinValue)
    {
        Console.WriteLine($"Last inventory: {response.LastInventoryDate}");
    }
}
```

```
    }  
  
    return response.VaultARN;  
}
```

- Weitere API-Informationen finden Sie unter [DescribeVault](#) in der APIAWS SDK for .NET - Referenz für .

Herunterladen eines Archivs

Das folgende Codebeispiel zeigt, wie Sie ein Amazon-S3-Glacier-Archiv herunterladen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird die - ArchiveTransferManager Klasse verwendet. API-Details finden Sie unter [ArchiveTransferManager](#).

```
/// <summary>  
/// Download an archive from an Amazon S3 Glacier vault using the Archive  
/// Transfer Manager.  
/// </summary>  
/// <param name="vaultName">The name of the vault containing the object.</param>  
/// <param name="archiveId">The Id of the archive to download.</param>  
/// <param name="localFilePath">The local directory where the file will  
/// be stored after download.</param>  
/// <returns>Async Task.</returns>  
public async Task<bool> DownloadArchiveWithArchiveManagerAsync(string vaultName,  
string archiveId, string localFilePath)  
{  
    try  
    {  
        var manager = new ArchiveTransferManager(_glacierService);
```



```
var options = new DownloadOptions
{
    StreamTransferProgress = Progress!,
};

// Download an archive.
Console.WriteLine("Initiating the archive retrieval job and then polling
SQS queue for the archive to be available.");
Console.WriteLine("When the archive is available, downloading will
begin.");
await manager.DownloadAsync(vaultName, archiveId, localFilePath,
options);

return true;
}
catch (AmazonGlacierException ex)
{
    Console.WriteLine(ex.Message);
    return false;
}
}

/// <summary>
/// Event handler to track the progress of the Archive Transfer Manager.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="args">The argument values from the object that raised the
/// event.</param>
static void Progress(object sender, StreamTransferProgressArgs args)
{
    if (args.PercentDone != _currentPercentage)
    {
        _currentPercentage = args.PercentDone;
        Console.WriteLine($"Downloaded {_currentPercentage}%");
    }
}
}
```

Auflisten von Aufträgen

Das folgende Codebeispiel zeigt, wie Sie Amazon S3-Glacier-Aufträge auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List Amazon S3 Glacier jobs.
/// </summary>
/// <param name="vaultName">The name of the vault to list jobs for.</param>
/// <returns>A list of Amazon S3 Glacier jobs.</returns>
public async Task<List<GlacierJobDescription>> ListJobsAsync(string vaultName)
{
    var request = new ListJobsRequest
    {
        // Using a hyphen "-" for the Account Id will
        // cause the SDK to use the Account Id associated
        // with the current account.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.ListJobsAsync(request);

    return response.JobList;
}
```

- Weitere API-Informationen finden Sie unter [ListJobs](#) in der APIAWS SDK for .NET -Referenz für

Auflisten von Tags

Das folgende Codebeispiel zeigt, wie Sie Tags für einen Amazon-S3-Glacier-Tresor auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List tags for an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to list tags for.</param>
/// <returns>A dictionary listing the tags attached to each object in the
/// vault and its tags.</returns>
public async Task<Dictionary<string, string>> ListTagsForVaultAsync(string
vaultName)
{
    var request = new ListTagsForVaultRequest
    {
        // Using a hyphen "-" for the Account Id will
        // cause the SDK to use the Account Id associated
        // with the default user.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.ListTagsForVaultAsync(request);

    return response.Tags;
}
```

- Weitere API-Informationen finden Sie unter [ListTagsForVault](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Tresoren

Das folgende Codebeispiel zeigt, wie Sie Amazon S3-Glacier-Tresore auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List the Amazon S3 Glacier vaults associated with the current account.
/// </summary>
/// <returns>A list containing information about each vault.</returns>
public async Task<List<DescribeVaultOutput>> ListVaultsAsync()
{
    var glacierVaultPaginator = _glacierService.Paginators.ListVaults(
        new ListVaultsRequest { AccountId = "-" });
    var vaultList = new List<DescribeVaultOutput>();

    await foreach (var vault in glacierVaultPaginator.VaultList)
    {
        vaultList.Add(vault);
    }

    return vaultList;
}
```

- Weitere API-Informationen finden Sie unter [ListVaults](#) in der APIAWS SDK for .NET -Referenz für .

Hochladen eines Archivs in einen Tresor

Das folgende Codebeispiel zeigt, wie Sie ein Archiv in einen Amazon S3-Glacier-Tresor hochladen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Upload an object to an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the Amazon S3 Glacier vault to upload
/// the archive to.</param>
/// <param name="archiveFilePath">The file path of the archive to upload to the
vault.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<string> UploadArchiveWithArchiveManager(string vaultName,
string archiveFilePath)
{
    try
    {
        var manager = new ArchiveTransferManager(_glacierService);

        // Upload an archive.
        var response = await manager.UploadAsync(vaultName, "upload archive
test", archiveFilePath);
        return response.ArchiveId;
    }
    catch (AmazonGlacierException ex)
    {
        Console.WriteLine(ex.Message);
        return string.Empty;
    }
}
```

- Weitere API-Informationen finden Sie unter [UploadArchive](#) in der APIAWS SDK for .NET - Referenz für .

SageMaker -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden SageMaker.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo SageMaker

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von SageMaker beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.SageMaker;
using Amazon.SageMaker.Model;

namespace SageMakerActions;

public static class HelloSageMaker
{
    static async Task Main(string[] args)
    {
```

```
var sagemakerClient = new AmazonSageMakerClient();

Console.WriteLine($"Hello Amazon SageMaker! Let's list some of your notebook
instances:");
Console.WriteLine();

// You can use await and any of the async methods to get a response.
// Let's get the first five notebook instances.
var response = await sagemakerClient.ListNotebookInstancesAsync(
    new ListNotebookInstancesRequest()
    {
        MaxResults = 5
    });

if (!response.NotebookInstances.Any())
{
    Console.WriteLine($"No notebook instances found.");
    Console.WriteLine("See https://docs.aws.amazon.com/sagemaker/latest/dg/
howitworks-create-ws.html to create one.");
}

foreach (var notebookInstance in response.NotebookInstances)
{
    Console.WriteLine($"Instance:
{notebookInstance.NotebookInstanceName}");
    Console.WriteLine($"Arn: {notebookInstance.NotebookInstanceArn}");
    Console.WriteLine($"Creation Date:
{notebookInstance.CreationTime.ToShortDateString()}");
    Console.WriteLine();
}
}
```

- Weitere API-Informationen finden Sie unter [ListNotebookInstances](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen Sie eine Pipeline

Das folgende Codebeispiel zeigt, wie Sie eine Pipeline in erstellen oder aktualisieren SageMaker.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a pipeline from a JSON definition, or update it if the pipeline
already exists.
/// </summary>
/// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
{
    try
    {
        var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
            new UpdatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
                PipelineName = name,
                RoleArn = roleArn
            });
        return updateResponse.PipelineArn;
    }
    catch (Amazon.SageMaker.Model.ResourceNotFoundException)
    {
        var createResponse = await _amazonSageMaker.CreatePipelineAsync(
            new CreatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
```



```
        PipelineName = name,
        RoleArn = roleArn
    });

    return createResponse.PipelineArn;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreatePipeline](#)
 - [UpdatePipeline](#)

Pipeline löschen

Das folgende Codebeispiel zeigt, wie Sie eine Pipeline in löschen SageMaker.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a SageMaker pipeline by name.
/// </summary>
/// <param name="pipelineName">The name of the pipeline to delete.</param>
/// <returns>The ARN of the pipeline.</returns>
public async Task<string> DeletePipelineByName(string pipelineName)
{
    var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
        new DeletePipelineRequest()
        {
            PipelineName = pipelineName
        });

    return deleteResponse.PipelineArn;
}
```

- Weitere API-Informationen finden Sie unter [DeletePipeline](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben einer Pipeline-Ausführung

Das folgende Codebeispiel zeigt, wie Sie eine Pipeline-Ausführung in beschreiben SageMaker.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Check the status of a run.
/// </summary>
/// <param name="pipelineExecutionArn">The ARN.</param>
/// <returns>The status of the pipeline.</returns>
public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
{
    var describeResponse = await
    _amazonSageMaker.DescribePipelineExecutionAsync(
        new DescribePipelineExecutionRequest()
        {
            PipelineExecutionArn = pipelineExecutionArn
        });

    return describeResponse.PipelineExecutionStatus;
}
```

- Weitere API-Informationen finden Sie unter [DescribePipelineExecution](#) in der APIAWS SDK for .NET -Referenz für .

Ausführen einer Pipeline

Das folgende Codebeispiel zeigt, wie Sie eine Pipeline-Ausführung in starten SageMaker.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Run a pipeline with input and output file locations.
/// </summary>
/// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
/// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
/// <param name="outputLocationUrl">The output location in Amazon S3.</param>
/// <param name="pipelineName">The name of the pipeline.</param>
/// <param name="executionRoleArn">The ARN of the role.</param>
/// <returns>The ARN of the pipeline run.</returns>
public async Task<string> ExecutePipeline(
    string queueUrl,
    string inputLocationUrl,
    string outputLocationUrl,
    string pipelineName,
    string executionRoleArn)
{
    var inputConfig = new VectorEnrichmentJobInputConfig()
    {
        DataSourceConfig = new()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = inputLocationUrl
            }
        },
        DocumentType = VectorEnrichmentJobDocumentType.CSV
    };

    var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
```

```
{
    S3Data = new VectorEnrichmentJobS3Data()
    {
        S3Uri = outputLocationUrl
    }
};

var jobConfig = new VectorEnrichmentJobConfig()
{
    ReverseGeocodingConfig = new ReverseGeocodingConfig()
    {
        XAttributeName = "Longitude",
        YAttributeName = "Latitude"
    }
};

#pragma warning disable SageMaker1002 // Property value does not match required
pattern is allowed here to match the pipeline definition.
var startExecutionResponse = await
_amazonSageMaker.StartPipelineExecutionAsync(
    new StartPipelineExecutionRequest()
    {
        PipelineName = pipelineName,
        PipelineExecutionDisplayName = pipelineName + "-example-execution",
        PipelineParameters = new List<Parameter>()
        {
            new Parameter() { Name = "parameter_execution_role", Value =
executionRoleArn },
            new Parameter() { Name = "parameter_queue_url", Value =
queueUrl },
            new Parameter() { Name = "parameter_vej_input_config", Value =
JsonSerializer.Serialize(inputConfig) },
            new Parameter() { Name = "parameter_vej_export_config", Value =
JsonSerializer.Serialize(exportConfig) },
            new Parameter() { Name = "parameter_step_1_vej_config", Value =
JsonSerializer.Serialize(jobConfig) }
        }
    });
#pragma warning restore SageMaker1002
return startExecutionResponse.PipelineExecutionArn;
}
```

- Weitere API-Informationen finden Sie unter [StartPipelineExecution](#) in der APIAWS SDK for .NET -Referenz für .

Szenarien

Erste Schritte mit Geodatenaufträgen und Pipelines

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Richten Sie Ressourcen für eine Pipeline ein.
- Richten Sie eine Pipeline ein, die einen Geodatenauftrag ausführt.
- Pipeline-Ausführung starten.
- Überwachen Sie den Status der Ausführung.
- Zeigen Sie die Ausgabe der Pipeline an.
- Bereinigen Sie Ressourcen.

Weitere Informationen finden Sie unter [Erstellen und Ausführen von SageMaker Pipelines mit - AWS SDKs auf Community.aws](#).

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Klasse, die - SageMaker Operationen umschließt.

```
using System.Text.Json;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

namespace SageMakerActions;

///
```

```
/// Wrapper class for Amazon SageMaker actions and logic.
/// </summary>
public class SageMakerWrapper
{
    private readonly IAmazonSageMaker _amazonSageMaker;
    public SageMakerWrapper(IAmazonSageMaker amazonSageMaker)
    {
        _amazonSageMaker = amazonSageMaker;
    }

    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
    already exists.
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return createResponse.PipelineArn;
        }
    }
}
```

```
    }
}

/// <summary>
/// Run a pipeline with input and output file locations.
/// </summary>
/// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
/// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
/// <param name="outputLocationUrl">The output location in Amazon S3.</param>
/// <param name="pipelineName">The name of the pipeline.</param>
/// <param name="executionRoleArn">The ARN of the role.</param>
/// <returns>The ARN of the pipeline run.</returns>
public async Task<string> ExecutePipeline(
    string queueUrl,
    string inputLocationUrl,
    string outputLocationUrl,
    string pipelineName,
    string executionRoleArn)
{
    var inputConfig = new VectorEnrichmentJobInputConfig()
    {
        DataSourceConfig = new()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = inputLocationUrl
            }
        },
        DocumentType = VectorEnrichmentJobDocumentType.CSV
    };

    var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
    {
        S3Data = new VectorEnrichmentJobS3Data()
        {
            S3Uri = outputLocationUrl
        }
    };

    var jobConfig = new VectorEnrichmentJobConfig()
    {
        ReverseGeocodingConfig = new ReverseGeocodingConfig()
    }
}
```

```

        {
            XAttributeName = "Longitude",
            YAttributeName = "Latitude"
        }
    };

#pragma warning disable SageMaker1002 // Property value does not match required
pattern is allowed here to match the pipeline definition.
    var startExecutionResponse = await
    _amazonSageMaker.StartPipelineExecutionAsync(
        new StartPipelineExecutionRequest()
        {
            PipelineName = pipelineName,
            PipelineExecutionDisplayName = pipelineName + "-example-execution",
            PipelineParameters = new List<Parameter>()
            {
                new Parameter() { Name = "parameter_execution_role", Value =
executionRoleArn },
                new Parameter() { Name = "parameter_queue_url", Value =
queueUrl },
                new Parameter() { Name = "parameter_vej_input_config", Value =
JsonSerializer.Serialize(inputConfig) },
                new Parameter() { Name = "parameter_vej_export_config", Value =
JsonSerializer.Serialize(exportConfig) },
                new Parameter() { Name = "parameter_step_1_vej_config", Value =
JsonSerializer.Serialize(jobConfig) }
            }
        });
#pragma warning restore SageMaker1002
    return startExecutionResponse.PipelineExecutionArn;
}

/// <summary>
/// Check the status of a run.
/// </summary>
/// <param name="pipelineExecutionArn">The ARN.</param>
/// <returns>The status of the pipeline.</returns>
public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
{
    var describeResponse = await
    _amazonSageMaker.DescribePipelineExecutionAsync(
        new DescribePipelineExecutionRequest()
        {

```



```

        PipelineExecutionArn = pipelineExecutionArn
    });

    return describeResponse.PipelineExecutionStatus;
}

/// <summary>
/// Delete a SageMaker pipeline by name.
/// </summary>
/// <param name="pipelineName">The name of the pipeline to delete.</param>
/// <returns>The ARN of the pipeline.</returns>
public async Task<string> DeletePipelineByName(string pipelineName)
{
    var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
        new DeletePipelineRequest()
        {
            PipelineName = pipelineName
        });

    return deleteResponse.PipelineArn;
}
}

```

Erstellen Sie eine Funktion, die Rückrufe aus der SageMaker Pipeline verarbeitet.

```

using System.Text.Json;
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

// Assembly attribute to enable the AWS Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SageMakerLambda;

/// <summary>

```

```
/// The AWS Lambda function handler for the Amazon SageMaker pipeline.
/// </summary>
public class SageMakerLambdaFunction
{
    /// <summary>
    /// Default constructor. This constructor is used by AWS Lambda to construct the
    instance. When invoked in a Lambda environment
    /// the AWS credentials will come from the AWS Identity and Access Management
    (IAM) role associated with the function. The AWS Region will be set to the
    /// Region that the Lambda function is running in.
    /// </summary>
    public SageMakerLambdaFunction()
    {
    }

    /// <summary>
    /// The AWS Lambda function handler that processes events from the SageMaker
    pipeline and starts a job or export.
    /// </summary>
    /// <param name="request">The custom SageMaker pipeline request object.</param>
    /// <param name="context">The Lambda context.</param>
    /// <returns>The dictionary of output parameters.</returns>
    public async Task<Dictionary<string, string>> FunctionHandler(PipelineRequest
request, ILambdaContext context)
    {
        var geoSpatialClient = new AmazonSageMakerGeospatialClient();
        var sageMakerClient = new AmazonSageMakerClient();
        var responseDictionary = new Dictionary<string, string>();
        context.Logger.LogInformation("Function handler started with request: " +
JsonSerializer.Serialize(request));
        if (request.Records != null && request.Records.Any())
        {
            context.Logger.LogInformation("Records found, this is a queue event.
Processing the queue records.");
            foreach (var message in request.Records)
            {
                await ProcessMessageAsync(message, context, geoSpatialClient,
sageMakerClient);
            }
        }
        else if (!string.IsNullOrEmpty(request.vej_export_config))
        {
            context.Logger.LogInformation("Export configuration found, this is an
export. Start the Vector Enrichment Job (VEJ) export.");
        }
    }
}
```

```
var outputConfig =
    JsonSerializer.Deserialize<ExportVectorEnrichmentJobOutputConfig>(
        request.vej_export_config);

var exportResponse = await
geoSpatialClient.ExportVectorEnrichmentJobAsync(
    new ExportVectorEnrichmentJobRequest()
    {
        Arn = request.vej_arn,
        ExecutionRoleArn = request.Role,
        OutputConfig = outputConfig
    });
context.Logger.LogInformation($"Export response:
{JsonSerializer.Serialize(exportResponse)}");
responseDictionary = new Dictionary<string, string>
{
    { "export_eoj_status", exportResponse.ExportStatus.ToString() },
    { "vej_arn", exportResponse.Arn }
};
}
else if (!string.IsNullOrEmpty(request.vej_name))
{
    context.Logger.LogInformation("Vector Enrichment Job name found,
starting the job.");
    var inputConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobInputConfig>(
            request.vej_input_config);

    var jobConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobConfig>(
            request.vej_config);

    var jobResponse = await geoSpatialClient.StartVectorEnrichmentJobAsync(
        new StartVectorEnrichmentJobRequest()
        {
            ExecutionRoleArn = request.Role,
            InputConfig = inputConfig,
            Name = request.vej_name,
            JobConfig = jobConfig
        });
    context.Logger.LogInformation("Job response: " +
        JsonSerializer.Serialize(jobResponse));
```

```
        responseDictionary = new Dictionary<string, string>
        {
            { "vej_arn", jobResponse.Arn },
            { "statusCode", jobResponse.HttpStatusCode.ToString() }
        };
    }
    return responseDictionary;
}

/// <summary>
/// Process a queue message and check the status of a SageMaker job.
/// </summary>
/// <param name="message">The queue message.</param>
/// <param name="context">The Lambda context.</param>
/// <param name="geoClient">The SageMaker GeoSpatial client.</param>
/// <param name="sageMakerClient">The SageMaker client.</param>
/// <returns>Async task.</returns>
private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context,
    AmazonSageMakerGeospatialClient geoClient, AmazonSageMakerClient
sageMakerClient)
{
    context.Logger.LogInformation($"Processed message {message.Body}");

    // Get information about the SageMaker job.
    var payload = JsonSerializer.Deserialize<QueuePayload>(message.Body);
    context.Logger.LogInformation($"Payload token {payload!.token}");
    var token = payload.token;

    if (payload.arguments.ContainsKey("vej_arn"))
    {
        // Use the job ARN and the token to get the job status.
        var job_arn = payload.arguments["vej_arn"];
        context.Logger.LogInformation($"Token: {token}, arn {job_arn}");

        var jobInfo = geoClient.GetVectorEnrichmentJobAsync(
            new GetVectorEnrichmentJobRequest()
            {
                Arn = job_arn
            });
        context.Logger.LogInformation("Job info: " +
            JsonSerializer.Serialize(jobInfo));
        if (jobInfo.Result.Status == VectorEnrichmentJobStatus.COMPLETED)
        {
```

```
        context.Logger.LogInformation($"Status completed, resuming
pipeline...");
        await sageMakerClient.SendPipelineExecutionStepSuccessAsync(
            new SendPipelineExecutionStepSuccessRequest()
            {
                CallbackToken = token,
                OutputParameters = new List<OutputParameter>()
                {
                    new OutputParameter()
                    { Name = "export_status", Value =
jobInfo.Result.Status }
                }
            });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.FAILED)
    {
        context.Logger.LogInformation($"Status failed, stopping
pipeline...");
        await sageMakerClient.SendPipelineExecutionStepFailureAsync(
            new SendPipelineExecutionStepFailureRequest()
            {
                CallbackToken = token,
                FailureReason = jobInfo.Result.ErrorDetails.ErrorMessage
            });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.IN_PROGRESS)
    {
        // Put this message back in the queue to reprocess later.
        context.Logger.LogInformation(
            $"Status still in progress, check back later.");
        throw new("Job still running.");
    }
}
}
```

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
public static class PipelineWorkflow
{
    public static IAmazonIdentityManagementService _iamClient = null!;
    public static SageMakerWrapper _sageMakerWrapper = null!;
```

```
public static IAmazonSQS _sqsClient = null!;  
public static IAmazonS3 _s3Client = null!;  
public static IAmazonLambda _lambdaClient = null!;  
public static IConfiguration _configuration = null!;  
  
public static string lambdaFunctionName = "SageMakerExampleFunction";  
public static string sageMakerRoleName = "SageMakerExampleRole";  
public static string lambdaRoleName = "SageMakerExampleLambdaRole";  
  
private static string[] lambdaRolePolicies = null!;  
private static string[] sageMakerRolePolicies = null!;  
  
static async Task Main(string[] args)  
{  
    var options = new AWSOptions() { Region = RegionEndpoint.USWest2 };  
    // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>(options)  
                .AddAWSService<IAmazonEC2>(options)  
                .AddAWSService<IAmazonSageMaker>(options)  
                .AddAWSService<IAmazonSageMakerGeospatial>(options)  
                .AddAWSService<IAmazonSQS>(options)  
                .AddAWSService<IAmazonS3>(options)  
                .AddAWSService<IAmazonLambda>(options)  
                .AddTransient<SageMakerWrapper>()  
            )  
        .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();  
  
    ServicesSetup(host);  
    string queueUrl = "";  
    string queueName = _configuration["queueName"];
```

```
string bucketName = _configuration["bucketName"];
var pipelineName = _configuration["pipelineName"];

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "Welcome to the Amazon SageMaker pipeline example scenario.");
    Console.WriteLine(
        "\nThis example workflow will guide you through setting up and
running an" +
        "\nAmazon SageMaker pipeline. The pipeline uses an AWS Lambda
function and an" +
        "\nAmazon SQS Queue. It runs a vector enrichment reverse geocode job
to" +
        "\nreverse geocode addresses in an input file and store the results
in an export file.");
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "First, we will set up the roles, functions, and queue needed by the
SageMaker pipeline.");
    Console.WriteLine(new string('-', 80));

    var lambdaRoleArn = await CreateLambdaRole();
    var sageMakerRoleArn = await CreateSageMakerRole();
    var functionArn = await SetupLambda(lambdaRoleArn, true);
    queueUrl = await SetupQueue(queueName);
    await SetupBucket(bucketName);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Now we can create and run our pipeline.");
    Console.WriteLine(new string('-', 80));

    await SetupPipeline(sageMakerRoleArn, functionArn, pipelineName);
    var executionArn = await ExecutePipeline(queueUrl, sageMakerRoleArn,
pipelineName, bucketName);
    await WaitForPipelineExecution(executionArn);

    await GetOutputResults(bucketName);

    Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("The pipeline has completed. To view the pipeline and
runs " +
                        "in SageMaker Studio, follow these instructions:" +
                        "\nhttps://docs.aws.amazon.com/sagemaker/latest/dg/
pipelines-studio.html");
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Finally, let's clean up our resources.");
        Console.WriteLine(new string('-', 80));

        await CleanupResources(true, queueUrl, pipelineName, bucketName);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("SageMaker pipeline scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources(true, queueUrl, pipelineName, bucketName);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _sageMakerWrapper = host.Services.GetRequiredService<SageMakerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _sqsClient = host.Services.GetRequiredService<IAmazonSQS>();
    _s3Client = host.Services.GetRequiredService<IAmazonS3>();
    _lambdaClient = host.Services.GetRequiredService<IAmazonLambda>();
}

/// <summary>
/// Set up AWS Lambda, either by updating an existing function or creating a new
function.
```



```
/// </summary>
/// <param name="roleArn">The role Amazon Resource Name (ARN) to use for the
Lambda function.</param>
/// <param name="askUser">True to ask the user before updating.</param>
/// <returns>The ARN of the function.</returns>
public static async Task<string> SetupLambda(string roleArn, bool askUser)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Setting up the Lambda function for the pipeline.");
    var handlerName =
"SageMakerLambda::SageMakerLambda.SageMakerLambdaFunction::FunctionHandler";
    var functionArn = "";
    try
    {
        var functionInfo = await _lambdaClient.GetFunctionAsync(new
GetFunctionRequest()
        {
            FunctionName = lambdaFunctionName
        });

        var updateFunction = true;
        if (askUser)
        {
            updateFunction = GetYesNoResponse(
                $"{\tThe Lambda function {lambdaFunctionName} already exists, do
you want to update it?");
        }

        if (updateFunction)
        {
            // Update the Lambda function.
            using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
            await _lambdaClient.UpdateFunctionCodeAsync(
                new UpdateFunctionCodeRequest()
                {
                    FunctionName = lambdaFunctionName,
                    ZipFile = zipMemoryStream,
                });
        }

        functionArn = functionInfo.Configuration.FunctionArn;
    }
    catch (ResourceNotFoundException)
```

```

    {
        Console.WriteLine($"\\tThe Lambda function {lambdaFunctionName} was not
found, creating the new function.");

        // Create the function if it does not already exist.
        using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
        var createResult = await _lambdaClient.CreateFunctionAsync(
            new CreateFunctionRequest()
            {
                FunctionName = lambdaFunctionName,
                Runtime = Runtime.Dotnet6,
                Description = "SageMaker example function.",
                Code = new FunctionCode()
                {
                    ZipFile = zipMemoryStream
                },
                Handler = handlerName,
                Role = roleArn,
                Timeout = 30
            });

        functionArn = createResult.FunctionArn;
    }

    Console.WriteLine($"\\tLambda ready with ARN {functionArn}.");
    Console.WriteLine(new string('-', 80));
    return functionArn;
}

/// <summary>
/// Create a role to be used by AWS Lambda. Does not create the role if it
already exists.
/// </summary>
/// <returns>The role ARN.</returns>
public static async Task<string> CreateLambdaRole()
{
    Console.WriteLine(new string('-', 80));

    lambdaRolePolicies = new string[]{
        "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
        "arn:aws:iam::aws:policy/AmazonSQSFullAccess",
        "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerGeospatialFullAccess",

```

```

        "arn:aws:iam::aws:policy/service-role/" +
        "AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy",
        "arn:aws:iam::aws:policy/service-role/" +
        "AWSLambdaSQSQueueExecutionRole"
    };

    var roleArn = await GetRoleArnIfExists(lambdaRoleName);
    if (!string.IsNullOrEmpty(roleArn))
    {
        return roleArn;
    }

    Console.WriteLine("\tCreating a role to for AWS Lambda to use.");

    var assumeRolePolicy = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\"," +
            "\"Principal\": {" +
                $"\"Service\": [" +
                    "\"sagemaker.amazonaws.com\"," +
                    "\"sagemaker-geospatial.amazonaws.com" +
                    "\", " +
                    "\"lambda.amazonaws.com\"," +
                    "\"s3.amazonaws.com\"" +
                "]" +
            "}," +
            "\"Action\": \"sts:AssumeRole\"" +
        "}]}" +
        ";

    var roleResult = await _iamClient!.CreateRoleAsync(
        new CreateRoleRequest()
        {
            AssumeRolePolicyDocument = assumeRolePolicy,
            Path = "/",
            RoleName = lambdaRoleName
        });
    foreach (var policy in lambdaRolePolicies)
    {
        await _iamClient.AttachRolePolicyAsync(
            new AttachRolePolicyRequest()
            {
                PolicyArn = policy,

```

```

        roleName = lambdaRoleName
    });
}

// Allow time for the role to be ready.
Thread.Sleep(10000);
Console.WriteLine($"\\tRole ready with ARN {roleResult.Role.Arn}.");
Console.WriteLine(new string('-', 80));

return roleResult.Role.Arn;
}

/// <summary>
/// Create a role to be used by SageMaker.
/// </summary>
/// <returns>The role Amazon Resource Name (ARN).</returns>
public static async Task<string> CreateSageMakerRole()
{
    Console.WriteLine(new string('-', 80));

    sageMakerRolePolicies = new string[]{
        "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
        "arn:aws:iam::aws:policy/AmazonSageMakerGeospatialFullAccess",
    };

    var roleArn = await GetRoleArnIfExists(sageMakerRoleName);
    if (!string.IsNullOrEmpty(roleArn))
    {
        return roleArn;
    }

    Console.WriteLine("\\tCreating a role to use with SageMaker.");

    var assumeRolePolicy = "{" +
        "\\\"Version\\\": \\\"2012-10-17\\\", \" +
        "\\\"Statement\\\": [{" +
            "\\\"Effect\\\": \\\"Allow\\\", \" +
            "\\\"Principal\\\": {\" +
                $"\\\"Service\\\": [\" +
                    "\\\"sagemaker.amazonaws.com\\\", \" +
                    "\\\"sagemaker-
geospatial.amazonaws.com\\\", \" +
                    "\\\"lambda.amazonaws.com\\\", \" +

```

```

        "\s3.amazonaws.com\" +
        "]" +
        }," +
        "\"Action\": \"sts:AssumeRole\" +
        "]" +
        }";

var roleResult = await _iamClient!.CreateRoleAsync(
    new CreateRoleRequest()
    {
        AssumeRolePolicyDocument = assumeRolePolicy,
        Path = "/",
        RoleName = sageMakerRoleName
    });

foreach (var policy in sageMakerRolePolicies)
{
    await _iamClient.AttachRolePolicyAsync(
        new AttachRolePolicyRequest()
        {
            PolicyArn = policy,
            RoleName = sageMakerRoleName
        });
}

// Allow time for the role to be ready.
Thread.Sleep(10000);
Console.WriteLine($"Role ready with ARN {roleResult.Role.Arn}.");
Console.WriteLine(new string('-', 80));
return roleResult.Role.Arn;
}

/// <summary>
/// Set up the SQS queue to use with the pipeline.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <returns>The URL for the queue.</returns>
public static async Task<string> SetupQueue(string queueName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up queue {queueName}.");

    try
    {

```

```
        var queueInfo = await _sqsClient.GetQueueUrlAsync(new
GetQueueUrlRequest()
    { QueueName = queueName });
        return queueInfo.QueueUrl;
    }
    catch (QueueDoesNotExistException)
    {
        var attrs = new Dictionary<string, string>
        {
            {
                QueueAttributeName.DelaySeconds,
                "5"
            },
            {
                QueueAttributeName.ReceiveMessageWaitTimeSeconds,
                "5"
            },
            {
                QueueAttributeName.VisibilityTimeout,
                "300"
            },
        };

        var request = new CreateQueueRequest
        {
            Attributes = attrs,
            QueueName = queueName,
        };

        var response = await _sqsClient.CreateQueueAsync(request);
        Thread.Sleep(10000);
        await ConnectLambda(response.QueueUrl);
        Console.WriteLine($"\\tQueue ready with Url {response.QueueUrl}.");
        Console.WriteLine(new string('-', 80));
        return response.QueueUrl;
    }
}

/// <summary>
/// Connect the queue to the Lambda function as an event source.
/// </summary>
/// <param name="queueUrl">The URL for the queue.</param>
/// <returns>Async task.</returns>
public static async Task ConnectLambda(string queueUrl)
```

```
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Connecting the Lambda function and queue for the
pipeline.");

    var queueAttributes = await _sqsClient.GetQueueAttributesAsync(
        new GetQueueAttributesRequest() { QueueUrl = queueUrl, AttributeNames =
new List<string>() { "All" } });
    var queueArn = queueAttributes.QueueARN;

    var eventSource = await _lambdaClient.ListEventSourceMappingsAsync(
        new ListEventSourceMappingsRequest()
        {
            FunctionName = lambdaFunctionName
        });

    if (!eventSource.EventSourceMappings.Any())
    {
        // Only add the event source mapping if it does not already exist.
        await _lambdaClient.CreateEventSourceMappingAsync(
            new CreateEventSourceMappingRequest()
            {
                EventSourceArn = queueArn,
                FunctionName = lambdaFunctionName,
                Enabled = true
            });
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up the bucket to use for pipeline input and output.
/// </summary>
/// <param name="bucketName">The name for the bucket.</param>
/// <returns>Async task.</returns>
public static async Task SetupBucket(string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up bucket {bucketName}.");

    var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
    bucketName);
}
```

```
    if (!bucketExists)
    {
        await _s3Client.PutBucketAsync(new PutBucketRequest()
        {
            BucketName = bucketName,
            BucketRegion = S3Region.USWest2
        });

        Thread.Sleep(5000);

        await _s3Client.PutObjectAsync(new PutObjectRequest()
        {
            BucketName = bucketName,
            Key = "samplefiles/latlongtest.csv",
            FilePath = "latlongtest.csv"
        });
    }

    Console.WriteLine($"\\tBucket {bucketName} ready.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Display some results from the output directory.
/// </summary>
/// <param name="bucketName">The name for the bucket.</param>
/// <returns>Async task.</returns>
public static async Task<string> GetOutputResults(string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Getting output results {bucketName}.");
    string outputKey = "";
    Thread.Sleep(15000);
    var outputFiles = await _s3Client.ListObjectsAsync(
        new ListObjectsRequest()
        {
            BucketName = bucketName,
            Prefix = "outputfiles/"
        });

    if (outputFiles.S3Objects.Any())
    {
```



```

        var sampleOutput = outputFiles.S3Objects.OrderBy(s =>
s.LastModified).Last();
        Console.WriteLine($"\\tOutput file: {sampleOutput.Key}");
        var outputSampleResponse = await _s3Client.GetObjectAsync(
            new GetObjectRequest()
            {
                BucketName = bucketName,
                Key = sampleOutput.Key
            });
        outputKey = sampleOutput.Key;
        StreamReader reader = new
StreamReader(outputSampleResponse.ResponseStream);
        await reader.ReadLineAsync();
        Console.WriteLine("\\tOutput file contents: \\n");
        for (int i = 0; i < 10; i++)
        {
            if (!reader.EndOfStream)
            {
                Console.WriteLine("\\t" + await reader.ReadLineAsync());
            }
        }
    }

    Console.WriteLine(new string('-', 80));
    return outputKey;
}

/// <summary>
/// Create a pipeline from the example pipeline JSON
/// that includes the Lambda, callback, processing, and export jobs.
/// </summary>
/// <param name="roleArn">The ARN of the role for the pipeline.</param>
/// <param name="functionArn">The ARN of the Lambda function for the pipeline.</
param>
/// <param name="pipelineName">The name for the pipeline.</param>
/// <returns>The ARN of the pipeline.</returns>
public static async Task<string> SetupPipeline(string roleArn, string
functionArn, string pipelineName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up the pipeline.");

    var pipelineJson = await File.ReadAllTextAsync("GeoSpatialPipeline.json");

```

```
// Add the correct function ARN instead of the placeholder.
pipelineJson = pipelineJson.Replace("*FUNCTION_ARN*", functionArn);

var pipelineArn = await _sageMakerWrapper.SetupPipeline(pipelineJson,
roleArn, pipelineName,
    "sdk example pipeline", pipelineName);

Console.WriteLine($"\\tPipeline set up with ARN {pipelineArn}.");
Console.WriteLine(new string('-', 80));

return pipelineArn;
}

/// <summary>
/// Start a pipeline run with job configurations.
/// </summary>
/// <param name="queueUrl">The URL for the queue used in the pipeline.</param>
/// <param name="roleArn">The ARN of the role.</param>
/// <param name="pipelineName">The name of the pipeline.</param>
/// <param name="bucketName">The name of the bucket.</param>
/// <returns>The pipeline run ARN.</returns>
public static async Task<string> ExecutePipeline(
    string queueUrl,
    string roleArn,
    string pipelineName,
    string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Starting pipeline execution.");

    var input = $"s3://{bucketName}/samplefiles/latlongtest.csv";
    var output = $"s3://{bucketName}/outputfiles/";

    var executionARN =
        await _sageMakerWrapper.ExecutePipeline(queueUrl, input, output,
            pipelineName, roleArn);

    Console.WriteLine($"\\tRun started with ARN {executionARN}.");
    Console.WriteLine(new string('-', 80));

    return executionARN;
}

/// <summary>
```

```

/// Wait for a pipeline run to complete.
/// </summary>
/// <param name="executionArn">The pipeline run ARN.</param>
/// <returns>Async task.</returns>
public static async Task WaitForPipelineExecution(string executionArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Waiting for pipeline to finish.");

    PipelineExecutionStatus status;
    do
    {
        status = await
_sageMakerWrapper.CheckPipelineExecutionStatus(executionArn);
        Thread.Sleep(30000);
        Console.WriteLine($"\\tStatus is {status}.");
    } while (status == PipelineExecutionStatus.Executing);

    Console.WriteLine($"\\tPipeline finished with status {status}.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="askUser">True to ask the user for cleanup.</param>
/// <param name="queueUrl">The URL of the queue to clean up.</param>
/// <param name="pipelineName">The name of the pipeline.</param>
/// <param name="bucketName">The name of the bucket.</param>
/// <returns>Async task.</returns>
public static async Task<bool> CleanupResources(
    bool askUser,
    string queueUrl,
    string pipelineName,
    string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    if (!askUser || GetYesNoResponse($"\\tDelete pipeline {pipelineName}? (y/
n)"))
    {
        Console.WriteLine($"\\tDeleting pipeline.");
        // Delete the pipeline.

```

```
        await _sageMakerWrapper.DeletePipelineByName(pipelineName);
    }

    if (!string.IsNullOrEmpty(queueUrl) && (!askUser ||
GetYesNoResponse($"\\tDelete queue {queueUrl}? (y/n)")))
    {
        Console.WriteLine($"\\tDeleting queue.");
        // Delete the queue.
        await _sqsClient.DeleteQueueAsync(new DeleteQueueRequest(queueUrl));
    }

    if (!askUser || GetYesNoResponse($"\\tDelete Amazon S3 bucket {bucketName}?
(y/n)"))
    {
        Console.WriteLine($"\\tDeleting bucket.");
        // Delete all objects in the bucket.
        var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
        {
            BucketName = bucketName
        });
        if (deleteList.KeyCount > 0)
        {
            await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.S3Objects
                    .Select(o => new KeyVersion { Key = o.Key }).ToList()
            });
        }

        // Now delete the bucket.
        await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
        {
            BucketName = bucketName
        });
    }

    if (!askUser || GetYesNoResponse($"\\tDelete lambda {lambdaFunctionName}? (y/
n)"))
    {
        Console.WriteLine($"\\tDeleting lambda function.");

        await _lambdaClient.DeleteFunctionAsync(new DeleteFunctionRequest()
```

```
        {
            FunctionName = lambdaFunctionName
        });
    }

    if (!askUser || GetYesNoResponse($"\tDelete role {lambdaRoleName}? (y/n)"))
    {
        Console.WriteLine($" \tDetaching policies and deleting role.");

        foreach (var policy in lambdaRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = lambdaRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = lambdaRoleName
        });
    }

    if (!askUser || GetYesNoResponse($" \tDelete role {sageMakerRoleName}? (y/
n)"))
    {
        Console.WriteLine($" \tDetaching policies and deleting role.");

        foreach (var policy in sageMakerRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = sageMakerRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = sageMakerRoleName
        });
    }
}
```

```
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

/// <summary>
/// Helper method to get a role's ARN if it already exists.
/// </summary>
/// <param name="roleName">The name of the AWS Identity and Access Management
(IAM) Role to look for.</param>
/// <returns>The role ARN if it exists, otherwise an empty string.</returns>
private static async Task<string> GetRoleArnIfExists(string roleName)
{
    Console.WriteLine($"Checking for role named {roleName}.");

    try
    {
        var existingRole = await _iamClient.GetRoleAsync(new GetRoleRequest()
        {
            RoleName = roleName
        });
        return existingRole.Role.Arn;
    }
    catch (NoSuchEntityException)
    {
        return string.Empty;
    }
}

/// <summary>
/// Helper method to 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;
}
```

```
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Secrets-Manager-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Secrets Manager verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Abrufen eines Secret-Werts

Das folgende Codebeispiel veranschaulicht, wie Sie einen Secrets-Manager-Geheimniswert abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.SecretsManager;
using Amazon.SecretsManager.Model;

/// <summary>
/// This example uses the Amazon Web Service Secrets Manager to retrieve
/// the secret value for the provided secret name.
/// </summary>
public class GetSecretValue
{
    /// <summary>
    /// The main method initializes the necessary values and then calls
    /// the GetSecretAsync and DecodeString methods to get the decoded
    /// secret value for the secret named in secretName.
    /// </summary>
    public static async Task Main()
    {
        string secretName = "<<{{MySecretName}}>>";
        string secret;

        IAmazonSecretsManager client = new AmazonSecretsManagerClient();

        var response = await GetSecretAsync(client, secretName);

        if (response is not null)
        {
            secret = DecodeString(response);

            if (!string.IsNullOrEmpty(secret))
            {
                Console.WriteLine($"The decoded secret value is: {secret}.");
            }
        }
    }
}
```



```
        else
        {
            Console.WriteLine("No secret value was returned.");
        }
    }
}

/// <summary>
/// Retrieves the secret value given the name of the secret to
/// retrieve.
/// </summary>
/// <param name="client">The client object used to retrieve the secret
/// value for the given secret name.</param>
/// <param name="secretName">The name of the secret value to retrieve.</
param>
/// <returns>The GetSecretValueResponse object returned by
/// GetSecretValueAsync.</returns>
public static async Task<GetSecretValueResponse> GetSecretAsync(
    IAmazonSecretsManager client,
    string secretName)
{
    GetSecretValueRequest request = new GetSecretValueRequest()
    {
        SecretId = secretName,
        VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT
if unspecified.
    };

    GetSecretValueResponse response = null;

    // For the sake of simplicity, this example handles only the most
    // general SecretsManager exception.
    try
    {
        response = await client.GetSecretValueAsync(request);
    }
    catch (AmazonSecretsManagerException e)
    {
        Console.WriteLine($"Error: {e.Message}");
    }

    return response;
}
```

```
/// <summary>
/// Decodes the secret returned by the call to GetSecretValueAsync and
/// returns it to the calling program.
/// </summary>
/// <param name="response">A GetSecretValueResponse object containing
/// the requested secret value returned by GetSecretValueAsync.</param>
/// <returns>A string representing the decoded secret value.</returns>
public static string DecodeString(GetSecretValueResponse response)
{
    // Decrypts secret using the associated AWS Key Management Service
    // Customer Master Key (CMK.) Depending on whether the secret is a
    // string or binary value, one of these fields will be populated.
    if (response.SecretString is not null)
    {
        var secret = response.SecretString;
        return secret;
    }
    else if (response.SecretBinary is not null)
    {
        var memoryStream = response.SecretBinary;
        StreamReader reader = new StreamReader(memoryStream);
        string decodedBinarySecret =
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(reader.ReadToEnd()));
        return decodedBinarySecret;
    }
    else
    {
        return string.Empty;
    }
}
}
```

- Weitere API-Informationen finden Sie unter [GetSecretValue](#) in der APIAWS SDK for .NET - Referenz für .

Amazon SES-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon SES verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Erstellen einer E-Mail-Vorlage

Das folgende Code-Beispiel zeigt, wie Sie eine Amazon-SES-E-Mail-Vorlage erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create an email template.
/// </summary>
/// <param name="name">Name of the template.</param>
/// <param name="subject">Email subject.</param>
/// <param name="text">Email body text.</param>
/// <param name="html">Email HTML body text.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateEmailTemplateAsync(string name, string subject,
string text,
    string html)
{
    var success = false;
```

```
try
{
    var response = await _amazonSimpleEmailService.CreateTemplateAsync(
        new CreateTemplateRequest
        {
            Template = new Template
            {
                TemplateName = name,
                SubjectPart = subject,
                TextPart = text,
                HtmlPart = html
            }
        });
    success = response.HttpStatusCode == HttpStatusCode.OK;
}
catch (Exception ex)
{
    Console.WriteLine("CreateEmailTemplateAsync failed with exception: " +
        ex.Message);
}

return success;
}
```

- Weitere API-Informationen finden Sie unter [CreateTemplate](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer E-Mail-Vorlage

Das folgende Code-Beispiel zeigt, wie Sie eine Amazon-SES-E-Mail-Vorlage löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an email template.
/// </summary>
/// <param name="templateName">Name of the template.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailTemplateAsync(string templateName)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.DeleteTemplateAsync(
            new DeleteTemplateRequest
            {
                TemplateName = templateName
            });
        success = response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Exception ex)
    {
        Console.WriteLine("DeleteEmailTemplateAsync failed with exception: " +
            ex.Message);
    }

    return success;
}
```

- Weitere API-Informationen finden Sie unter [DeleteTemplate](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Identität

Das folgende Code-Beispiel zeigt, wie Sie eine Amazon-SES-Identität löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an email identity.
/// </summary>
/// <param name="identityEmail">The identity email to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteIdentityAsync(string identityEmail)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.DeleteIdentityAsync(
            new DeleteIdentityRequest
            {
                Identity = identityEmail
            });
        success = response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Exception ex)
    {
        Console.WriteLine("DeleteIdentityAsync failed with exception: " +
            ex.Message);
    }

    return success;
}
```

- Weitere API-Informationen finden Sie unter [DeleteIdentity](#) in der APIAWS SDK for .NET - Referenz für .

Sendelimits abrufen

Das folgende Code-Beispiel zeigt, wie Sie Sendelimits für Amazon SES abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get information on the current account's send quota.
/// </summary>
/// <returns>The send quota response data.</returns>
public async Task<GetSendQuotaResponse> GetSendQuotaAsync()
{
    var result = new GetSendQuotaResponse();
    try
    {
        var response = await _amazonSimpleEmailService.GetSendQuotaAsync(
            new GetSendQuotaRequest());
        result = response;
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetSendQuotaAsync failed with exception: " +
ex.Message);
    }


    return result;
}
```

- Weitere API-Informationen finden Sie unter [GetSendQuota](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen des Status einer Identität

Das folgende Code-Beispiel zeigt, wie Sie den Status einer Amazon-SES-Identität abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get identity verification status for an email.
/// </summary>
/// <returns>The verification status of the email.</returns>
public async Task<VerificationStatus> GetIdentityStatusAsync(string email)
{
    var result = VerificationStatus.TemporaryFailure;
    try
    {
        var response =
            await
            _amazonSimpleEmailService.GetIdentityVerificationAttributesAsync(
                new GetIdentityVerificationAttributesRequest
                {
                    Identities = new List<string> { email }
                });

        if (response.VerificationAttributes.ContainsKey(email))
            result = response.VerificationAttributes[email].VerificationStatus;
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetIdentityStatusAsync failed with exception: " +
            ex.Message);
    }

    return result;
}
```

- Weitere API-Informationen finden Sie unter [GetIdentityVerificationAttributes](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von E-Mail-Vorlagen

Das folgende Code-Beispiel zeigt, wie Sie Amazon-SES-E-Mail-Vorlagen auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List email templates for the current account.
/// </summary>
/// <returns>A list of template metadata.</returns>
public async Task<List<TemplateMetadata>> ListEmailTemplatesAsync()
{
    var result = new List<TemplateMetadata>();
    try
    {
        var response = await _amazonSimpleEmailService.ListTemplatesAsync(
            new ListTemplatesRequest());
        result = response.TemplatesMetadata;
    }
    catch (Exception ex)
    {
        Console.WriteLine("ListEmailTemplatesAsync failed with exception: " +
            ex.Message);
    }

    return result;
}
```

- Weitere API-Informationen finden Sie unter [ListTemplates](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Identitäten

Das folgende Codebeispiel zeigt, wie Sie Amazon SES-Identitäten auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the identities of a specified type for the current account.
/// </summary>
/// <param name="identityType">IdentityType to list.</param>
/// <returns>The list of identities.</returns>
public async Task<List<string>> ListIdentitiesAsync(IdentityType identityType)
{
    var result = new List<string>();
    try
    {
        var response = await _amazonSimpleEmailService.ListIdentitiesAsync(
            new ListIdentitiesRequest
            {
                IdentityType = identityType
            });
        result = response.Identities;
    }
    catch (Exception ex)
    {
        Console.WriteLine("ListIdentitiesAsync failed with exception: " +
ex.Message);
    }

    return result;
}
```

- Weitere API-Informationen finden Sie unter [ListIdentities](#) in der APIAWS SDK for .NET - Referenz für .

E-Mail senden

Das folgende Codebeispiel zeigt, wie Sie E-Mails mit Amazon SES senden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Send an email by using Amazon SES.
/// </summary>
/// <param name="toAddresses">List of recipients.</param>
/// <param name="ccAddresses">List of cc recipients.</param>
/// <param name="bccAddresses">List of bcc recipients.</param>
/// <param name="bodyHtml">Body of the email in HTML.</param>
/// <param name="bodyText">Body of the email in plain text.</param>
/// <param name="subject">Subject line of the email.</param>
/// <param name="senderAddress">From address.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendEmailAsync(List<string> toAddresses,
    List<string> ccAddresses, List<string> bccAddresses,
    string bodyHtml, string bodyText, string subject, string senderAddress)
{
    var messageId = "";
    try
    {
        var response = await _amazonSimpleEmailService.SendEmailAsync(
            new SendEmailRequest
            {
                Destination = new Destination
                {
                    BccAddresses = bccAddresses,
                    CcAddresses = ccAddresses,
                    ToAddresses = toAddresses
                }
            }
        );
    }
}
```

```
        },
        Message = new Message
        {
            Body = new Body
            {
                Html = new Content
                {
                    Charset = "UTF-8",
                    Data = bodyHtml
                },
                Text = new Content
                {
                    Charset = "UTF-8",
                    Data = bodyText
                }
            },
            Subject = new Content
            {
                Charset = "UTF-8",
                Data = subject
            }
        },
        Source = senderAddress
    });
    messageId = response.MessageId;
}
catch (Exception ex)
{
    Console.WriteLine("SendEmailAsync failed with exception: " +
ex.Message);
}

return messageId;
}
```

- Weitere API-Informationen finden Sie unter [SendEmail](#) in der APIAWS SDK for .NET -Referenz für .

Senden einer Vorlage-E-Mail

Das folgende Code-Beispiel zeigt, wie man mit Amazon SES eine Vorlage-E-Mail sendet.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Send an email using a template.
/// </summary>
/// <param name="sender">Address of the sender.</param>
/// <param name="recipients">Addresses of the recipients.</param>
/// <param name="templateName">Name of the email template.</param>
/// <param name="templateDataObject">Data for the email template.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendTemplateEmailAsync(string sender, List<string>
recipients,
    string templateName, object templateDataObject)
{
    var messageId = "";
    try
    {
        // Template data should be serialized JSON from either a class or a
dynamic object.
        var templateData = JsonSerializer.Serialize(templateDataObject);

        var response = await _amazonSimpleEmailService.SendTemplatedEmailAsync(
            new SendTemplatedEmailRequest
            {
                Source = sender,
                Destination = new Destination
                {
                    ToAddresses = recipients
                },
                Template = templateName,
                TemplateData = templateData
            });
        messageId = response.MessageId;
    }
    catch (Exception ex)
```

```
    {  
        Console.WriteLine("SendTemplateEmailAsync failed with exception: " +  
ex.Message);  
    }  
  
    return messageId;  
}
```

- Weitere API-Informationen finden Sie unter [SendTemplatedEmail](#) in der APIAWS SDK for .NET -Referenz für .

Verifizieren einer E-Mail-Identität

Das folgende Code-Beispiel zeigt, wie Sie eine E-Mail-Identität mit Amazon SES verifizieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Starts verification of an email identity. This request sends an email  
/// from Amazon SES to the specified email address. To complete  
/// verification, follow the instructions in the email.  
/// </summary>  
/// <param name="recipientEmailAddress">Email address to verify.</param>  
/// <returns>True if successful.</returns>  
public async Task<bool> VerifyEmailIdentityAsync(string recipientEmailAddress)  
{  
    var success = false;  
    try  
    {  
        var response = await _amazonSimpleEmailService.VerifyEmailIdentityAsync(  
            new VerifyEmailIdentityRequest  
            {
```

```
        emailAddress = recipientEmailAddress
    });

    success = response.HttpStatusCode == HttpStatusCode.OK;
}
catch (Exception ex)
{
    Console.WriteLine("VerifyEmailIdentityAsync failed with exception: " +
ex.Message);
}

return success;
}
```

- Weitere API-Informationen finden Sie unter [VerifyEmailIdentity](#) in der APIAWS SDK for .NET - Referenz für .

Amazon SNS-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon SNS verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.


Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Amazon SNS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon SNS.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"  \tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListTopics](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

Überprüfen, ob eine Telefonnummer den Empfang abgemeldet ist

Das folgende Codebeispiel zeigt, wie Sie überprüfen können, ob eine Telefonnummer den Empfang von Amazon SNS-Nachrichten deaktiviert hat.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// This example shows how to use the Amazon Simple Notification Service
/// (Amazon SNS) to check whether a phone number has been opted out.
/// </summary>
public class IsPhoneNumOptedOut
{
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await CheckIfOptedOutAsync(client, phoneNumber);
    }
}
```

```

    /// <summary>
    /// Checks to see if the supplied phone number has been opted out.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS Client object used
    /// to check if the phone number has been opted out.</param>
    /// <param name="phoneNumber">A string representing the phone number
    /// to check.</param>
    public static async Task
    CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
    {
        var request = new CheckIfPhoneNumberIsOptedOutRequest
        {
            PhoneNumber = phoneNumber,
        };

        try
        {
            var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                string optOutStatus = response.IsOptedOut ? "opted out" : "not
opted out.";
                Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
            }
        }
        catch (AuthorizationErrorException ex)
        {
            Console.WriteLine($"{ex.Message}");
        }
    }
}

```

- Weitere API-Informationen finden Sie unter [CheckIfPhoneNumbersIsOptedOut](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Themas

Das folgende Codebeispiel zeigt, wie Sie ein Amazon SNS-Thema erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie ein Thema mit einem bestimmten Namen.

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

/// <summary>
/// This example shows how to use Amazon Simple Notification Service
/// (Amazon SNS) to add a new Amazon SNS topic.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}");
    }

    /// <summary>
    /// Creates a new SNS topic using the supplied topic name.
    /// </summary>
    /// <param name="client">The initialized SNS client object used to
    /// create the new topic.</param>
    /// <param name="topicName">A string representing the topic name.</param>
    /// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>
    public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
    {
        var request = new CreateTopicRequest
```

```
        {
            Name = topicName,
        };

        var response = await client.CreateTopicAsync(request);

        return response.TopicArn;
    }
}
```

Erstellen Sie ein neues Thema mit einem Namen und spezifischen FIFO- und Deduplizierungsattributen.

```
/// <summary>
/// Create a new topic with a name and specific FIFO and de-duplication
attributes.
/// </summary>
/// <param name="topicName">The name for the topic.</param>
/// <param name="useFifoTopic">True to use a FIFO topic.</param>
/// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
/// <returns>The ARN of the new topic.</returns>
public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
{
    var createTopicRequest = new CreateTopicRequest()
    {
        Name = topicName,
    };

    if (useFifoTopic)
    {
        // Update the name if it is not correct for a FIFO topic.
        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
```

```

        { "FifoTopic", "true" }
    };
    if (useContentBasedDeduplication)
    {
        createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
    }
}

var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
return createResponse.TopicArn;
}

```

- Weitere API-Informationen finden Sie unter [CreateTopic](#) in der APIAWS SDK for .NET - Referenz für .

Löschen von Abonnements

Das folgende Codebeispiel zeigt, wie Sie ein Amazon SNS-Abonnement löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Melden Sie sich mit einem Abonnement-ARN von einem Thema ab.

```

/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {

```

```
        SubscriptionArn = subscriptionArn
    });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Details zu API finden Sie unter [Abmelden](#) in der AWS SDK for .NET -API-Referenz.

Löschen eines Themas

Das folgende Codebeispiel zeigt, wie Sie ein Amazon SNS-Thema und alle Abonnements für dieses Thema löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie ein Thema mit seinem Themen-ARN.

```
/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteTopic](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen der Eigenschaften eines Themas

Das folgende Codebeispiel zeigt, wie Sie die Eigenschaften eines Amazon SNS-Themas abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;

/// <summary>
/// This example shows how to retrieve the attributes of an Amazon Simple
/// Notification Service (Amazon SNS) topic.
/// </summary>
public class GetTopicAttributes
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-west-2:000000000000:ExampleSNSTopic";
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var attributes = await GetTopicAttributesAsync(client, topicArn);
        DisplayTopicAttributes(attributes);
    }

    /// <summary>
    /// Given the ARN of the Amazon SNS topic, this method retrieves the topic
    /// attributes.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the attributes for the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic for which to retrieve
    /// the attributes.</param>
    /// <returns>A Dictionary of topic attributes.</returns>
```

```
public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
{
    var response = await client.GetTopicAttributesAsync(topicArn);

    return response.Attributes;
}

/// <summary>
/// This method displays the attributes for an Amazon SNS topic.
/// </summary>
/// <param name="topicAttributes">A Dictionary containing the
/// attributes for an Amazon SNS topic.</param>
public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
{
    foreach (KeyValuePair<string, string> entry in topicAttributes)
    {
        Console.WriteLine($"{entry.Key}: {entry.Value}\n");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [GetTopicAttributes](#) in der APIAWS SDK for .NET - Referenz für .

Listen der Abonnenten eines Themas

Das folgende Codebeispiel zeigt, wie Sie die Liste der Abonnenten eines Amazon SNS-Themas abrufen.

AWS SDK for .NET

Note

Auf [GitHub](#) gibt es mehr Beispiele. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example will retrieve a list of the existing Amazon Simple
/// Notification Service (Amazon SNS) subscriptions.
/// </summary>
public class ListSubscriptions
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        Console.WriteLine("Enter a topic ARN to list subscriptions for a
specific topic, " +
                        "or press Enter to list subscriptions for all
topics.");
        var topicArn = Console.ReadLine();
        Console.WriteLine();

        var subscriptions = await GetSubscriptionsListAsync(client, topicArn);

        DisplaySubscriptionList(subscriptions);
    }

    /// <summary>
    /// Gets a list of the existing Amazon SNS subscriptions, optionally by
    specifying a topic ARN.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to obtain the list of subscriptions.</param>
    /// <param name="topicArn">The optional ARN of a specific topic. Defaults to
    null.</param>
    /// <returns>A list containing information about each subscription.</
returns>
    public static async Task<List<Subscription>>
GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string topicArn
= null)
    {
```

```
var results = new List<Subscription>();

if (!string.IsNullOrEmpty(topicArn))
{
    var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
        new ListSubscriptionsByTopicRequest()
        {
            TopicArn = topicArn,
        });

    // Get the entire list using the paginator.
    await foreach (var subscription in paginateByTopic.Subscriptions)
    {
        results.Add(subscription);
    }
}
else
{
    var paginateAllSubscriptions =
client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());

    // Get the entire list using the paginator.
    await foreach (var subscription in
paginateAllSubscriptions.Subscriptions)
    {
        results.Add(subscription);
    }
}

return results;
}

/// <summary>
/// Display a list of Amazon SNS subscription information.
/// </summary>
/// <param name="subscriptionList">A list containing details for existing
/// Amazon SNS subscriptions.</param>
public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
{
    foreach (var subscription in subscriptionList)
    {
        Console.WriteLine($"Owner: {subscription.Owner}");
    }
}
```

```
        Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
        Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
        Console.WriteLine($"Endpoint: {subscription.Endpoint}");
        Console.WriteLine($"Protocol: {subscription.Protocol}");
        Console.WriteLine();
    }
}
}
```

- Weitere API-Informationen finden Sie unter [ListSubscriptions](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Themen

Das folgende Codebeispiel zeigt, wie Sie Amazon SNS-Themen auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// Lists the Amazon Simple Notification Service (Amazon SNS)
/// topics for the current account.
/// </summary>
public class ListSNSTopics
{
    public static async Task Main()
    {
```

```
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await GetTopicListAsync(client);
    }

    /// <summary>
    /// Retrieves the list of Amazon SNS topics in groups of up to 100
    /// topics.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the list of topics.</param>
    public static async Task GetTopicListAsync(IAmazonSimpleNotificationService
client)
    {
        // If there are more than 100 Amazon SNS topics, the call to
        // ListTopicsAsync will return a value to pass to the
        // method to retrieve the next 100 (or less) topics.
        string nextToken = string.Empty;

        do
        {
            var response = await client.ListTopicsAsync(nextToken);
            DisplayTopicsList(response.Topics);
            nextToken = response.NextToken;
        }
        while (!string.IsNullOrEmpty(nextToken));
    }

    /// <summary>
    /// Displays the list of Amazon SNS Topic ARNs.
    /// </summary>
    /// <param name="topicList">The list of Topic ARNs.</param>
    public static void DisplayTopicsList(List<Topic> topicList)
    {
        foreach (var topic in topicList)
        {
            Console.WriteLine($"{topic.TopicArn}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListTopics](#) in der APIAWS SDK for .NET -Referenz für .

Eine Nachricht mit Attributen veröffentlichen

Das folgende Codebeispiel zeigt, wie Sie eine Nachricht mit einem Attribut mithilfe von Amazon SNS veröffentlichen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Veröffentlichen Sie eine Nachricht zu einem Thema mit Gruppen-, Duplizierungs- und Attributoptionen.

```
/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
```

```
        "\r\nAll messages within the same group will be
received in the order " +
        "they were published.");

        Console.WriteLine();
        var messageId = GetUserResponse("Enter a message ID for
this message:", "1");

        if (!_useContentBasedDeduplication)
        {
            Console.WriteLine("Because you are not using content-based
deduplication, " +
                "you must enter a deduplication ID.");

            Console.WriteLine("Enter a deduplication ID for this message.");
            deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
        }

        if (GetYesNoResponse("Add an attribute to this message?"))
        {
            Console.WriteLine("Enter a number for an attribute.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", "1");
            int.TryParse(selection, out var selectionNumber);

            if (selectionNumber > 0 && selectionNumber < _tones.Length)
            {
                toneAttribute = _tones[selectionNumber - 1];
            }
        }

        var messageId = await SnsWrapper.PublishToTopicWithAttribute(
            _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

        Console.WriteLine($"Message published with id {messageID}.");
    }

    keepSendingMessages = GetYesNoResponse("Send another message?", false);
```

```

    }
}

```

Wenden Sie die Benutzerauswahl auf die Veröffentlichungsaktion an.

```

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
    and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String" } }
                }
        }
    }
}

```

```
        };  
    }  
  
    var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);  
    return publishResponse.MessageId;  
}
```

- Details zu API finden Sie unter [Veröffentlichen](#) in der AWS SDK for .NET -API-Referenz.

Veröffentlichen einer SMS-Nachricht

Das folgende Codebeispiel zeigt, wie SMS-Nachrichten mit Amazon SNS veröffentlicht werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace SNSMessageExample  
{  
    using System;  
    using System.Threading.Tasks;  
    using Amazon;  
    using Amazon.SimpleNotificationService;  
    using Amazon.SimpleNotificationService.Model;  
  
    public class SNSMessage  
    {  
        private AmazonSimpleNotificationServiceClient snsClient;  
  
        /// <summary>  
        /// Initializes a new instance of the <see cref="SNSMessage"/> class.  
        /// Constructs a new SNSMessage object initializing the Amazon Simple  
        /// Notification Service (Amazon SNS) client using the supplied  
        /// Region endpoint.  
        /// </summary>  
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
```



```
/// sending test messages with this object.</param>
public SNSMessage(RegionEndpoint regionEndpoint)
{
    snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);
}

/// <summary>
/// Sends the SMS message passed in the text parameter to the phone number
/// in phoneNum.
/// </summary>
/// <param name="phoneNum">The ten-digit phone number to which the text
/// message will be sent.</param>
/// <param name="text">The text of the message to send.</param>
/// <returns>Async task.</returns>
public async Task SendTextMessageAsync(string phoneNum, string text)
{
    if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
    {
        return;
    }

    // Now actually send the message.
    var request = new PublishRequest
    {
        Message = text,
        PhoneNumber = phoneNum,
    };

    try
    {
        var response = await snsClient.PublishAsync(request);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending message: {ex}");
    }
}
}
```

- Details zu API finden Sie unter [Veröffentlichen](#) in der AWS SDK for .NET -API-Referenz.

Veröffentlichung für ein Thema

Das folgende Codebeispiel zeigt, wie Nachrichten in einem Amazon SNS-Thema veröffentlicht werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Veröffentlichen einer Nachricht für ein Thema.

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

/// <summary>
/// This example publishes a message to an Amazon Simple Notification
/// Service (Amazon SNS) topic.
/// </summary>
public class PublishToSNSTopic
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-east-2:000000000000:ExampleSNSTopic";
        string messageText = "This is an example message to publish to the
ExampleSNSTopic.";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await PublishToTopicAsync(client, topicArn, messageText);
    }

    /// <summary>
    /// Publishes a message to an Amazon SNS topic.
    /// </summary>
    /// <param name="client">The initialized client object used to publish
    /// to the Amazon SNS topic.</param>
```

```

    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="messageText">The text of the message.</param>
    public static async Task PublishToTopicAsync(
        IAmazonSimpleNotificationService client,
        string topicArn,
        string messageText)
    {
        var request = new PublishRequest
        {
            TopicArn = topicArn,
            Message = messageText,
        };

        var response = await client.PublishAsync(request);

        Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
    }
}

```

Veröffentlichen Sie eine Nachricht zu einem Thema mit Attributs- und optionalen Duplizierungs- sowie Gruppen-IDs.

```

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
    and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,

```

```
string? deduplicationId = null,
string? groupId = null)
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String" } }
            };
    }


    var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}
```

- Details zu API finden Sie unter [Veröffentlichen](#) in der AWS SDK for .NET -API-Referenz.

Abonnieren einer SQS-Warteschlange für ein Thema.

Das folgende Code-Beispiel zeigt, wie man eine Amazon SQS-Warteschlange abonniert, damit sie Benachrichtigungen von einem Amazon SNS-Thema erhält.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Abonnieren Sie eine Warteschlange für ein Thema mit optionalen Filtern.

```
/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }

    var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}
```

- Details zu API finden Sie unter [Abonnieren](#) in der AWS SDK for .NET -API-Referenz.

Abonnieren einer E-Mail-Adresse für ein Thema

Das folgende Codebeispiel zeigt, wie Sie eine E-Mail-Adresse für ein Amazon SNS-Thema abonnieren.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Creates a new subscription to a topic.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
/// to create an Amazon SNS subscription.</param>
/// <param name="topicArn">The ARN of the topic to subscribe to.</param>
/// <returns>A SubscribeResponse object which includes the subscription
/// ARN for the new subscription.</returns>
public static async Task<SubscribeResponse> TopicSubscribeAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
{
    SubscribeRequest request = new SubscribeRequest()
    {
        TopicArn = topicArn,
        ReturnSubscriptionArn = true,
        Protocol = "email",
        Endpoint = "recipient@example.com",
    };

    var response = await client.SubscribeAsync(request);


    return response;
}
```

- Details zu API finden Sie unter [Abonnieren](#) in der AWS SDK for .NET -API-Referenz.

Abonnieren eines Themas mit einem Filter

Das folgende Codebeispiel zeigt, wie Sie ein Amazon SNS-Thema mit einem Filter abonnieren.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Abonnieren Sie eine Warteschlange für ein Thema mit optionalen Filtern.

```
/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }

    var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}
```

- Details zu API finden Sie unter [Abonnieren](#) in der AWS SDK for .NET -API-Referenz.

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
/// <summary>
/// Console application to run a workflow scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
```



```
// Set up dependency injection for Amazon EventBridge.
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<IAmazonSQS>()
            .AddAWSService<IAmazonSimpleNotificationService>()
            .AddTransient<SNSWrapper>()
            .AddTransient<SQSWrapper>()
    )
    .Build();

ServicesSetup(host);
PrintDescription();

await RunScenario();

}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
    SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
}

/// <summary>
/// Run the scenario for working with topics and queues.
/// </summary>
/// <returns>True if successful.</returns>
public static async Task<bool> RunScenario()
{
    try
    {
        await SetupTopic();

        await SetupQueues();
    }
}
```

```
        await PublishMessages();

        foreach (var queueUrl in _queueUrls)
        {
            var messages = await PollForMessages(queueUrl);
            if (messages.Any())
            {
                await DeleteMessages(queueUrl, messages);
            }
        }
        await CleanupResources();

        Console.WriteLine("Messaging with topics and queues workflow is
complete.");
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources();
        Console.WriteLine(new string('-', 80));
        return false;
    }
}

/// <summary>
/// Print a description for the tasks in the workflow.
/// </summary>
/// <returns>Async task.</returns>
private static void PrintDescription()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Welcome to messaging with topics and queues.");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"In this workflow, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
        $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
        $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");
}
```

```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
            $"\r\nFIFO topics deliver messages in order and support
deduplication and message filtering." +
            $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
"example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
is supported." +
                $"\r\nDeduplication IDs are either set in the message
or automatically generated " +
                $"\r\nfrom content using a hash function.\r\n" +
                $"\r\nIf a message is successfully published to an SNS
FIFO topic, any message " +
                $"\r\npublished and determined to have the same
deduplication ID, " +
                $"\r\nwithin the five-minute deduplication interval,
is accepted but not delivered.\r\n" +
                $"\r\nFor more information about deduplication, " +
                $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");
        }
    }

```

```
        _useContentBasedDeduplication = GetYesNoResponse("Use content-based
deduplication instead of entering a deduplication ID?");
        Console.WriteLine(new string('-', 80));
    }

    _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
_useContentBasedDeduplication);

    Console.WriteLine($"Your new topic with the name {_topicName}" +
        $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
        $"\r\nhas been created.\r\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)
    {
        var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
        if (_useFifoTopic)
        {
            // Only explain this once.
            if (i == 0)
            {
                Console.WriteLine(
                    "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
            }

            var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);
        }
    }
}
```

```
        _queueUrls[i] = queueUrl;

        Console.WriteLine($"Your new queue with the name {queueName}" +
            $"\r\nand queue URL {queueUrl}" +
            $"\r\nhas been created.\r\n");

        if (i == 0)
        {
            Console.WriteLine(
                $"The queue URL is used to retrieve the queue ARN,\r\n" +
                $"which is used to create a subscription.");
            Console.WriteLine(new string('-', 80));
        }

        var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

        if (i == 0)
        {
            Console.WriteLine(
                $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                $"messages from an SNS topic");
        }

        await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

        await SetupFilters(i, queueArn, queueName);
    }
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
```

```
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +
                "If you add a filter to this subscription, then only the
filtered messages " +
                "will be received in the queue.");

            Console.WriteLine(
                "For information about message filtering, " +
                "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

            Console.WriteLine(
                "For this example, you can filter messages by a " +
                "TONE attribute.");
        }

        var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

        string? filterPolicy = null;
        if (useFilter)
        {
            filterPolicy = CreateFilterPolicy();
        }
        var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
queueArn);
        _subscriptionArns[queueCount] = subscriptionArn;

        Console.WriteLine(
            $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
            $"with the subscription ARN {subscriptionArn}");
        Console.WriteLine(new string('-', 80));
    }
}
```

```
/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
        int.TryParse(selection, out selectionNumber);
        if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
        {
            filterSelections.Add(_tones[selectionNumber - 1]);
        }
    } while (selectionNumber != 0);

    var filters = new Dictionary<string, List<string>>
    {
        { "tone", filterSelections }
    };
    string filterPolicy = JsonSerializer.Serialize(filters);
    return filterPolicy;
}

/// <summary>
/// Publish messages using user settings.
/// </summary>
```

```
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                "\r\nAll messages within the same group will be
received in the order " +
                "they were published.");

            Console.WriteLine();
            var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
                    "you must enter a deduplication ID.");

                Console.WriteLine("Enter a deduplication ID for this message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
                Console.WriteLine("Enter a number for an attribute.");
                for (int i = 0; i < _tones.Length; i++)
                {
                    Console.WriteLine($"{i + 1}. {_tones[i]}");
                }
            }
        }
    }
}
```



```
        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageID = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

    keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");

    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();
    while (moreMessages)
    {
        var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

        moreMessages = newMessages.Any();
        if (moreMessages)
```

```
        {
            messages.AddRange(newMessages);
        }
    }

    Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

    foreach (var message in messages)
    {
        Console.WriteLine("\tMessage:" +
            $"{"\n\t{message.Body}");
    }

    Console.WriteLine(new string('-', 80));
    return messages;
}

/// <summary>
/// Delete the message using handles in a batch.
/// </summary>
/// <returns>Async task.</returns>
public static async Task DeleteMessages(string queueUrl, List<Message> messages)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
    await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CleanupResources()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    try
    {
        foreach (var queueUrl in _queueUrls)
        {
```

```

        if (!string.IsNullOrEmpty(queueUrl))
        {
            var deleteQueue =
                GetYesNoResponse($"Delete queue with url {queueUrl}?");
            if (deleteQueue)
            {
                await SqsWrapper.DeleteQueueByUrl(queueUrl);
            }
        }

        foreach (var subscriptionArn in _subscriptionArns)
        {
            if (!string.IsNullOrEmpty(subscriptionArn))
            {
                await SnsWrapper.UnsubscribeByArn(subscriptionArn);
            }
        }

        var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
        if (deleteTopic)
        {
            await SnsWrapper.DeleteTopicByArn(_topicArn);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
{
    if (UseConsole)
    {

```

```

        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}

/// <summary>
/// Helper method to get a string response from the user through the console.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static string GetUserResponse(string question, string defaultAnswer)
{
    if (UseConsole)
    {
        var response = "";
        while (string.IsNullOrEmpty(response))
        {
            Console.WriteLine(question);
            response = Console.ReadLine();
        }
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}
}

```

Erstellen Sie eine Klasse, die Amazon-SQS-Operationen einschließt.

```

/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{

```

```
private readonly IAmazonSQS _amazonSQSClient;

/// <summary>
/// Constructor for the Amazon SQS wrapper.
/// </summary>
/// <param name="amazonSQS">The injected Amazon SQS client.</param>
public SQSWrapper(IAmazonSQS amazonSQS)
{
    _amazonSQSClient = amazonSQS;
}

/// <summary>
/// Create a queue with a specific name.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <param name="useFifoQueue">True to use a FIFO queue.</param>
/// <returns>The url for the queue.</returns>
public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
{
    int maxMessage = 256 * 1024;
    var queueAttributes = new Dictionary<string, string>
    {
        {
            QueueAttributeName.MaximumMessageSize,
            maxMessage.ToString()
        }
    };

    var createQueueRequest = new CreateQueueRequest()
    {
        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
    }
}
```

```

        createQueueRequest.Attributes.Add(
            QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

    return getAttributesResponse.QueueARN;
}

/// <summary>
/// Set the policy attribute of a queue for a topic.
/// </summary>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="queueUrl">The url for the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
{
    var queuePolicy = "{" +
        "\nVersion\": \"2012-10-17\", " +
        "\nStatement\": [{" +

```

```
        "\"Effect\": \"Allow\"," +
        "\"Principal\": {" +
            $"\"Service\": " +
            "\"sns.amazonaws.com\"" +
            "}," +
        "\"Action\": \"sqs:SendMessage\"," +
        $"\"Resource\": \"{queueArn}\"," +
        "\"Condition\": {" +
            "\"ArnEquals\": {" +
                $"\"aws:SourceArn\": \"{topicArn}\""
            +
                "}" +
            "}" +
        "}]";
    var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
        new SetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
        });
    return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Receive messages from a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>The list of messages.</returns>
public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
{
    // Setting WaitTimeSeconds to non-zero enables long polling.
    // For information about long polling, see
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
    var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
        new ReceiveMessageRequest()
        {
            QueueUrl = queueUrl,
            MaxNumberOfMessages = maxMessages,
            WaitTimeSeconds = 1
        });
}
```

```
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```



```
}  
}
```

Erstellen Sie eine Klasse, die Amazon-SNS-Operationen einschließt.

```
/// <summary>  
/// Wrapper for Amazon Simple Notification Service (SNS) operations.  
/// </summary>  
public class SNSWrapper  
{  
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;  
  
    /// <summary>  
    /// Constructor for the Amazon SNS wrapper.  
    /// </summary>  
    /// <param name="amazonSNS">The injected Amazon SNS client.</param>  
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)  
    {  
        _amazonSNSClient = amazonSNS;  
    }  
  
    /// <summary>  
    /// Create a new topic with a name and specific FIFO and de-duplication  
attributes.  
    /// </summary>  
    /// <param name="topicName">The name for the topic.</param>  
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>  
    /// <param name="useContentBasedDeduplication">True to use content-based de-  
duplication.</param>  
    /// <returns>The ARN of the new topic.</returns>  
    public async Task<string> CreateTopicWithName(string topicName, bool  
useFifoTopic, bool useContentBasedDeduplication)  
    {  
        var createTopicRequest = new CreateTopicRequest()  
        {  
            Name = topicName,  
        };  
  
        if (useFifoTopic)  
        {  
            // Update the name if it is not correct for a FIFO topic.        }  
    }  
}
```

```
        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
            { "FifoTopic", "true" }
        };
        if (useContentBasedDeduplication)
        {
            createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
        }
    }

    var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
    return createResponse.TopicArn;
}

/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }
}
```

```
    }

    var subscribeResponse = await
    _amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}

/// <summary>
/// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="message">The message to publish.</param>
/// <param name="attributeName">The optional attribute for the message.</param>
/// <param name="attributeValue">The optional attribute value for the message.</
param>
/// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
/// <param name="groupId">The optional group ID for the message.</param>
/// <returns>The ID of the message published.</returns>
public async Task<string> PublishToTopicWithAttribute(
    string topicArn,
    string message,
    string? attributeName = null,
    string? attributeValue = null,
    string? deduplicationId = null,
    string? groupId = null)
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
            }
    }
}
```

```
        };
    }

    var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}

/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)

- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Veröffentlichen](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Abonnieren](#)
- [Unsubscribe](#)

Serverless-Beispiele

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von .NET.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
}
```

Amazon SQS-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon SQS verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon SQS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon SQS .

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });
    }
}
```

```
        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListQueues](#) in der APIAWS SDK for .NET -Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

Autorisieren eines Buckets zum Senden von Nachrichten an eine Warteschlange

Das folgende Codebeispiel zeigt, wie Sie einen Amazon S3-Bucket autorisieren, Nachrichten an eine Amazon SQSWarteschlange zu senden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;

public class AuthorizeS3ToSendMessage
```



```
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// AuthorizeS3ToSendMessageAsync method to authorize the named
    /// bucket to send messages in response to S3 events.
    /// </summary>
    public static async Task Main()
    {
        string queueUrl = "https://sqs.us-east-2.amazonaws.com/0123456789ab/
Example_Queue";
        string bucketName = "doc-example-bucket";

        // Create an Amazon SQS client object using the
        // default user. If the AWS Region you want to use
        // is different, supply the AWS Region as a parameter.
        IAmazonSQS client = new AmazonSQSClient();

        var queueARN = await client.AuthorizeS3ToSendMessageAsync(queueUrl,
bucketName);

        if (!string.IsNullOrEmpty(queueARN))
        {
            Console.WriteLine($"The Amazon S3 bucket: {bucketName} has been
successfully authorized.");
            Console.WriteLine($"{bucketName} can now send messages to the queue
with ARN: {queueARN}.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [SendMessage](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Warteschlange

Das folgende Codebeispiel zeigt, wie Sie eine Amazon SQS-Warteschlange erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Warteschlange mit einem bestimmten Namen.

```
/// <summary>
/// Create a queue with a specific name.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <param name="useFifoQueue">True to use a FIFO queue.</param>
/// <returns>The url for the queue.</returns>
public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
{
    int maxMessage = 256 * 1024;
    var queueAttributes = new Dictionary<string, string>
    {
        {
            QueueAttributeName.MaximumMessageSize,
            maxMessage.ToString()
        }
    };

    var createQueueRequest = new CreateQueueRequest()
    {
        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
    }
}
```

```

        createQueueRequest.Attributes.Add(
            QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

```

Erstellen Sie eine Amazon SQS-Warteschlange und senden Sie eine Nachricht an sie.

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
    // Specify your AWS Region (an example Region is shown).
    private static readonly string QueueName = "Example_Queue";
    private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
    private static IAmazonSQS client;

    public static async Task Main()
    {
        client = new AmazonSQSClient(ServiceRegion);
        var createQueueResponse = await CreateQueue(client, QueueName);

        string queueUrl = createQueueResponse.QueueUrl;

        Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
        {
            { "Title", new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },

```

```

        { "Author", new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
        { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
    };

    string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";

    var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
messageAttributes);
    }

    /// <summary>
    /// Creates a new Amazon SQS queue using the queue name passed to it
    /// in queueName.
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueName">A string representing the name of the queue
    /// to create.</param>
    /// <returns>A CreateQueueResponse that contains information about the
    /// newly created queue.</returns>
    public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
string queueName)
    {
        var request = new CreateQueueRequest
        {
            QueueName = queueName,
            Attributes = new Dictionary<string, string>
            {
                { "DelaySeconds", "60" },
                { "MessageRetentionPeriod", "86400" },
            },
        };

        var response = await client.CreateQueueAsync(request);
        Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

        return response;
    }

    /// <summary>
    /// Sends a message to an SQS queue.

```

```
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueUrl">The URL of the queue to which to send the
    /// message.</param>
    /// <param name="messageBody">A string representing the body of the
    /// message to be sent to the queue.</param>
    /// <param name="messageAttributes">Attributes for the message to be
    /// sent to the queue.</param>
    /// <returns>A SendMessageResponse object that contains information
    /// about the message that was sent.</returns>
    public static async Task<SendMessageResponse> SendMessage(
        IAmazonSQS client,
        string queueUrl,
        string messageBody,
        Dictionary<string, MessageAttributeValue> messageAttributes)
    {
        var sendMessageRequest = new SendMessageRequest
        {
            DelaySeconds = 10,
            MessageAttributes = messageAttributes,
            MessageBody = messageBody,
            QueueUrl = queueUrl,
        };

        var response = await client.SendMessageAsync(sendMessageRequest);
        Console.WriteLine($"Sent a message with id : {response.MessageId}");

        return response;
    }
}
```

- Weitere API-Informationen finden Sie unter [CreateQueue](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Nachrichtenstapels aus einer Warteschlange

Das folgende Codebeispiel zeigt, wie Sie einen Stapel von Nachrichten aus einer Amazon SQS-Warteschlange löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a batch of messages from a queue by its url.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
{
    var deleteRequest = new DeleteMessageBatchRequest()
    {
        QueueUrl = queueUrl,
        Entries = new List<DeleteMessageBatchRequestEntry>()
    };
    foreach (var message in messages)
    {
        deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
        {
            ReceiptHandle = message.ReceiptHandle,
            Id = message.MessageId
        });
    }

    var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

    return deleteResponse.Failed.Any();
}
```

- Weitere API-Informationen finden Sie unter [DeleteMessageBatch](#) in der APIAWS SDK for .NET -Referenz für .

Löschen einer Nachricht aus einer Warteschlange

Das folgende Codebeispiel zeigt, wie Sie eine Nachricht aus einer Amazon SQS-Warteschlange löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie eine Nachricht aus einer Amazon SQS-Warteschlange und löschen Sie dann die Nachricht.

```
public static async Task Main()
{
    // If the AWS Region you want to use is different from
    // the AWS Region defined for the default user, supply
    // the specify your AWS Region to the client constructor.
    var client = new AmazonSQSClient();
    string queueName = "Example_Queue";

    var queueUrl = await GetQueueUrl(client, queueName);
    Console.WriteLine($"The SQS queue's URL is {queueUrl}");

    var response = await ReceiveAndDeleteMessage(client, queueUrl);

    Console.WriteLine($"Message: {response.Messages[0]}");
}

/// <summary>
/// Retrieve the queue URL for the queue named in the queueName
/// property using the client object.
/// </summary>
/// <param name="client">The Amazon SQS client used to retrieve the
/// queue URL.</param>
/// <param name="queueName">A string representing name of the queue
/// for which to retrieve the URL.</param>
/// <returns>The URL of the queue.</returns>
```

```
public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
{
    var request = new GetQueueUrlRequest
    {
        QueueName = queueName,
    };

    GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
    return response.QueueUrl;
}

/// <summary>
/// Retrieves the message from the quque at the URL passed in the
/// queueURL parameters using the client.
/// </summary>
/// <param name="client">The SQS client used to retrieve a message.</param>
/// <param name="queueUrl">The URL of the queue from which to retrieve
/// a message.</param>
/// <returns>The response from the call to ReceiveMessageAsync.</returns>
public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
{
    // Receive a single message from the queue.
    var receiveMessageRequest = new ReceiveMessageRequest
    {
        AttributeNames = { "SentTimestamp" },
        MaxNumberOfMessages = 1,
        MessageAttributeNames = { "All" },
        QueueUrl = queueUrl,
        VisibilityTimeout = 0,
        WaitTimeSeconds = 0,
    };

    var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

    // Delete the received message from the queue.
    var deleteMessageRequest = new DeleteMessageRequest
    {
        QueueUrl = queueUrl,
        ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
    };
}
```



```
        await client.DeleteMessageAsync(deleteMessageRequest);

        return receiveMessageResponse;
    }
}
```

- Weitere API-Informationen finden Sie unter [DeleteMessage](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Warteschlange

Das folgende Codebeispiel zeigt, wie Sie eine Amazon SQS-Warteschlange löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie eine Warteschlange mithilfe ihrer URL.

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteQueue](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Attributen für eine Warteschlange

Das folgende Codebeispiel zeigt, wie Attribute für eine Amazon SQS-Warteschlange abgerufen werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

    return getAttributesResponse.QueueARN;
}
```

- Weitere API-Informationen finden Sie unter [GetQueueAttributes](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen der URL einer Warteschlange

Das folgende Codebeispiel zeigt, wie Sie die URL einer Amazon SQS-Warteschlange abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

public class GetQueueUrl
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
    /// queue.
    /// </summary>
    public static async Task Main()
    {
        // If the Amazon SQS message queue is not in the same AWS Region as your
        // default user, you need to provide the AWS Region as a parameter to
the
        // client constructor.
        var client = new AmazonSQSClient();

        string queueName = "New-Example-Queue";

        try
        {
            var response = await client.GetQueueUrlAsync(queueName);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"The URL for {queueName} is:
{response.QueueUrl}");
            }
        }
    }
}
```

```
    }
    catch (QueueDoesNotExistException ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine($"The queue {queueName} was not found.");
    }
}
}
```

- Weitere API-Informationen finden Sie unter [GetQueueUrl](#) in der APIAWS SDK for .NET - Referenz für .

Empfangen von Nachrichten aus einer Warteschlange

Das folgende Codebeispiel zeigt, wie Nachrichten aus einer Amazon SQS-Warteschlange empfangen werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie Nachrichten aus einer Warteschlange mithilfe der entsprechenden URL.

```
/// <summary>
/// Receive messages from a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>The list of messages.</returns>
public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
{
    // Setting WaitTimeSeconds to non-zero enables long polling.
    // For information about long polling, see
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
```

```

var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
    new ReceiveMessageRequest()
    {
        QueueUrl = queueUrl,
        MaxNumberOfMessages = maxMessages,
        WaitTimeSeconds = 1
    });
return messageResponse.Messages;
}

```

Empfangen Sie eine Nachricht aus einer Amazon SQS-Warteschlange und löschen Sie die Nachricht.

```

public static async Task Main()
{
    // If the AWS Region you want to use is different from
    // the AWS Region defined for the default user, supply
    // the specify your AWS Region to the client constructor.
    var client = new AmazonSQSClient();
    string queueName = "Example_Queue";

    var queueUrl = await GetQueueUrl(client, queueName);
    Console.WriteLine($"The SQS queue's URL is {queueUrl}");

    var response = await ReceiveAndDeleteMessage(client, queueUrl);

    Console.WriteLine($"Message: {response.Messages[0]}");
}

/// <summary>
/// Retrieve the queue URL for the queue named in the queueName
/// property using the client object.
/// </summary>
/// <param name="client">The Amazon SQS client used to retrieve the
/// queue URL.</param>
/// <param name="queueName">A string representing name of the queue
/// for which to retrieve the URL.</param>
/// <returns>The URL of the queue.</returns>
public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
{
    var request = new GetQueueUrlRequest

```

```
    {
        QueueName = queueName,
    };

    GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
    return response.QueueUrl;
}

/// <summary>
/// Retrieves the message from the quque at the URL passed in the
/// queueURL parameters using the client.
/// </summary>
/// <param name="client">The SQS client used to retrieve a message.</param>
/// <param name="queueUrl">The URL of the queue from which to retrieve
/// a message.</param>
/// <returns>The response from the call to ReceiveMessageAsync.</returns>
public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
{
    // Receive a single message from the queue.
    var receiveMessageRequest = new ReceiveMessageRequest
    {
        AttributeNames = { "SentTimestamp" },
        MaxNumberOfMessages = 1,
        MessageAttributeNames = { "All" },
        QueueUrl = queueUrl,
        VisibilityTimeout = 0,
        WaitTimeSeconds = 0,
    };

    var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

    // Delete the received message from the queue.
    var deleteMessageRequest = new DeleteMessageRequest
    {
        QueueUrl = queueUrl,
        ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
    };

    await client.DeleteMessageAsync(deleteMessageRequest);

    return receiveMessageResponse;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [ReceiveMessage](#) in der APIAWS SDK for .NET - Referenz für .

Senden einer Nachricht an eine Warteschlange

Das folgende Codebeispiel zeigt, wie eine Nachricht an eine Amazon SQS-Warteschlange gesendet wird.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Amazon SQS-Warteschlange und senden Sie eine Nachricht an sie.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
    // Specify your AWS Region (an example Region is shown).
    private static readonly string QueueName = "Example_Queue";
    private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
    private static IAmazonSQS client;

    public static async Task Main()
    {
        client = new AmazonSQSClient(ServiceRegion);
        var createQueueResponse = await CreateQueue(client, QueueName);
```

```
        string queueUrl = createQueueResponse.QueueUrl;

        Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
    {
        { "Title",    new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
        { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
        { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
    };

        string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";

        var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
messageAttributes);
    }

    /// <summary>
    /// Creates a new Amazon SQS queue using the queue name passed to it
    /// in queueName.
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueName">A string representing the name of the queue
    /// to create.</param>
    /// <returns>A CreateQueueResponse that contains information about the
    /// newly created queue.</returns>
    public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
string queueName)
    {
        var request = new CreateQueueRequest
        {
            QueueName = queueName,
            Attributes = new Dictionary<string, string>
            {
                { "DelaySeconds", "60" },
                { "MessageRetentionPeriod", "86400" },
            },
        };

        var response = await client.CreateQueueAsync(request);
    }
}
```



```
        Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

        return response;
    }

    /// <summary>
    /// Sends a message to an SQS queue.
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueUrl">The URL of the queue to which to send the
    /// message.</param>
    /// <param name="messageBody">A string representing the body of the
    /// message to be sent to the queue.</param>
    /// <param name="messageAttributes">Attributes for the message to be
    /// sent to the queue.</param>
    /// <returns>A SendMessageResponse object that contains information
    /// about the message that was sent.</returns>
    public static async Task<SendMessageResponse> SendMessage(
        IAmazonSQS client,
        string queueUrl,
        string messageBody,
        Dictionary<string, MessageAttributeValue> messageAttributes)
    {
        var sendMessageRequest = new SendMessageRequest
        {
            DelaySeconds = 10,
            MessageAttributes = messageAttributes,
            MessageBody = messageBody,
            QueueUrl = queueUrl,
        };

        var response = await client.SendMessageAsync(sendMessageRequest);
        Console.WriteLine($"Sent a message with id : {response.MessageId}");

        return response;
    }
}
```

- Weitere API-Informationen finden Sie unter [SendMessage](#) in der APIAWS SDK for .NET - Referenz für .

Festlegen von Warteschlangenattributen

Das folgende Codebeispiel zeigt, wie Sie die Attribute für eine Amazon SQS-Warteschlange festlegen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Legen Sie das Richtlinienattribut einer Warteschlange für ein Thema fest.

```

/// <summary>
/// Set the policy attribute of a queue for a topic.
/// </summary>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="queueUrl">The url for the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
{
    var queuePolicy = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": { " +
                "\"Service\": " +
                    "\"sns.amazonaws.com\"" +
                "}, " +
            "\"Action\": \"sqs:SendMessage\", " +
            "\"Resource\": \"{queueArn}\", " +
            "\"Condition\": { " +
                "\"ArnEquals\": { " +
                    "\"aws:SourceArn\": \"{topicArn}\""
+
                    "}" +
                "}" +
            "}] " +
        "}";

```

```
var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(  
    new SetQueueAttributesRequest()  
    {  
        QueueUrl = queueUrl,  
        Attributes = new Dictionary<string, string>() { { "Policy",  
queuePolicy } }  
    });  
return attributesResponse.HttpStatusCode == HttpStatusCode.OK;  
}
```

- Weitere API-Informationen finden Sie unter [SetQueueAttributes](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
/// <summary>  
/// Console application to run a workflow scenario for topics and queues.  
/// </summary>  
public static class TopicsAndQueues  
{
```

```

private static bool _useFifoTopic = false;
private static bool _useContentBasedDeduplication = false;
private static string _topicName = null!;
private static string _topicArn = null!;

private static readonly int _queueCount = 2;
private static readonly string[] _queueUrls = new string[_queueCount];
private static readonly string[] _subscriptionArns = new string[_queueCount];
private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
public static SNSWrapper SnsWrapper { get; set; } = null!;
public static SQSWrapper SqsWrapper { get; set; } = null!;
public static bool UseConsole { get; set; } = true;
static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EventBridge.
    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<IAmazonSQS>()
                .AddAWSService<IAmazonSimpleNotificationService>()
                .AddTransient<SNSWrapper>()
                .AddTransient<SQSWrapper>()
            )
        .Build();

    ServicesSetup(host);
    PrintDescription();

    await RunScenario();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
}

```

```
SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
}

/// <summary>
/// Run the scenario for working with topics and queues.
/// </summary>
/// <returns>True if successful.</returns>
public static async Task<bool> RunScenario()
{
    try
    {
        await SetupTopic();

        await SetupQueues();

        await PublishMessages();

        foreach (var queueUrl in _queueUrls)
        {
            var messages = await PollForMessages(queueUrl);
            if (messages.Any())
            {
                await DeleteMessages(queueUrl, messages);
            }
        }
        await CleanupResources();

        Console.WriteLine("Messaging with topics and queues workflow is
complete.");
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources();
        Console.WriteLine(new string('-', 80));
        return false;
    }
}

/// <summary>
/// Print a description for the tasks in the workflow.
```

```
/// </summary>
/// <returns>Async task.</returns>
private static void PrintDescription()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Welcome to messaging with topics and queues.");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"In this workflow, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
        $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
        $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up the SNS topic to be used with the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string> SetupTopic()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
        $"\r\nFIFO topics deliver messages in order and support
deduplication and message filtering." +
        $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

    _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
topics?");

    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        _topicName = GetUserResponse("Enter a name for your SNS topic: ",
"example-topic");
        Console.WriteLine(
            "Because you have selected a FIFO topic, '.fifo' must be appended to
the topic name.\r\n");
    }
}
```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
is supported." +
            $"\r\nDeduplication IDs are either set in the message
or automatically generated " +
            $"\r\nfrom content using a hash function.\r\n" +
            $"\r\nIf a message is successfully published to an SNS
FIFO topic, any message " +
            $"\r\npublished and determined to have the same
deduplication ID, " +
            $"\r\nwithin the five-minute deduplication interval,
is accepted but not delivered.\r\n" +
            $"\r\nFor more information about deduplication, " +
            $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

        _useContentBasedDeduplication = GetYesNoResponse("Use content-based
deduplication instead of entering a deduplication ID?");
        Console.WriteLine(new string('-', 80));
    }

    _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
_useContentBasedDeduplication);

    Console.WriteLine($"Your new topic with the name {_topicName}" +
        $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
        $"\r\nhas been created.\r\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)

```

```
    {
        var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
        if (_useFifoTopic)
        {
            // Only explain this once.
            if (i == 0)
            {
                Console.WriteLine(
                    "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
            }

            var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

            _queueUrls[i] = queueUrl;

            Console.WriteLine($"Your new queue with the name {queueName}" +
                $"\r\nand queue URL {queueUrl}" +
                $"\r\nhas been created.\r\n");

            if (i == 0)
            {
                Console.WriteLine(
                    $"The queue URL is used to retrieve the queue ARN,\r\n" +
                    $"which is used to create a subscription.");
                Console.WriteLine(new string('-', 80));
            }

            var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

            if (i == 0)
            {
                Console.WriteLine(
                    $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                    $"messages from an SNS topic");
            }

            await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

            await SetupFilters(i, queueArn, queueName);
        }
    }
}
```



```
    }
  }

  Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
  if (_useFifoTopic)
  {
    Console.WriteLine(new string('-', 80));
    // Only explain this once.
    if (queueCount == 0)
    {
      Console.WriteLine(
        "Subscriptions to a FIFO topic can have filters." +
        "If you add a filter to this subscription, then only the
filtered messages " +
        "will be received in the queue.");

      Console.WriteLine(
        "For information about message filtering, " +
        "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

      Console.WriteLine(
        "For this example, you can filter messages by a " +
        "TONE attribute.");
    }

    var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

    string? filterPolicy = null;
    if (useFilter)
    {
```

```

        filterPolicy = CreateFilterPolicy();
    }
    var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
    queueArn);
    _subscriptionArns[queueCount] = subscriptionArn;

    Console.WriteLine(
        $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
        $"with the subscription ARN {subscriptionArn}");
    Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"  \t{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
        int.TryParse(selection, out selectionNumber);
        if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
        {

```

```
        filterSelections.Add(_tones[selectionNumber - 1]);
    }
} while (selectionNumber != 0);

var filters = new Dictionary<string, List<string>>
{
    { "tone", filterSelections }
};
string filterPolicy = JsonSerializer.Serialize(filters);
return filterPolicy;
}

/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                "\r\nAll messages within the same group will be
received in the order " +
                "they were published.");

            Console.WriteLine();
            var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

            if (!_useContentBasedDeduplication)
            {
```

```
        Console.WriteLine("Because you are not using content-based
deduplication, " +
                           "you must enter a deduplication ID.");

        Console.WriteLine("Enter a deduplication ID for this message.");
        deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
    }

    if (GetYesNoResponse("Add an attribute to this message?"))
    {
        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageId = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
}
```

```
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();
    while (moreMessages)
    {
        var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

        moreMessages = newMessages.Any();
        if (moreMessages)
        {
            messages.AddRange(newMessages);
        }
    }

    Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

    foreach (var message in messages)
    {
        Console.WriteLine("\tMessage:" +
            $"{"\n\t{message.Body}");
    }

    Console.WriteLine(new string('-', 80));
    return messages;
}

/// <summary>
/// Delete the message using handles in a batch.
/// </summary>
/// <returns>Async task.</returns>
public static async Task DeleteMessages(string queueUrl, List<Message> messages)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
    await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
}
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
```

```
        Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
{
    if (UseConsole)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}

/// <summary>
/// Helper method to get a string response from the user through the console.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static string GetUserResponse(string question, string defaultAnswer)
{
    if (UseConsole)
    {
        var response = "";
        while (string.IsNullOrEmpty(response))
        {
            Console.WriteLine(question);
            response = Console.ReadLine();
        }
    }
}
```

```
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}
}
```

Erstellen Sie eine Klasse, die Amazon-SQS-Operationen einschließt.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        }
    }
}
```



```
};

var createQueueRequest = new CreateQueueRequest()
{
    QueueName = queueName,
    Attributes = queueAttributes
};

if (useFifoQueue)
{
    // Update the name if it is not correct for a FIFO queue.
    if (!queueName.EndsWith(".fifo"))
    {
        createQueueRequest.QueueName = queueName + ".fifo";
    }

    // Add an attribute for a FIFO queue.
    createQueueRequest.Attributes.Add(
        QueueAttributeName.FifoQueue, "true");
}

var createResponse = await _amazonSQSClient.CreateQueueAsync(
    new CreateQueueRequest()
    {
        QueueName = queueName
    });
return createResponse.QueueUrl;
}

/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);
}
```

```

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
            "\"Version\": \"2012-10-17\"," +
            "\"Statement\": [{" +
                "\"Effect\": \"Allow\"," +
                "\"Principal\": {" +
                    "\"Service\": " +
                        "\"sns.amazonaws.com\"" +
                    "}," +
                "\"Action\": \"sqs:SendMessage\"," +
                "\"Resource\": \"{queueArn}\"," +
                "\"Condition\": {" +
                    "\"ArnEquals\": {" +
                        "\"aws:SourceArn\": \"{topicArn}\""
+
                    "}" +
                "}" +
            "}]}" +
            "}";

        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });

        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.

```

```
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>The list of messages.</returns>
public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
{
    // Setting WaitTimeSeconds to non-zero enables long polling.
    // For information about long polling, see
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
    var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
        new ReceiveMessageRequest()
        {
            QueueUrl = queueUrl,
            MaxNumberOfMessages = maxMessages,
            WaitTimeSeconds = 1
        });
    return messageResponse.Messages;
}

/// <summary>
/// Delete a batch of messages from a queue by its url.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
{
    var deleteRequest = new DeleteMessageBatchRequest()
    {
        QueueUrl = queueUrl,
        Entries = new List<DeleteMessageBatchRequestEntry>()
    };
    foreach (var message in messages)
    {
        deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
        {
            ReceiptHandle = message.ReceiptHandle,
            Id = message.MessageId
        });
    }

    var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);
}
```

```

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}

```

Erstellen Sie eine Klasse, die Amazon-SNS-Operationen einschließt.

```

/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
    attributes.

```

```
/// </summary>
/// <param name="topicName">The name for the topic.</param>
/// <param name="useFifoTopic">True to use a FIFO topic.</param>
/// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
/// <returns>The ARN of the new topic.</returns>
public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
{
    var createTopicRequest = new CreateTopicRequest()
    {
        Name = topicName,
    };

    if (useFifoTopic)
    {
        // Update the name if it is not correct for a FIFO topic.
        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
            { "FifoTopic", "true" }
        };
        if (useContentBasedDeduplication)
        {
            createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
        }
    }

    var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
    return createResponse.TopicArn;
}

/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
```

```
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
```

```
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }

    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
```

```
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von .NET.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {

```

```

        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
}

```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von .NET.

```

using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {

```

```
List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
List<SQSBatchResponse.BatchItemFailure>();
foreach(var message in evnt.Records)
{
    try
    {
        //process your message
        await ProcessMessageAsync(message, context);
    }
    catch (System.Exception)
    {
        //Add failed message identifier to the batchItemFailures list
        batchItemFailures.Add(new
SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
    }
}
return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

Step-Functions-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Step Functions verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Step Functions

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Step Functions.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

public class HelloStepFunctions
{
    static async Task Main()
    {
        var stepFunctionsClient = new AmazonStepFunctionsClient();

        Console.Clear();
        Console.WriteLine("Welcome to AWS Step Functions");
        Console.WriteLine("Let's list up to 10 of your state machines:");
        var stateMachineListRequest = new ListStateMachinesRequest { MaxResults =
10 };

        // Get information for up to 10 Step Functions state machines.
        var response = await
stepFunctionsClient.ListStateMachinesAsync(stateMachineListRequest);
```

```
        if (response.StateMachines.Count > 0)
        {
            response.StateMachines.ForEach(stateMachine =>
            {
                Console.WriteLine($"State Machine Name: {stateMachine.Name}\tAmazon
Resource Name (ARN): {stateMachine.StateMachineArn}");
            });
        }
        else
        {
            Console.WriteLine("\tNo state machines were found.");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [ListStateMachines](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Erstellen eines Zustandsautomaten

Das folgende Codebeispiel zeigt, wie Sie einen Step-Functions-Zustandsautomaten erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Create a Step Functions state machine.
/// </summary>
/// <param name="stateMachineName">Name for the new Step Functions state
/// machine.</param>
/// <param name="definition">A JSON string that defines the Step Functions
/// state machine.</param>
/// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
/// <returns></returns>
public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
{
    var request = new CreateStateMachineRequest
    {
        Name = stateMachineName,
        Definition = definition,
        RoleArn = roleArn
    };

    var response =
        await _amazonStepFunctions.CreateStateMachineAsync(request);
    return response.StateMachineArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateStateMachine](#) in der APIAWS SDK for .NET - Referenz für .

Erstellen einer Aktivität

Das folgende Codebeispiel zeigt, wie Sie eine Step-Functions-Aktivität erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
```

```
/// Create a Step Functions activity using the supplied name.
/// </summary>
/// <param name="activityName">The name for the new Step Functions activity.</
param>
/// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
public async Task<string> CreateActivity(string activityName)
{
    var response = await _amazonStepFunctions.CreateActivityAsync(new
CreateActivityRequest { Name = activityName });
    return response.ActivityArn;
}
```

- Weitere API-Informationen finden Sie unter [CreateActivity](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines Zustandsautomaten

Das folgende Codebeispiel zeigt, wie Sie einen Step-Functions-Zustandsautomaten löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a Step Functions state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// state machine.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteStateMachine(string stateMachineArn)
{
    var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
    { StateMachineArn = stateMachineArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [DeleteStateMachine](#) in der APIAWS SDK for .NET - Referenz für .

Löschen einer Aktivität

Das folgende Codebeispiel zeigt, wie Sie eine Step-Functions-Aktivität löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Delete a Step Machine activity.
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the activity.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteActivity(string activityArn)
{
    var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteActivity](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben eines Zustandsautomaten

Das folgende Codebeispiel zeigt, wie ein Step-Functions-Zustandsautomat beschrieben wird.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Retrieve information about the specified Step Functions state machine.
/// </summary>
/// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine to retrieve.</param>
/// <returns>Information about the specified Step Functions state machine.</
returns>
public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
StateMachineArn)
{
    var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
    return response;
}
```

- Weitere API-Informationen finden Sie unter [DescribeStateMachine](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben einer Ausführung eines Zustandsautomaten

Das folgende Codebeispiel zeigt, wie Sie eine Ausführung eines Step Functions-Zustandsautomaten beschreiben.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve information about the specified Step Functions execution.
/// </summary>
/// <param name="executionArn">The Amazon Resource Name (ARN) of the
/// Step Functions execution.</param>
/// <returns>The API response returned by the API.</returns>
public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
{
    var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
    return response;
}
```

- Weitere API-Informationen finden Sie unter [DescribeExecution](#) in der APIAWS SDK for .NET - Referenz für .

Abrufen von Aufgabendaten für eine Aktivität

Das folgende Codebeispiel zeigt, wie Sie Aufgabendaten für eine Step-Functions-Aktivität abrufen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve a task with the specified Step Functions activity
/// with the specified Amazon Resource Name (ARN).
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the Step Functions activity.</param>
/// <param name="workerName">The name of the Step Functions worker.</param>
/// <returns>The response from the Step Functions activity.</returns>
public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
```

```
{
    var response = await _amazonStepFunctions.GetActivityTaskAsync(new
    GetActivityTaskRequest
    { ActivityArn = activityArn, WorkerName = workerName });
    return response;
}
```

- Weitere API-Informationen finden Sie unter [GetActivityTask](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Aktivitäten

Das folgende Codebeispiel zeigt, wie Step-Functions-Aktivitäten aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List the Step Functions activities for the current account.
/// </summary>
/// <returns>A list of ActivityListItems.</returns>
public async Task<List<ActivityListItem>> ListActivitiesAsync()
{
    var request = new ListActivitiesRequest();
    var activities = new List<ActivityListItem>();

    do
    {
        var response = await _amazonStepFunctions.ListActivitiesAsync(request);

        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }
    }
}
```

```
        activities.AddRange(response.Activities);
    }
    while (request.NextToken is not null);

    return activities;
}
```

- Weitere API-Informationen finden Sie unter [ListActivities](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Zustandsautomatenausführungen

Das folgende Codebeispiel zeigt, wie Sie Step-Functions-Zustandsautomatenausführungen auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve information about executions of a Step Functions
/// state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>A list of ExecutionListItem objects.</returns>
public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
{
    var executions = new List<ExecutionListItem>();
    ListExecutionsResponse response;
    var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

    do
    {
```

```
        response = await _amazonStepFunctions.ListExecutionsAsync(request);
        executions.AddRange(response.Executions);
        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }
    } while (response.NextToken is not null);

    return executions;
}
```

- Weitere API-Informationen finden Sie unter [ListExecutions](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten von Zustandsautomaten

Das folgende Codebeispiel zeigt, wie Step-Functions-Zustandsautomaten aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Retrieve a list of Step Functions state machines.
/// </summary>
/// <returns>A list of StateMachineListItem objects.</returns>
public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
{
    var stateMachines = new List<StateMachineListItem>();
    var listStateMachinesPaginator =
        _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

    await foreach (var response in listStateMachinesPaginator.Responses)
    {
        stateMachines.AddRange(response.StateMachines);
    }
}
```

```

    }

    return stateMachines;
}

```

- Weitere API-Informationen finden Sie unter [ListStateMachines](#) in der APIAWS SDK for .NET - Referenz für .

Senden einer Erfolgsantwort an eine Aufgabe

Das folgende Codebeispiel zeigt, wie Sie eine Erfolgsantwort an eine Step-Functions-Aufgabe senden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// <summary>
/// Indicate that the Step Functions task, indicated by the
/// task token, has completed successfully.
/// </summary>
/// <param name="taskToken">Identifies the task.</param>
/// <param name="taskResponse">The response received from executing the task.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
{
    var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
    { TaskToken = taskToken, Output = taskResponse });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Weitere API-Informationen finden Sie unter [SendTaskSuccess](#) in der APIAWS SDK for .NET - Referenz für .

Starten einer Zustandsautomatenausführung

Das folgende Codebeispiel zeigt, wie Sie eine Ausführung eines Step-Functions-Zustandsautomaten starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Start execution of an AWS Step Functions state machine.
/// </summary>
/// <param name="executionName">The name to use for the execution.</param>
/// <param name="executionJson">The JSON string to pass for execution.</param>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
/// execution.</returns>
public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
{
    var executionRequest = new StartExecutionRequest
    {
        Input = executionJson,
        StateMachineArn = stateMachineArn
    };

    var response = await
_amazonStepFunctions.StartExecutionAsync(executionRequest);
    return response.ExecutionArn;
}
```

- Weitere API-Informationen finden Sie unter [StartExecution](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit Zustandsautomaten

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine -Aktivität.
- Erstellen Sie einen Zustandsautomaten aus einer Amazon States Language-Definition, die die zuvor erstellte Aktivität als Schritt enthält.
- Führen Sie den Zustandsautomaten aus und reagieren Sie auf die Aktivität mit Benutzereingabe.
- Rufen Sie nach Abschluss der Ausführung den endgültigen Status und die Ausgabe ab und bereinigen Sie dann die Ressourcen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
global using System.Text.Json;
global using Amazon.StepFunctions;
global using Microsoft.Extensions.Configuration;
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;
global using StepFunctionsActions;
global using LogLevel = Microsoft.Extensions.Logging.LogLevel;
```



```
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.StepFunctions.Model;

namespace StepFunctionsBasics;

public class StepFunctionsBasics
{
    private static ILogger _logger = null!;
    private static IConfigurationRoot _configuration = null!;
    private static IAmazonIdentityManagementService _iamService = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Step Functions.
        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<IAmazonStepFunctions>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<StepFunctionsWrapper>()
                )
            .Build();

        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<StepFunctionsBasics>();

        // Load configuration settings.
        _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 activityName = _configuration["ActivityName"];
        var stateMachineName = _configuration["StateMachineName"];
    }
}
```

```
var roleName = _configuration["RoleName"];
var repoBaseDir = _configuration["RepoBaseDir"];
var jsonFilePath = _configuration["JsonFilePath"];
var jsonFileName = _configuration["JsonFileName"];

var uiMethods = new UiMethods();
var stepFunctionsWrapper =
host.Services.GetRequiredService<StepFunctionsWrapper>();

    _iamService =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();

    // Load definition for the state machine from a JSON file.
    var stateDefinitionJson = File.ReadAllText($"{repoBaseDir}{jsonFilePath}
{jsonFileName}");

    Console.Clear();
    uiMethods.DisplayOverview();
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("Create activity");
    Console.WriteLine("Let's start by creating an activity.");
    string activityArn;
    string stateMachineArn;

    // Check to see if the activity already exists.
    var activityList = await stepFunctionsWrapper.ListActivitiesAsync();
    var existingActivity = activityList.FirstOrDefault(activity => activity.Name
== activityName);
    if (existingActivity is not null)
    {
        activityArn = existingActivity.ActivityArn;
        Console.WriteLine($"Activity, {activityName}, already exists.");
    }
    else
    {
        activityArn = await stepFunctionsWrapper.CreateActivity(activityName);
    }

    // Swap the placeholder in the JSON file with the Amazon Resource Name (ARN)
    // of the recently created activity.
    var stateDefinition =
stateDefinitionJson.Replace("#{DOC_EXAMPLE_ACTIVITY_ARN}", activityArn);
```

```
uiMethods.DisplayTitle("Create state machine");
Console.WriteLine("Now we'll create a state machine.");

// Find or create an IAM role that can be assumed by Step Functions.
var role = await GetOrCreateStateMachineRole(roleName);

// See if the state machine already exists.
var stateMachineList = await stepFunctionsWrapper.ListStateMachinesAsync();
var existingStateMachine =
    stateMachineList.FirstOrDefault(stateMachine => stateMachine.Name ==
stateMachineName);
if (existingStateMachine is not null)
{
    Console.WriteLine($"State machine, {stateMachineName}, already
exists.");
    stateMachineArn = existingStateMachine.StateMachineArn;
}
else
{
    // Create the state machine.
    stateMachineArn =
        await stepFunctionsWrapper.CreateStateMachine(stateMachineName,
stateDefinition, role.Arn);
    uiMethods.PressEnter();
}

Console.WriteLine("The state machine has been created.");
var describeStateMachineResponse = await
stepFunctionsWrapper.DescribeStateMachineAsync(stateMachineArn);

Console.WriteLine($"{describeStateMachineResponse.Name}\t{describeStateMachineResponse.State
Console.WriteLine($"Current status: {describeStateMachineResponse.Status}");
Console.WriteLine($"Amazon Resource Name (ARN) of the role assumed by the
state machine: {describeStateMachineResponse.RoleArn}");

var userName = string.Empty;
Console.Write("Before we start the state machine, tell me what should
ChatSFN call you? ");
userName = Console.ReadLine();

// Keep asking until the user enters a string value.
while (string.IsNullOrEmpty(userName))
{
```

```
        Console.WriteLine("Enter your name: ");
        userName = Console.ReadLine();
    }

    var executionJson = @"{"name": "" + userName + @""}";

    // Start the state machine execution.
    Console.WriteLine("Now we'll start execution of the state machine.");
    var executionArn = await
stepFunctionsWrapper.StartExecutionAsync(executionJson, stateMachineArn);
    Console.WriteLine("State machine started.");

    Console.WriteLine($"Thank you, {userName}. Now let's get started...");
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("ChatSFN");

    var isDone = false;
    var response = new GetActivityTaskResponse();
    var taskToken = string.Empty;
    var userChoice = string.Empty;

    while (!isDone)
    {
        response = await stepFunctionsWrapper.GetActivityTaskAsync(activityArn,
"MvpWorker");
        taskToken = response.TaskToken;

        // Parse the returned JSON string.
        var taskJsonResponse = JsonDocument.Parse(response.Input);
        var taskJsonObject = taskJsonResponse.RootElement;
        var message = taskJsonObject.GetProperty("message").GetString();
        var actions =
taskJsonObject.GetProperty("actions").EnumerateArray().Select(x =>
x.ToString()).ToList();
        Console.WriteLine($"\\n{message}\\n");

        // Prompt the user for another choice.
        Console.WriteLine("ChatSFN: What would you like me to do?");
        actions.ForEach(action => Console.WriteLine($"\\t{action}"));
        Console.Write($"\\n{userName}, tell me your choice: ");
        userChoice = Console.ReadLine();
        if (userChoice?.ToLower() == "done")
        {
```

```
        isDone = true;
    }

    Console.WriteLine($"You have selected: {userChoice}");
    var jsonResponse = @"{""action"": "" + userChoice + @""}";

    await stepFunctionsWrapper.SendTaskSuccessAsync(taskToken,
jsonResponse);
    }

    await stepFunctionsWrapper.StopExecution(executionArn);
    Console.WriteLine("Now we will wait for the execution to stop.");
    DescribeExecutionResponse executionResponse;
    do
    {
        executionResponse = await
stepFunctionsWrapper.DescribeExecutionAsync(executionArn);
    } while (executionResponse.Status == ExecutionStatus.RUNNING);

    Console.WriteLine("State machine stopped.");
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("State machine executions");
    Console.WriteLine("Now let's take a look at the execution values for the
state machine.");

    // List the executions.
    var executions = await
stepFunctionsWrapper.ListExecutionsAsync(stateMachineArn);

    uiMethods.DisplayTitle("Step function execution values");
    executions.ForEach(execution =>
    {
        Console.WriteLine($"{execution.Name}\t{execution.StartDate} to
{execution.StopDate}");
    });

    uiMethods.PressEnter();

    // Now delete the state machine and the activity.
    uiMethods.DisplayTitle("Clean up resources");
    Console.WriteLine("Deleting the state machine...");

    await stepFunctionsWrapper.DeleteStateMachine(stateMachineArn);
```

```
        Console.WriteLine("State machine deleted.");

        Console.WriteLine("Deleting the activity...");
        await stepFunctionsWrapper.DeleteActivity(activityArn);
        Console.WriteLine("Activity deleted.");

        Console.WriteLine("The Amazon Step Functions scenario is now complete.");
    }

    static async Task<Role> GetOrCreateStateMachineRole(string roleName)
    {
        // Define the policy document for the role.
        var stateMachineRolePolicy = @"{
            ""Version"": ""2012-10-17"",
            ""Statement"": [{
                ""Sid"": "",
                ""Effect"": ""Allow"",
                ""Principal"": {
                    ""Service"": ""states.amazonaws.com""},
                ""Action"": ""sts:AssumeRole""}]}";

        var role = new Role();
        var roleExists = false;

        try
        {
            var getRoleResponse = await _iamService.GetRoleAsync(new GetRoleRequest
{ RoleName = roleName });
            roleExists = true;
            role = getRoleResponse.Role;
        }
        catch (NoSuchEntityException)
        {
            // The role doesn't exist. Create it.
            Console.WriteLine($"Role, {roleName} doesn't exist. Creating it...");
        }

        if (!roleExists)
        {
            var request = new CreateRoleRequest
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = stateMachineRolePolicy,
            };
        }
    }
}
```

```
        var createRoleResponse = await _iamService.CreateRoleAsync(request);
        role = createRoleResponse.Role;
    }

    return role;
}

namespace StepFunctionsBasics;

/// <summary>
/// Some useful methods to make screen display easier.
/// </summary>
public class UiMethods
{
    private readonly string _sepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Welcome to the AWS Step Functions Demo");

        Console.WriteLine("This example application will do the following:");
        Console.WriteLine("\t 1. Create an activity.");
        Console.WriteLine("\t 2. Create a state machine.");
        Console.WriteLine("\t 3. Start an execution.");
        Console.WriteLine("\t 4. Run the worker, then stop it.");
        Console.WriteLine("\t 5. List executions.");
        Console.WriteLine("\t 6. Clean up the resources created for the example.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        _ = Console.ReadLine();
    }
}
```

```
/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter"></param>
/// <returns></returns>
private 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);
}
}
```

Definieren Sie eine Klasse, die Zustandsautomaten- und Aktivitätsaktionen umschließt.

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

/// <summary>
/// Wrapper that performs AWS Step Functions actions.
/// </summary>
public class StepFunctionsWrapper
{
    private readonly IAmazonStepFunctions _amazonStepFunctions;
```



```
    /// <summary>
    /// The constructor for the StepFunctionsWrapper. Initializes the
    /// client object passed to it.
    /// </summary>
    /// <param name="amazonStepFunctions">An initialized Step Functions client
object.</param>
    public StepFunctionsWrapper(IAmazonStepFunctions amazonStepFunctions)
    {
        _amazonStepFunctions = amazonStepFunctions;
    }

    /// <summary>
    /// Create a Step Functions activity using the supplied name.
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }

    /// <summary>
    /// Create a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    /// machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    /// state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
    public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
    {
        var request = new CreateStateMachineRequest
        {
            Name = stateMachineName,
            Definition = definition,
            RoleArn = roleArn
        };
    }
}
```

```
        var response =
            await _amazonStepFunctions.CreateStateMachineAsync(request);
        return response.StateMachineArn;
    }

    /// <summary>
    /// Delete a Step Machine activity.
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the activity.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteActivity(string activityArn)
    {
        var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// state machine.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteStateMachine(string stateMachineArn)
    {
        var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
        { StateMachineArn = stateMachineArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Retrieve information about the specified Step Functions execution.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of the
    /// Step Functions execution.</param>
    /// <returns>The API response returned by the API.</returns>
    public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
    {
```

```
        var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
        return response;
    }

    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }

    /// <summary>
    /// Retrieve a task with the specified Step Functions activity
    /// with the specified Amazon Resource Name (ARN).
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the Step Functions activity.</param>
    /// <param name="workerName">The name of the Step Functions worker.</param>
    /// <returns>The response from the Step Functions activity.</returns>
    public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
    {
        var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
        { ActivityArn = activityArn, WorkerName = workerName });
        return response;
    }

    /// <summary>
    /// List the Step Functions activities for the current account.
    /// </summary>
    /// <returns>A list of ActivityListItems.</returns>
```

```
public async Task<List<ActivityListItem>> ListActivitiesAsync()
{
    var request = new ListActivitiesRequest();
    var activities = new List<ActivityListItem>();

    do
    {
        var response = await _amazonStepFunctions.ListActivitiesAsync(request);

        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }

        activities.AddRange(response.Activities);
    }
    while (request.NextToken is not null);

    return activities;
}

/// <summary>
/// Retrieve information about executions of a Step Functions
/// state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>A list of ExecutionListItem objects.</returns>
public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
{
    var executions = new List<ExecutionListItem>();
    ListExecutionsResponse response;
    var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

    do
    {
        response = await _amazonStepFunctions.ListExecutionsAsync(request);
        executions.AddRange(response.Executions);
        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }
    }
}
```

```
    }
    } while (response.NextToken is not null);

    return executions;
}

/// <summary>
/// Retrieve a list of Step Functions state machines.
/// </summary>
/// <returns>A list of StateMachineListItem objects.</returns>
public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
{
    var stateMachines = new List<StateMachineListItem>();
    var listStateMachinesPaginator =
        _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

    await foreach (var response in listStateMachinesPaginator.Responses)
    {
        stateMachines.AddRange(response.StateMachines);
    }

    return stateMachines;
}

/// <summary>
/// Indicate that the Step Functions task, indicated by the
/// task token, has completed successfully.
/// </summary>
/// <param name="taskToken">Identifies the task.</param>
/// <param name="taskResponse">The response received from executing the task.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
{
    var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
    { TaskToken = taskToken, Output = taskResponse });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

```
/// <summary>
/// Start execution of an AWS Step Functions state machine.
/// </summary>
/// <param name="executionName">The name to use for the execution.</param>
/// <param name="executionJson">The JSON string to pass for execution.</param>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
/// execution.</returns>
public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
{
    var executionRequest = new StartExecutionRequest
    {
        Input = executionJson,
        StateMachineArn = stateMachineArn
    };

    var response = await
_amazonStepFunctions.StartExecutionAsync(executionRequest);
    return response.ExecutionArn;
}

/// <summary>
/// Stop execution of a Step Functions workflow.
/// </summary>
/// <param name="executionArn">The Amazon Resource Name (ARN) of
/// the Step Functions execution to stop.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StopExecution(string executionArn)
{
    var response =
        await _amazonStepFunctions.StopExecutionAsync(new StopExecutionRequest
{ ExecutionArn = executionArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [CreateActivity](#)
 - [CreateStateMachine](#)
 - [DeleteActivity](#)
 - [DeleteStateMachine](#)
 - [DescribeExecution](#)
 - [DescribeStateMachine](#)
 - [GetActivityTask](#)
 - [ListActivities](#)
 - [ListStateMachines](#)
 - [SendTaskSuccess](#)
 - [StartExecution](#)
 - [StopExecution](#)

AWS STS -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS STS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Übernehmen einer Rolle

Das folgende Codebeispiel zeigt, wie Sie eine Rolle mit übernehmen AWS STS.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

namespace AssumeRoleExample
{
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
        id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
```



```
{
    // Create the SecurityToken client and then display the identity of the
    // default user.
    var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";

    var client = new
Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

    // Get and display the information about the identity of the default
    user.
    var callerIdRequest = new GetCallerIdentityRequest();
    var caller = await client.GetCallerIdentityAsync(callerIdRequest);
    Console.WriteLine($"Original Caller: {caller.Arn}");

    // Create the request to use with the AssumeRoleAsync call.
    var assumeRoleReq = new AssumeRoleRequest()
    {
        DurationSeconds = 1600,
        RoleSessionName = "Session1",
        RoleArn = roleArnToAssume
    };

    var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

    // Now create a new client based on the credentials of the caller
    assuming the role.
    var client2 = new AmazonSecurityTokenServiceClient(credentials:
assumeRoleRes.Credentials);

    // Get and display information about the caller that has assumed the
    defined role.
    var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
    Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
}
}
```

- Weitere API-Informationen finden Sie unter [AssumeRole](#) in der APIAWS SDK for .NET - Referenz für .

AWS Support -Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit verwenden AWS Support.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Support

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Support beginnen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using Amazon.AWSSupport;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public static class HelloSupport
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the AWS Support service.
    }
}
```

```
// Use your AWS profile name, or leave it blank to use the default profile.
// You must have one of the following AWS Support plans: Business,
Enterprise On-Ramp, or Enterprise. Otherwise, an exception will be thrown.
using var host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonAWSSupport>()
    ).Build();

// Now the client is available for injection.
var supportClient = host.Services.GetRequiredService<IAmazonAWSSupport>();

// You can use await and any of the async methods to get a response.
var response = await supportClient.DescribeServicesAsync();
Console.WriteLine($"\\tHello AWS Support! There are {response.Services.Count}
services available.");
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeServices](#) in der APIAWS SDK for .NET - Referenz für .

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

Hinzufügen einer Mitteilung zu einem Fall

Das folgende Codebeispiel zeigt, wie Sie einem Support-Fall eine - AWS Support Kommunikation mit einem Anhang hinzufügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add communication to a case, including optional attachment set ID and CC
email addresses.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <param name="body">Body text of the communication.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set.</param>
/// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
/// <returns>True if successful.</returns>
public async Task<bool> AddCommunicationToCase(string caseId, string body,
    string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
{
    var response = await _amazonSupport.AddCommunicationToCaseAsync(
        new AddCommunicationToCaseRequest()
        {
            CaseId = caseId,
            CommunicationBody = body,
            AttachmentSetId = attachmentSetId,
            CcEmailAddresses = ccEmailAddresses
        });
    return response.Result;
}
```

- Weitere API-Informationen finden Sie unter [AddCommunicationToCase](#) in der APIAWS SDK for .NET -Referenz für .

Hinzufügen eines Anhangs zu einem Set

Das folgende Codebeispiel zeigt, wie Sie einem Anhangssatz einen AWS Support Anhang hinzufügen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Add an attachment to a set, or create a new attachment set if one does not
exist.
/// </summary>
/// <param name="data">The data for the attachment.</param>
/// <param name="fileName">The file name for the attachment.</param>
/// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
/// <returns>The setId of the attachment.</returns>
public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
{
    var response = await _amazonSupport.AddAttachmentsToSetAsync(
        new AddAttachmentsToSetRequest
        {
            AttachmentSetId = attachmentSetId,
            Attachments = new List<Attachment>
            {
                new Attachment
                {
                    Data = data,
                    FileName = fileName
                }
            }
        });
    return response.AttachmentSetId;
}
```

- Weitere API-Informationen finden Sie unter [AddAttachmentsToSet](#) in der APIAWS SDK for .NET -Referenz für .

Erstellen eines Falls

Das folgende Codebeispiel zeigt, wie Sie einen neuen AWS Support Fall erstellen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a new support case.
/// </summary>
/// <param name="serviceCode">Service code for the new case.</param>
/// <param name="categoryCode">Category for the new case.</param>
/// <param name="severityCode">Severity code for the new case.</param>
/// <param name="subject">Subject of the new case.</param>
/// <param name="body">Body text of the new case.</param>
/// <param name="language">Optional language support for your case.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
/// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
/// <returns>The caseId of the new support case.</returns>
public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
{
    var response = await _amazonSupport.CreateCaseAsync(
        new CreateCaseRequest()
        {
            ServiceCode = serviceCode,
            CategoryCode = categoryCode,
            SeverityCode = severityCode,
            Subject = subject,
            Language = language,
            AttachmentSetId = attachmentSetId,
            IssueType = issueType,
            CommunicationBody = body
        });
    return response.CaseId;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [CreateCase](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben eines Anhangs

Das folgende Codebeispiel zeigt, wie Sie einen Anhang für einen AWS Support -Fall beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get description of a specific attachment.
/// </summary>
/// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
/// <returns>The attachment object.</returns>
public async Task<Attachment> DescribeAttachment(string attachmentId)
{
    var response = await _amazonSupport.DescribeAttachmentAsync(
        new DescribeAttachmentRequest()
        {
            AttachmentId = attachmentId
        });
    return response.Attachment;
}
```

- Weitere API-Informationen finden Sie unter [DescribeAttachment](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben von Fällen

Das folgende Codebeispiel zeigt, wie Sie - AWS Support Fälle beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get case details for a list of case ids, optionally with date filters.
/// </summary>
/// <param name="caseIds">The list of case IDs.</param>
/// <param name="displayId">Optional display ID.</param>
/// <param name="includeCommunication">True to include communication. Defaults
to true.</param>
/// <param name="includeResolvedCases">True to include resolved cases. Defaults
to false.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <param name="language">Optional language support for your case.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <returns>A list of CaseDetails.</returns>
public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
displayId = null, bool includeCommunication = true,
    bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
beforeTime = null,
    string language = "en")
{
    var results = new List<CaseDetails>();
    var paginateCases = _amazonSupport.Paginators.DescribeCases(
        new DescribeCasesRequest()
        {
            CaseIdList = caseIds,
            DisplayId = displayId,
            IncludeCommunications = includeCommunication,
            IncludeResolvedCases = includeResolvedCases,
```



```
        AfterTime = afterTime?.ToString("s"),
        BeforeTime = beforeTime?.ToString("s"),
        Language = language
    });
    // Get the entire list using the paginator.
    await foreach (var cases in paginateCases.Cases)
    {
        results.Add(cases);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [DescribeCases](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben von Mitteilungen

Das folgende Codebeispiel zeigt, wie Sie die AWS Support Kommunikation für einen Fall beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Describe the communications for a case, optionally with a date filter.
/// </summary>
/// <param name="caseId">The ID of the support case.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <returns>The list of communications for the case.</returns>
```

```
public async Task<List<Communication>> DescribeCommunications(string caseId,
DateTime? afterTime = null, DateTime? beforeTime = null)
{
    var results = new List<Communication>();
    var paginateCommunications =
    _amazonSupport.Paginators.DescribeCommunications(
        new DescribeCommunicationsRequest()
        {
            CaseId = caseId,
            AfterTime = afterTime?.ToString("s"),
            BeforeTime = beforeTime?.ToString("s")
        });
    // Get the entire list using the paginator.
    await foreach (var communications in paginateCommunications.Communications)
    {
        results.Add(communications);
    }
    return results;
}
```

- Weitere API-Informationen finden Sie unter [DescribeCommunications](#) in der APIAWS SDK for .NET -Referenz für .

Beschreiben von Services

Das folgende Codebeispiel zeigt, wie Sie die Liste der - AWS Services beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the descriptions of AWS services.
/// </summary>
```

```
/// <param name="name">Optional language for services.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <returns>The list of AWS service descriptions.</returns>
public async Task<List<Service>> DescribeServices(string language = "en")
{
    var response = await _amazonSupport.DescribeServicesAsync(
        new DescribeServicesRequest()
        {
            Language = language
        });
    return response.Services;
}
```

- Weitere API-Informationen finden Sie unter [DescribeServices](#) in der APIAWS SDK for .NET - Referenz für .

Beschreiben der Schweregrade

Das folgende Codebeispiel zeigt, wie Sie AWS Support Schweregrade beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get the descriptions of support severity levels.
/// </summary>
/// <param name="name">Optional language for severity levels.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <returns>The list of support severity levels.</returns>
public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
{
    var response = await _amazonSupport.DescribeSeverityLevelsAsync(
```

```
        new DescribeSeverityLevelsRequest()
        {
            Language = language
        });
    return response.SeverityLevels;
}
```

- Weitere API-Informationen finden Sie unter [DescribeSeverityLevels](#) in der APIAWS SDK for .NET -Referenz für .

Lösen eines Falls

Das folgende Codebeispiel zeigt, wie Sie einen - AWS Support Fall lösen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Resolve a support case by caseId.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}
```

- Weitere API-Informationen finden Sie unter [ResolveCase](#) in der APIAWS SDK for .NET - Referenz für .

Szenarien

Erste Schritte mit Fällen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie verfügbare Services und Schweregrade für Fälle ab und zeigen Sie sie an.
- Erstellen Sie einen Supportfall mit einem ausgewählten Service, einer ausgewählten Kategorie und einem ausgewählten Schweregrad.
- Rufen Sie eine Liste der offenen Fälle für den aktuellen Tag ab und zeigen Sie sie an.
- Fügen Sie dem neuen Fall einen Anhangssatz und eine Mitteilung hinzu.
- Beschreiben Sie den neuen Anhang und die Mitteilung für den Fall.
- Lösen Sie den Fall.
- Rufen Sie eine Liste der gelösten Fälle für den aktuellen Tag ab und zeigen Sie sie an.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
/// <summary>
/// Hello AWS Support example.
/// </summary>
public static class SupportCaseScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.
```

To use the AWS Support API, you must have one of the following AWS Support plans: Business, Enterprise On-Ramp, or Enterprise.

This .NET example performs the following tasks:

1. Get and display services. Select a service from the list.
 2. Select a category from the selected service.
 3. Get and display severity levels and select a severity level from the list.
 4. Create a support case using the selected service, category, and severity level.
 5. Get and display a list of open support cases for the current day.
 6. Create an attachment set with a sample text file to add to the case.
 7. Add a communication with the attachment to the support case.
 8. List the communications of the support case.
 9. Describe the attachment set.
 10. Resolve the support case.
 11. Get a list of resolved cases for the current day.
- */

```
private static SupportWrapper _supportWrapper = null!;

static async Task Main(string[] args)
{
    // Set up dependency injection for the AWS Support service.
    // 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<IAmazonAWSSupport>(new AWSOptions() { Profile
= "default" }))
            .AddTransient<SupportWrapper>()
        )
        .Build();

    var logger = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
    }).CreateLogger(typeof(SupportCaseScenario));

    _supportWrapper = host.Services.GetRequiredService<SupportWrapper>();
}
```

```
Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the AWS Support case example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    var apiSupported = await _supportWrapper.VerifySubscription();
    if (!apiSupported)
    {
        logger.LogError("You must have a Business, Enterprise On-Ramp, or
Enterprise Support " +
                        "plan to use the AWS Support API. \n\tPlease
upgrade your subscription to run these examples.");
        return;
    }

    var service = await DisplayAndSelectServices();

    var category = DisplayAndSelectCategories(service);

    var severityLevel = await DisplayAndSelectSeverity();

    var caseId = await CreateSupportCase(service, category, severityLevel);

    await DescribeTodayOpenCases();

    var attachmentSetId = await CreateAttachmentSet();

    await AddCommunicationToCase(attachmentSetId, caseId);

    var attachmentId = await ListCommunicationsForCase(caseId);

    await DescribeCaseAttachment(attachmentId);

    await ResolveCase(caseId);

    await DescribeTodayResolvedCases();

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("AWS Support case example scenario complete.");
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
```

```
        logger.LogError(ex, "There was a problem executing the scenario.");
    }
}

/// <summary>
/// List some available services from AWS Support, and select a service for the
example.
/// </summary>
/// <returns>The selected service.</returns>
private static async Task<Service> DisplayAndSelectServices()
{
    Console.WriteLine(new string('-', 80));
    var services = await _supportWrapper.DescribeServices();
    Console.WriteLine($"AWS Support client returned {services.Count}
services.");

    Console.WriteLine($"1. Displaying first 10 services:");
    for (int i = 0; i < 10 && i < services.Count; i++)
    {
        Console.WriteLine($"  \t{i + 1}. {services[i].Name}");
    }

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > services.Count)
    {
        Console.WriteLine(
            "Select an example support service by entering a number from the
preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }
    Console.WriteLine(new string('-', 80));

    return services[choiceNumber - 1];
}

/// <summary>
/// List the available categories for a service and select a category for the
example.
/// </summary>
/// <param name="service">Service to use for displaying categories.</param>
/// <returns>The selected category.</returns>
private static Category DisplayAndSelectCategories(Service service)
{

```



```
        Console.WriteLine(new string('-', 80));

        Console.WriteLine($"2. Available support categories for Service
\"{service.Name}\"");
        for (int i = 0; i < service.Categories.Count; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {service.Categories[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > service.Categories.Count)
        {
            Console.WriteLine(
                "Select an example support category by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        Console.WriteLine(new string('-', 80));

        return service.Categories[choiceNumber - 1];
    }

    /// <summary>
    /// List available severity levels from AWS Support, and select a level for the
example.
    /// </summary>
    /// <returns>The selected severity level.</returns>
    private static async Task<SeverityLevel> DisplayAndSelectSeverity()
    {
        Console.WriteLine(new string('-', 80));
        var severityLevels = await _supportWrapper.DescribeSeverityLevels();

        Console.WriteLine($"3. Get and display available severity levels:");
        for (int i = 0; i < 10 && i < severityLevels.Count; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {severityLevels[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > severityLevels.Count)
        {
            Console.WriteLine(
```

```
        "Select an example severity level by entering a number from the
preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }
    Console.WriteLine(new string('-', 80));

    return severityLevels[choiceNumber - 1];
}

/// <summary>
/// Create an example support case.
/// </summary>
/// <param name="service">Service to use for the new case.</param>
/// <param name="category">Category to use for the new case.</param>
/// <param name="severity">Severity to use for the new case.</param>
/// <returns>The caseId of the new support case.</returns>
private static async Task<string> CreateSupportCase(Service service,
    Category category, SeverityLevel severity)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"4. Create an example support case" +
        $" with the following settings:" +
        $" \n\tService: {service.Name}, Category: {category.Name}
" +
        $"and Severity Level: {severity.Name}.");
    var caseId = await _supportWrapper.CreateCase(service.Code, category.Code,
severity.Code,
        "Example case for testing, ignore.", "This is my example support
case.");

    Console.WriteLine($" \tNew case created with ID {caseId}");

    Console.WriteLine(new string('-', 80));

    return caseId;
}

/// <summary>
/// List open cases for the current day.
/// </summary>
/// <returns>Async task.</returns>
private static async Task DescribeTodayOpenCases()
{
```

```
    Console.WriteLine($"5. List the open support cases for the current day.");
    // Describe the cases. If it is empty, try again and allow time for the new
case to appear.
    List<CaseDetails> currentOpenCases = null!;
    while (currentOpenCases == null || currentOpenCases.Count == 0)
    {
        Thread.Sleep(1000);
        currentOpenCases = await _supportWrapper.DescribeCases(
            new List<string>(),
            null,
            false,
            false,
            DateTime.UtcNow.Date,
            DateTime.UtcNow);
    }

    foreach (var openCase in currentOpenCases)
    {
        Console.WriteLine($"\\tCase: {openCase.CaseId} created
{openCase.TimeCreated}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create an attachment set for a support case.
/// </summary>
/// <returns>The attachment set id.</returns>
private static async Task<string> CreateAttachmentSet()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"6. Create an attachment set for a support case.");
    var fileName = "example_attachment.txt";

    // Create the file if it does not already exist.
    if (!File.Exists(fileName))
    {
        await using StreamWriter sw = File.CreateText(fileName);
        await sw.WriteLineAsync(
            "This is a sample file for attachment to a support case.");
    }
}
```

```
        await using var ms = new MemoryStream(await
File.ReadAllBytesAsync(fileName));

        var attachmentSetId = await _supportWrapper.AddAttachmentToSet(
            ms,
            fileName);

        Console.WriteLine($"\\tNew attachment set created with id: \\n
\\t{attachmentSetId.Substring(0, 65)}...");

        Console.WriteLine(new string('-', 80));

        return attachmentSetId;
    }

    /// <summary>
    /// Add an attachment set and communication to a case.
    /// </summary>
    /// <param name="attachmentSetId">Id of the attachment set.</param>
    /// <param name="caseId">Id of the case to receive the attachment set.</param>
    /// <returns>Async task.</returns>
    private static async Task AddCommunicationToCase(string attachmentSetId, string
caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Add attachment set and communication to {caseId}.");

        await _supportWrapper.AddCommunicationToCase(
            caseId,
            "This is an example communication added to a support case.",
            attachmentSetId);

        Console.WriteLine($"\\tNew attachment set and communication added to
{caseId}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the communications for a case.
    /// </summary>
    /// <param name="caseId">Id of the case to describe.</param>
    /// <returns>An attachment id.</returns>
    private static async Task<string> ListCommunicationsForCase(string caseId)
```

```
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"8. List communications for case {caseId}.");

    var communications = await _supportWrapper.DescribeCommunications(caseId);
    var attachmentId = "";
    foreach (var communication in communications)
    {
        Console.WriteLine(
            $"{communication.TimeCreated} has
{communication.AttachmentSet.Count} attachments.");
        if (communication.AttachmentSet.Any())
        {
            attachmentId = communication.AttachmentSet.First().AttachmentId;
        }
    }

    Console.WriteLine(new string('-', 80));
    return attachmentId;
}

/// <summary>
/// Describe an attachment by id.
/// </summary>
/// <param name="attachmentId">Id of the attachment to describe.</param>
/// <returns>Async task.</returns>
private static async Task DescribeCaseAttachment(string attachmentId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Describe the attachment set.");

    var attachment = await _supportWrapper.DescribeAttachment(attachmentId);
    var data = Encoding.ASCII.GetString(attachment.Data.ToArray());
    Console.WriteLine($"{attachment.FileName} with data:
\n\t{data}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Resolve the support case.
/// </summary>
/// <param name="caseId">Id of the case to resolve.</param>
/// <returns>Async task.</returns>
```

```
private static async Task ResolveCase(string caseId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"10. Resolve case {caseId}.");

    var status = await _supportWrapper.ResolveCase(caseId);
    Console.WriteLine($"\\tCase {caseId} has final status {status}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List resolved cases for the current day.
/// </summary>
/// <returns>Async Task.</returns>
private static async Task DescribeTodayResolvedCases()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"11. List the resolved support cases for the current
day.");
    var currentCases = await _supportWrapper.DescribeCases(
        new List<string>(),
        null,
        false,
        true,
        DateTime.UtcNow.Date,
        DateTime.UtcNow);

    foreach (var currentCase in currentCases)
    {
        if (currentCase.Status == "resolved")
        {
            Console.WriteLine(
                $"\\tCase: {currentCase.CaseId}: status {currentCase.Status}");
        }
    }

    Console.WriteLine(new string('-', 80));
}
}
```

Wrapper-Methoden, die vom Szenario für AWS Support Aktionen verwendet werden.

```
/// <summary>
/// Wrapper methods to use AWS Support for working with support cases.
/// </summary>
public class SupportWrapper
{
    private readonly IAmazonAWSSupport _amazonSupport;
    public SupportWrapper(IAmazonAWSSupport amazonSupport)
    {
        _amazonSupport = amazonSupport;
    }

    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
    /// Currently "en" (English) and "ja" (Japanese) are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = language
            });
        return response.Services;
    }

    /// <summary>
    /// Get the descriptions of support severity levels.
    /// </summary>
    /// <param name="name">Optional language for severity levels.
    /// Currently "en" (English) and "ja" (Japanese) are supported.</param>
    /// <returns>The list of support severity levels.</returns>
    public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
    {
        var response = await _amazonSupport.DescribeSeverityLevelsAsync(
            new DescribeSeverityLevelsRequest()
            {
```

```
        Language = language
    });
    return response.SeverityLevels;
}

/// <summary>
/// Create a new support case.
/// </summary>
/// <param name="serviceCode">Service code for the new case.</param>
/// <param name="categoryCode">Category for the new case.</param>
/// <param name="severityCode">Severity code for the new case.</param>
/// <param name="subject">Subject of the new case.</param>
/// <param name="body">Body text of the new case.</param>
/// <param name="language">Optional language support for your case.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
/// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
/// <returns>The caseId of the new support case.</returns>
public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
{
    var response = await _amazonSupport.CreateCaseAsync(
        new CreateCaseRequest()
        {
            ServiceCode = serviceCode,
            CategoryCode = categoryCode,
            SeverityCode = severityCode,
            Subject = subject,
            Language = language,
            AttachmentSetId = attachmentSetId,
            IssueType = issueType,
            CommunicationBody = body
        });
    return response.CaseId;
}
```



```
    /// <summary>
    /// Add an attachment to a set, or create a new attachment set if one does not
    exist.
    /// </summary>
    /// <param name="data">The data for the attachment.</param>
    /// <param name="fileName">The file name for the attachment.</param>
    /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
    new attachment set if empty.</param>
    /// <returns>The setId of the attachment.</returns>
    public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
    string? attachmentSetId = null)
    {
        var response = await _amazonSupport.AddAttachmentsToSetAsync(
            new AddAttachmentsToSetRequest
            {
                AttachmentSetId = attachmentSetId,
                Attachments = new List<Attachment>
                {
                    new Attachment
                    {
                        Data = data,
                        FileName = fileName
                    }
                }
            });
        return response.AttachmentSetId;
    }

    /// <summary>
    /// Get description of a specific attachment.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment, usually fetched by
    describing the communications of a case.</param>
    /// <returns>The attachment object.</returns>
    public async Task<Attachment> DescribeAttachment(string attachmentId)
    {
        var response = await _amazonSupport.DescribeAttachmentAsync(
            new DescribeAttachmentRequest()
            {
                AttachmentId = attachmentId
            });
        return response.Attachment;
    }
}
```

```
}

/// <summary>
/// Add communication to a case, including optional attachment set ID and CC
email addresses.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <param name="body">Body text of the communication.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set.</param>
/// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
/// <returns>True if successful.</returns>
public async Task<bool> AddCommunicationToCase(string caseId, string body,
    string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
{
    var response = await _amazonSupport.AddCommunicationToCaseAsync(
        new AddCommunicationToCaseRequest()
        {
            CaseId = caseId,
            CommunicationBody = body,
            AttachmentSetId = attachmentSetId,
            CcEmailAddresses = ccEmailAddresses
        });
    return response.Result;
}

/// <summary>
/// Describe the communications for a case, optionally with a date filter.
/// </summary>
/// <param name="caseId">The ID of the support case.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <returns>The list of communications for the case.</returns>
public async Task<List<Communication>> DescribeCommunications(string caseId,
    DateTime? afterTime = null, DateTime? beforeTime = null)
{
    var results = new List<Communication>();
    var paginateCommunications =
    _amazonSupport.Paginators.DescribeCommunications(
```

```
        new DescribeCommunicationsRequest()
        {
            CaseId = caseId,
            AfterTime = afterTime?.ToString("s"),
            BeforeTime = beforeTime?.ToString("s")
        });
    // Get the entire list using the paginator.
    await foreach (var communications in paginateCommunications.Communications)
    {
        results.Add(communications);
    }
    return results;
}

/// <summary>
/// Get case details for a list of case ids, optionally with date filters.
/// </summary>
/// <param name="caseIds">The list of case IDs.</param>
/// <param name="displayId">Optional display ID.</param>
/// <param name="includeCommunication">True to include communication. Defaults
to true.</param>
/// <param name="includeResolvedCases">True to include resolved cases. Defaults
to false.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <param name="language">Optional language support for your case.
/// Currently "en" (English) and "ja" (Japanese) are supported.</param>
/// <returns>A list of CaseDetails.</returns>
public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
displayId = null, bool includeCommunication = true,
    bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
beforeTime = null,
    string language = "en")
{
    var results = new List<CaseDetails>();
    var paginateCases = _amazonSupport.Paginators.DescribeCases(
        new DescribeCasesRequest()
        {
            CaseIdList = caseIds,
            DisplayId = displayId,
```

```
        IncludeCommunications = includeCommunication,
        IncludeResolvedCases = includeResolvedCases,
        AfterTime = afterTime?.ToString("s"),
        BeforeTime = beforeTime?.ToString("s"),
        Language = language
    });
// Get the entire list using the paginator.
await foreach (var cases in paginateCases.Cases)
{
    results.Add(cases);
}
return results;
}

/// <summary>
/// Resolve a support case by caseId.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}

/// <summary>
/// Verify the support level for AWS Support API access.
/// </summary>
/// <returns>True if the subscription level supports API access.</returns>
public async Task<bool> VerifySubscription()
{
    try
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = "en"
            }
        );
    }
}
```

```
        });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Amazon.AWSSupport.AmazonAWSSupportException ex)
    {
        if (ex.ErrorCode == "SubscriptionRequiredException")
        {
            return false;
        }
        else throw;
    }
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for .NET -API-Referenz.
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)
 - [DescribeCases](#)
 - [DescribeCommunications](#)
 - [DescribeServices](#)
 - [DescribeSeverityLevels](#)
 - [ResolveCase](#)

Amazon TranscribeBeispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Transcribe verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Erstellen eines benutzerdefinierten Vokabulars

Das folgende Codebeispiel zeigt, wie Sie ein benutzerdefiniertes Amazon Transcribe-Vokabular erstellen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Create a custom vocabulary using a list of phrases. Custom vocabularies
/// improve transcription accuracy for one or more specific words.
/// </summary>
/// <param name="languageCode">The language code of the vocabulary.</param>
/// <param name="phrases">Phrases to use in the vocabulary.</param>
/// <param name="vocabularyName">Name for the vocabulary.</param>
/// <returns>The state of the custom vocabulary.</returns>
public async Task<VocabularyState> CreateCustomVocabulary(LanguageCode
languageCode,
    List<string> phrases, string vocabularyName)
{
    var response = await _amazonTranscribeService.CreateVocabularyAsync(
        new CreateVocabularyRequest
        {
            LanguageCode = languageCode,
            Phrases = phrases,
            VocabularyName = vocabularyName
        });
}
```

```
        return response.VocabularyState;
    }
```

- Weitere API-Informationen finden Sie unter [CreateVocabulary](#) in der APIAWS SDK for .NET - Referenz für .

Löschen eines benutzerdefinierten Vokabulars

Das folgende Codebeispiel zeigt, wie Sie ein benutzerdefiniertes Amazon Transcribe-Vokabular löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete an existing custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.DeleteVocabularyAsync(
        new DeleteVocabularyRequest
        {
            VocabularyName = vocabularyName
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteVocabulary](#) in der APIAWS SDK for .NET - Referenz für .

Einen Auftrag für medizinische Transkription löschen

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Transcribe-Medical-Transkriptionsauftrag löschen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a medical transcription job. Also deletes the transcript associated
with the job.
/// </summary>
/// <param name="jobName">Name of the medical transcription job to delete.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteMedicalTranscriptionJob(string jobName)
{
    var response = await
        _amazonTranscribeService.DeleteMedicalTranscriptionJobAsync(
            new DeleteMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName
            });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteMedicalTranscriptionJob](#) in der APIAWS SDK for .NET -Referenz für .

Löschen eines Transkriptionsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag löschen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Delete a transcription job. Also deletes the transcript associated with the
job.
/// </summary>
/// <param name="jobName">Name of the transcription job to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTranscriptionJob(string jobName)
{
    var response = await _amazonTranscribeService.DeleteTranscriptionJobAsync(
        new DeleteTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Weitere API-Informationen finden Sie unter [DeleteTranscriptionJob](#) in der APIAWS SDK for .NET -Referenz für .

Ein benutzerdefiniertes Vokabular erhalten

Das folgende Codebeispiel zeigt, wie Sie ein benutzerdefiniertes Amazon Transcribe-Vokabular abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
/// <summary>
/// Get information about a custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary.</param>
/// <returns>The state of the custom vocabulary.</returns>
public async Task<VocabularyState> GetCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.GetVocabularyAsync(
        new GetVocabularyRequest()
        {
            VocabularyName = vocabularyName
        });
    return response.VocabularyState;
}
```

- Weitere API-Informationen finden Sie unter [GetVocabulary](#) in der APIAWS SDK for .NET - Referenz für .

Einen Transkriptionsauftrag erhalten

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag abrufen.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Get details about a transcription job.
/// </summary>
/// <param name="jobName">A unique name for the transcription job.</param>
/// <returns>A TranscriptionJob instance with information on the requested
job.</returns>
public async Task<TranscriptionJob> GetTranscriptionJob(string jobName)
{
    var response = await _amazonTranscribeService.GetTranscriptionJobAsync(
        new GetTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName
        });
    return response.TranscriptionJob;
}
```

- Weitere API-Informationen finden Sie unter [GetTranscriptionJob](#) in der APIAWS SDK for .NET - Referenz für .

Benutzerdefinierte Vokabulare auflisten

Das folgende Codebeispiel zeigt, wie Sie benutzerdefinierte Amazon Transcribe-Vokabulare auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List custom vocabularies for the current account. Optionally specify a name
/// filter and a specific state to filter the vocabularies list.
/// </summary>
/// <param name="nameContains">Optional string the vocabulary name must
contain.</param>
```

```
/// <param name="stateEquals">Optional state of the vocabulary.</param>
/// <returns>List of information about the vocabularies.</returns>
public async Task<List<VocabularyInfo>> ListCustomVocabularies(string?
nameContains = null,
    VocabularyState? stateEquals = null)
{
    var response = await _amazonTranscribeService.ListVocabulariesAsync(
        new ListVocabulariesRequest()
        {
            NameContains = nameContains,
            StateEquals = stateEquals
        });
    return response.Vocabularies;
}
```

- Weitere API-Informationen finden Sie unter [ListVocabularies](#) in der APIAWS SDK for .NET - Referenz für .

Auflisten medizinischer Transkriptionsjobs

Das folgende Codebeispiel zeigt, wie Sie Amazon Transcribe Medical-Transkriptionsaufträge auflisten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// List medical transcription jobs, optionally with a name filter.
/// </summary>
/// <param name="jobNameContains">Optional name filter for the medical
transcription jobs.</param>
/// <returns>A list of summaries about medical transcription jobs.</returns>
```

```

    public async Task<List<MedicalTranscriptionJobSummary>>
    ListMedicalTranscriptionJobs(
        string? jobNameContains = null)
    {
        var response = await
        _amazonTranscribeService.ListMedicalTranscriptionJobsAsync(
            new ListMedicalTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.MedicalTranscriptionJobSummaries;
    }

```

- Weitere API-Informationen finden Sie unter [ListMedicalTranscriptionJobs](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Transkriptionsaufträgen

Das folgende Codebeispiel zeigt, wie AmazonAmazon TranscribeTranskriptionsaufträge aufgelistet werden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

    /// <summary>
    /// List transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the transcription
    jobs.</param>
    /// <returns>A list of transcription job summaries.</returns>
    public async Task<List<TranscriptionJobSummary>> ListTranscriptionJobs(string?
    jobNameContains = null)
    {

```

```
var response = await _amazonTranscribeService.ListTranscriptionJobsAsync(  
    new ListTranscriptionJobsRequest()  
    {  
        JobNameContains = jobNameContains  
    });  
return response.TranscriptionJobSummaries;  
}
```

- Weitere API-Informationen finden Sie unter [ListTranscriptionJobs](#) in der APIAWS SDK for .NET -Referenz für .

Einen Auftrag für medizinische Transkription starten

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Transcribe-Medical-Transkriptionsauftrag starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>  
/// Start a medical transcription job for a media file. This method returns  
/// as soon as the job is started.  
/// </summary>  
/// <param name="jobName">A unique name for the medical transcription job.</  
param>  
/// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3  
location.</param>  
/// <param name="mediaFormat">The format of the media file.</param>  
/// <param name="outputBucketName">Location for the output, typically an Amazon  
S3 location.</param>  
/// <param name="transcriptionType">Conversation or dictation transcription  
type.</param>
```


```
/// <returns>A MedicalTranscriptionJob instance with information on the new job.</
returns>
public async Task<MedicalTranscriptionJob> StartMedicalTranscriptionJob(
    string jobName, string mediaFileUri,
    MediaFormat mediaFormat, string outputBucketName,
    Amazon.TranscribeService.Type transcriptionType)
{
    var response = await
    _amazonTranscribeService.StartMedicalTranscriptionJobAsync(
        new StartMedicalTranscriptionJobRequest()
        {
            MedicalTranscriptionJobName = jobName,
            Media = new Media()
            {
                MediaFileUri = mediaFileUri
            },
            MediaFormat = mediaFormat,
            LanguageCode =
                LanguageCode
                .EnUS, // The value must be en-US for medical
transcriptions.
            OutputBucketName = outputBucketName,
            OutputKey =
                jobName, // The value is a key used to fetch the output of the
transcription.
            Specialty = Specialty.PRIMARYCARE, // The value PRIMARYCARE must be
set.
            Type = transcriptionType
        });
    return response.MedicalTranscriptionJob;
}
```

- Weitere API-Informationen finden Sie unter [StartMedicalTranscriptionJob](#) in der APIAWS SDK for .NET -Referenz für .

Einen Transkriptionsauftrag starten

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag starten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Start a transcription job for a media file. This method returns
/// as soon as the job is started.
/// </summary>
/// <param name="jobName">A unique name for the transcription job.</param>
/// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
location.</param>
/// <param name="mediaFormat">The format of the media file.</param>
/// <param name="languageCode">The language code of the media file, such as en-
US.</param>
/// <param name="vocabularyName">Optional name of a custom vocabulary.</param>
/// <returns>A TranscriptionJob instance with information on the new job.</
returns>
public async Task<TranscriptionJob> StartTranscriptionJob(string jobName, string
mediaFileUri,
    MediaFormat mediaFormat, LanguageCode languageCode, string? vocabularyName)
{
    var response = await _amazonTranscribeService.StartTranscriptionJobAsync(
        new StartTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName,
            Media = new Media()
            {
                MediaFileUri = mediaFileUri
            },
            MediaFormat = mediaFormat,
            LanguageCode = languageCode,
            Settings = vocabularyName != null ? new Settings()
            {
                VocabularyName = vocabularyName
            } : null
        });
    return response.TranscriptionJob;
}
```



```
}
```

- Weitere API-Informationen finden Sie unter [StartTranscriptionJob](#) in der APIAWS SDK for .NET -Referenz für .

Ein benutzerdefiniertes Vokabular aktualisieren

Das folgende Codebeispiel zeigt, wie Sie ein benutzerdefiniertes Amazon Transcribe-Vokabular aktualisieren.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// <summary>
/// Update a custom vocabulary with new values. Update overwrites all existing
information.
/// </summary>
/// <param name="languageCode">The language code of the vocabulary.</param>
/// <param name="phrases">Phrases to use in the vocabulary.</param>
/// <param name="vocabularyName">Name for the vocabulary.</param>
/// <returns>The state of the custom vocabulary.</returns>
public async Task<VocabularyState> UpdateCustomVocabulary(LanguageCode
languageCode,
    List<string> phrases, string vocabularyName)
{
    var response = await _amazonTranscribeService.UpdateVocabularyAsync(
        new UpdateVocabularyRequest()
        {
            LanguageCode = languageCode,
            Phrases = phrases,
            VocabularyName = vocabularyName
        });
    return response.VocabularyState;
}
```

```
}
```

- Weitere API-Informationen finden Sie unter [UpdateVocabulary](#) in der APIAWS SDK for .NET - Referenz für .

Amazon Translate-Beispiele mit AWS SDK for .NET

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen durchführen und gängige Szenarien implementieren, indem Sie die AWS SDK for .NET mit Amazon Translate verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

Beschreiben eines Übersetzungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Translate-Übersetzungsauftrag beschreiben.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// The following example shows how to retrieve the details of
/// a text translation job using Amazon Translate.
/// </summary>
public class DescribeTextTranslation
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();

        // The Job Id is generated when the text translation job is started
        // with a call to the StartTextTranslationJob method.
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new DescribeTextTranslationJobRequest
        {
            JobId = jobId,
        };

        var jobProperties = await DescribeTranslationJobAsync(client, request);

        DisplayTranslationJobDetails(jobProperties);
    }

    /// <summary>
    /// Retrieve information about an Amazon Translate text translation job.
    /// </summary>
    /// <param name="client">The initialized Amazon Translate client object.</
param>
    /// <param name="request">The DescribeTextTranslationJobRequest object.</
param>
    /// <returns>The TextTranslationJobProperties object containing
    /// information about the text translation job.</returns>
    public static async Task<TextTranslationJobProperties>
DescribeTranslationJobAsync(
    AmazonTranslateClient client,
    DescribeTextTranslationJobRequest request)
    {
```

```
        var response = await client.DescribeTextTranslationJobAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            return response.TextTranslationJobProperties;
        }
        else
        {
            return null;
        }
    }

    /// <summary>
    /// Displays the properties of the text translation job.
    /// </summary>
    /// <param name="jobProperties">The properties of the text translation
    /// job returned by the call to DescribeTextTranslationJobAsync.</param>
    public static void DisplayTranslationJobDetails(TextTranslationJobProperties
jobProperties)
    {
        if (jobProperties is null)
        {
            Console.WriteLine("No text translation job properties found.");
            return;
        }

        // Display the details of the text translation job.
        Console.WriteLine($"{jobProperties.JobId}: {jobProperties.JobName}");
    }
}
```

- Weitere API-Informationen finden Sie unter [DescribeTextTranslationJob](#) in der APIAWS SDK for .NET -Referenz für .

Auflisten von Übersetzungsaufträgen

Das folgende Codebeispiel zeigt, wie Sie die Amazon Translate-Übersetzungsaufträge auflisten.

AWS SDK for .NET

 Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// List Amazon Translate translation jobs, along with details about each job.
/// </summary>
public class ListTranslationJobs
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var filter = new TextTranslationJobFilter
        {
            JobStatus = "COMPLETED",
        };

        var request = new ListTextTranslationJobsRequest
        {
            MaxResults = 10,
            Filter = filter,
        };

        await ListJobsAsync(client, request);
    }

    /// <summary>
    /// List Amazon Translate text translation jobs.
    /// </summary>
    /// <param name="client">The initialized Amazon Translate client object.</
param>
    /// <param name="request">An Amazon Translate
```

```

    /// ListTextTranslationJobsRequest object detailing which text
    /// translation jobs are of interest.</param>
    public static async Task ListJobsAsync(
        AmazonTranslateClient client,
        ListTextTranslationJobsRequest request)
    {
        ListTextTranslationJobsResponse response;

        do
        {
            response = await client.ListTextTranslationJobsAsync(request);

            ShowTranslationJobDetails(response.TextTranslationJobPropertiesList);

            request.NextToken = response.NextToken;
        }
        while (response.NextToken is not null);
    }

    /// <summary>
    /// List existing translation job details.
    /// </summary>
    /// <param name="properties">A list of Amazon Translate text
    /// translation jobs.</param>
    public static void
    ShowTranslationJobDetails(List<TextTranslationJobProperties> properties)
    {
        properties.ForEach(prop =>
        {
            Console.WriteLine($"{prop.JobId}: {prop.JobName}");
            Console.WriteLine($"Status: {prop.JobStatus}");
            Console.WriteLine($"Submitted time: {prop.SubmittedTime}");
        });
    }
}

```

- Weitere API-Informationen finden Sie unter [ListTextTranslationJobs](#) in der APIAWS SDK for .NET -Referenz für .

Starten eines Übersetzungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Translate-Übersetzungsauftrag starten.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// This example shows how to use Amazon Translate to process the files in
/// an Amazon Simple Storage Service (Amazon S3) bucket. The translated results
/// will also be stored in an Amazon S3 bucket.
/// </summary>
public class BatchTranslate
{
    public static async Task Main()
    {
        var contentType = "text/plain";

        // Set this variable to an S3 bucket location with a folder."
        // Input files must be in a folder and not at the bucket root."
        var s3InputUri = "s3://DOC-EXAMPLE-BUCKET1/FOLDER/";
        var s3OutputUri = "s3://DOC-EXAMPLE-BUCKET2/";

        // This role must have permissions to read the source bucket and to read
and
        // write to the destination bucket where the translated text will be
stored.
        var dataAccessRoleArn = "arn:aws:iam::0123456789ab:role/
S3TranslateRole";

        var client = new AmazonTranslateClient();
```

```
var inputConfig = new InputDataConfig
{
    ContentType = contentType,
    S3Uri = s3InputUri,
};

var outputConfig = new OutputDataConfig
{
    S3Uri = s3OutputUri,
};

var request = new StartTextTranslationJobRequest
{
    JobName = "ExampleTranslationJob",
    DataAccessRoleArn = dataAccessRoleArn,
    InputDataConfig = inputConfig,
    OutputDataConfig = outputConfig,
    SourceLanguageCode = "en",
    TargetLanguageCodes = new List<string> { "fr" },
};

var response = await StartTextTranslationAsync(client, request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"{response.JobId}: {response.JobStatus}");
}
}

/// <summary>
/// Start the Amazon Translate text translation job.
/// </summary>
/// <param name="client">The initialized AmazonTranslateClient object.</
param>
/// <param name="request">The request object that includes details such
/// as source and destination bucket names and the IAM Role that will
/// be used to access the buckets.</param>
/// <returns>The StartTextTranslationResponse object that includes the
/// details of the request response.</returns>
public static async Task<StartTextTranslationJobResponse>
StartTextTranslationAsync(AmazonTranslateClient client,
StartTextTranslationJobRequest request)
{
    var response = await client.StartTextTranslationJobAsync(request);
```



```
        return response;
    }
}
```

- Weitere API-Informationen finden Sie unter [StartTextTranslationJob](#) in der APIAWS SDK for .NET -Referenz für .

Stoppen eines Übersetzungsauftrags

Das folgende Codebeispiel zeigt, wie Sie einen Amazon Translate-Übersetzungsauftrag beenden.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

/// <summary>
/// Shows how to stop a running Amazon Translation Service text translation
/// job.
/// </summary>
public class StopTextTranslationJob
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new StopTextTranslationJobRequest
        {
            JobId = jobId,
        };
    }
}
```

```
        await StopTranslationJobAsync(client, request);
    }

    /// <summary>
    /// Sends a request to stop a text translation job.
    /// </summary>
    /// <param name="client">Initialized AmazonTrnslateClient object.</param>
    /// <param name="request">The request object to be passed to the
    /// StopTextJobAsync method.</param>
    public static async Task StopTranslationJobAsync(
        AmazonTranslateClient client,
        StopTextTranslationJobRequest request)
    {
        var response = await client.StopTextTranslationJobAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{response.JobId} as status:
{response.JobStatus}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter [StopTextTranslationJob](#) in der APIAWS SDK for .NET -Referenz für .

Übersetzen von Text

Das folgende Codebeispiel zeigt, wie Sie Text mit Amazon Translate übersetzen.

AWS SDK for .NET

Note

Auf gibt es mehr GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
using Amazon.S3;
using Amazon.S3.Transfer;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// Take text from a file stored a Amazon Simple Storage Service (Amazon S3)
/// object and translate it using the Amazon Transfer Service.
/// </summary>
public class TranslateText
{
    public static async Task Main()
    {
        // If the region you want to use is different from the region
        // defined for the default user, supply it as a parameter to the
        // Amazon Translate client object constructor.
        var client = new AmazonTranslateClient();

        // Set the source language to "auto" to request Amazon Translate to
        // automatically detect te language of the source text.

        // You can get a list of the languages supposed by Amazon Translate
        // in the Amazon Translate Developer's Guide here:
        // https://docs.aws.amazon.com/translate/latest/dg/what-is.html
        string srcLang = "en"; // English.
        string destLang = "fr"; // French.

        // The Amazon Simple Storage Service (Amazon S3) bucket where the
        // source text file is stored.
        string srcBucket = "DOC-EXAMPLE-BUCKET";
        string srcTextFile = "source.txt";

        var srcText = await GetSourceTextAsync(srcBucket, srcTextFile);
        var destText = await TranslatingTextAsync(client, srcLang, destLang,
srcText);

        ShowText(srcText, destText);
    }

    /// <summary>
    /// Use the Amazon S3 TransferUtility to retrieve the text to translate
    /// from an object in an S3 bucket.
    /// </summary>
    /// <param name="srcBucket">The name of the S3 bucket where the
```

```
    /// text is stored.
    /// </param>
    /// <param name="srcTextFile">The key of the S3 object that
    /// contains the text to translate.</param>
    /// <returns>A string representing the source text.</returns>
    public static async Task<string> GetSourceTextAsync(string srcBucket, string
srcTextFile)
    {
        string srcText = string.Empty;

        var s3Client = new AmazonS3Client();
        TransferUtility utility = new TransferUtility(s3Client);

        using var stream = await utility.OpenStreamAsync(srcBucket,
srcTextFile);

        StreamReader file = new System.IO.StreamReader(stream);

        srcText = file.ReadToEnd();
        return srcText;
    }

    /// <summary>
    /// Use the Amazon Translate Service to translate the document from the
    /// source language to the specified destination language.
    /// </summary>
    /// <param name="client">The Amazon Translate Service client used to
    /// perform the translation.</param>
    /// <param name="srcLang">The language of the source text.</param>
    /// <param name="destLang">The destination language for the translated
    /// text.</param>
    /// <param name="text">A string representing the text to ranslate.</param>
    /// <returns>The text that has been translated to the destination
    /// language.</returns>
    public static async Task<string> TranslatingTextAsync(AmazonTranslateClient
client, string srcLang, string destLang, string text)
    {
        var request = new TranslateTextRequest
        {
            SourceLanguageCode = srcLang,
            TargetLanguageCode = destLang,
            Text = text,
        };
    }
```

```
        var response = await client.TranslateTextAsync(request);

        return response.TranslatedText;
    }

    /// <summary>
    /// Show the original text followed by the translated text.
    /// </summary>
    /// <param name="srcText">The original text to be translated.</param>
    /// <param name="destText">The translated text.</param>
    public static void ShowText(string srcText, string destText)
    {
        Console.WriteLine("Source text:");
        Console.WriteLine(srcText);
        Console.WriteLine();
        Console.WriteLine("Translated text:");
        Console.WriteLine(destText);
    }
}
```

- Weitere API-Informationen finden Sie unter [TranslateText](#) in der APIAWS SDK for .NET - Referenz für .

Serviceübergreifende Beispiele mit AWS SDK for .NET

Die folgenden Beispielanwendungen verwenden die AWS SDK for .NET , um über mehrere hinweg zu arbeiten AWS-Services.

Serviceübergreifende Beispiele zielen auf eine fortgeschrittene Erfahrung ab, um Sie beim Erstellen von Anwendungen zu unterstützen.

Beispiele

- [Erstellen einer Publish- und Abonnement-Anwendung, die Nachrichten übersetzt](#)
- [Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können](#)
- [Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten](#)
- [Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben](#)

- [Erstellen einer Anwendung, die Kundenfeedback analysiert und Audio generiert](#)
- [Erkennen von Objekten in Bildern mit Amazon Rekognition mithilfe eines AWS -SDK](#)

Erstellen einer Publish- und Abonnement-Anwendung, die Nachrichten übersetzt

AWS SDK for .NET

Zeigt, wie man die .NET-API für Amazon Simple Notification Service verwendet, um eine Webanwendung zu erstellen, die über Abonnement- und Veröffentlichungsfunktionalität verfügt. Darüber hinaus übersetzt diese Beispielanwendung auch Nachrichten.

Vollständiger Quellcode und Anweisungen zum Einrichten und Ausführen finden Sie im vollständigen Beispiel auf [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon SNS
- Amazon Translate

Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können

AWS SDK for .NET

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Vollständiger Quellcode und Anweisungen zum Einrichten und Ausführen finden Sie im vollständigen Beispiel auf [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten

AWS SDK for .NET

Zeigt, wie man die Amazon-DynamoDB-.NET-API verwendet, um eine dynamische Webanwendung zu erstellen, die DynamoDB-Arbeitsdaten verfolgt.

Vollständiger Quellcode und Anweisungen zum Einrichten und Ausführen finden Sie im vollständigen Beispiel auf [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

AWS SDK for .NET

Zeigt, wie Sie mit einer Webanwendung erstellen AWS SDK for .NET , die Arbeitselemente in einer Amazon-Aurora-Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail sendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem RESTful-.NET-Backend zu interagieren.

- Integrieren Sie eine React-Webanwendung mit - AWS Services.
- Auflisten, Hinzufügen, Aktualisieren und Löschen von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem enthaltenen AWS CloudFormation Skript bereit und verwalten Sie sie.

Vollständiger Quellcode und Anweisungen zum Einrichten und Ausführen finden Sie im vollständigen Beispiel auf [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Erstellen einer Anwendung, die Kundenfeedback analysiert und Audio generiert

AWS SDK for .NET

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.
- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Textes eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Bereitstellungsanweisungen finden Sie im Projekt unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Erkennen von Objekten in Bildern mit Amazon Rekognition mithilfe eines AWS -SDK

AWS SDK for .NET

Zeigt, wie Sie die Amazon-Rekognition-.NET-API verwenden, um eine App zu erstellen, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu identifizieren, die sich in einem Bucket von Amazon Simple Storage Service (Amazon S3) befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

Vollständiger Quellcode und Anweisungen zum Einrichten und Ausführen finden Sie im vollständigen Beispiel auf [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES

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 geteilte Verantwortung zwischen 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, die alle in der AWS Cloud angebotenen Services ausführt, und für die Bereitstellung von Services, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei höchster Priorität AWS, und die Effektivität unserer Sicherheit wird regelmäßig von externen Prüfern im Rahmen der [AWS -Compliance-Programme](#) getestet und überprüft.

Sicherheit in der Cloud – Ihre Verantwortung wird durch den AWS Service bestimmt, den Sie verwenden, sowie durch andere Faktoren wie die Vertraulichkeit Ihrer Daten, die Anforderungen Ihrer Organisation und die geltenden Gesetze und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der geteilten Verantwortung](#) durch die spezifischen Amazon Web Services (AWS)-Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [AWS Seite zur Servicesicherheit](#) und [AWS in den Services, die im Rahmen der AWS Compliance-Bemühungen nach Compliance-Programmen enthalten sind](#).

Themen

- [Datenschutz in diesem AWS Produkt oder Service](#)
- [Identitäts- und Zugriffsverwaltung](#)
- [Compliance-Validierung für dieses AWS Produkt oder diesen Service](#)
- [Ausfallsicherheit für dieses AWS Produkt oder diesen Service](#)
- [Infrastruktursicherheit für dieses AWS Produkt oder diesen Service](#)
- [Erzwingen einer TLS-Mindestversion in der AWS SDK for .NET](#)
- [Amazon S3-Verschlüsselungs-Client-Migration](#)

Datenschutz in diesem AWS Produkt oder Service

Das AWS [Modell der geteilten Verantwortung](#)Modell gilt für den Datenschutz in diesem AWS Produkt oder Service. Wie in diesem Modell beschrieben, AWS ist für den Schutz der globalen Infrastruktur verantwortlich, auf der alle ausgeführt werden 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 Ihnen, -Anmeldeinformationen zu schützen AWS-Konto und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einzurichten. 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).
- Verwenden Sie SSL/TLS für die Kommunikation mit - AWS Ressourcen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API- und Benutzeraktivitätsprotokollierung mit ein AWS CloudTrail.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen Standardsicherheitskontrollen in 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 auf AWS über eine Befehlszeilenschnittstelle oder eine API FIPS-140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-2](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit diesem AWS Produkt oder Service oder anderen AWS-Services über die Konsole, API AWS CLI oder AWS SDKs arbeiten. 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) ist ein AWS-Service, mit dem ein Administrator den Zugriff auf AWS-Ressourcen sicher steuern kann. IAM-Administratoren steuern, wer für die Nutzung von AWS-Ressourcen authentifiziert (angemeldet) und autorisiert (im Besitz von Berechtigungen) werden kann. IAM ist ein AWS-Service, den Sie ohne zusätzliche Kosten verwenden können.

Themen

- [Zielgruppe](#)
- [Authentifizierung mit Identitäten](#)
- [Verwalten des Zugriffs mit Richtlinien](#)
- [Funktionsweise AWS-Services von mit IAM](#)
- [Fehlerbehebung für AWS Identität und Zugriff](#)

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, unterscheidet sich je nach Ihrer Arbeit in AWS.

Service-Benutzer – Wenn Sie AWS-Services zur Ausführung von Aufgaben verwenden, stellt Ihnen Ihr Administrator die Anmeldeinformationen und Berechtigungen bereit, die Sie benötigen. Wenn Sie für Ihre Arbeit weitere AWS-Funktionen ausführen, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Wenn Sie nicht auf ein Feature zugreifen können, finden Sie weitere Informationen unter [Fehlerbehebung für AWS Identität und Zugriff](#) oder im Benutzerhandbuch des von AWS-Service Ihnen verwendeten.

Service-Administrator – Wenn Sie in Ihrem Unternehmen für die AWS-Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollständigen Zugriff auf AWS. Ihre Aufgabe besteht darin, zu bestimmen, auf welche AWS-Funktionen und Ressourcen Ihre Service-Benutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachvollziehen zu können. Weitere Informationen darüber, wie Ihr Unternehmen IAM mit verwenden kann, finden Sie im Benutzerhandbuch der AWS-Service, die Sie verwenden.

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 , die AWS-Service Sie verwenden.

Authentifizierung mit Identitäten

Die Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten bei anmelden. Sie müssen als Root-Benutzer des AWS-Kontos, als IAM-Benutzer oder durch Übernahme einer IAM-Rolle authentifiziert (bei angemeldet AWS) sein.

Sie können sich bei AWS als Verbundidentität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt werden. AWS IAM Identity Center (IAM Identity Center)-Benutzer, die Single-Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für Verbundidentitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie AWS über einen Verbund auf zugreifen, übernehmen Sie indirekt eine Rolle.

Je nachdem, um welchen Benutzertyp es sich handelt, können Sie sich bei der AWS Management Console oder im - AWS Zugriffsportal anmelden. Weitere Informationen zur Anmeldung bei AWS finden Sie unter [So melden Sie sich bei Ihrem an AWS-Konto](#) im AWS-Anmeldung - Benutzerhandbuch.

Wenn Sie AWS programmgesteuert auf zugreifen, AWS stellt ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (Command Line Interface, CLI) bereit, um Ihre Anforderungen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie keine AWS Tools verwenden, müssen Sie Anforderungen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode zum eigenständigen Signieren von Anforderungen finden Sie unter [Signieren von AWS API-Anforderungen](#) im IAM-Benutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. empfiehlt beispielsweise, AWS die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter [Multi-Faktor-Authentifizierung](#) im AWS IAM Identity Center - Benutzerhandbuch und [Verwenden der Multi-Faktor-Authentifizierung \(MFA\) in AWS](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie ein erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services und Ressourcen im Konto hat. Diese Identität wird als AWS-Konto

Root-Benutzer bezeichnet und Sie melden sich mit der E-Mail-Adresse und dem Passwort an, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Verbundidentität

Fordern Sie als bewährte Methode menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, auf, den Verbund mit einem Identitätsanbieter zu verwenden, um AWS-Services mithilfe temporärer Anmeldeinformationen auf zuzugreifen.

Eine Verbundidentität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, ein Web-Identitätsanbieter, die AWS Directory Service, das Identity-Center-Verzeichnis oder jeder Benutzer, der mit AWS-Services Anmeldeinformationen auf zugreift, die über eine Identitätsquelle bereitgestellt werden. Wenn Verbundidentitäten auf zugreifen AWS-Konten, übernehmen sie Rollen und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen oder eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und synchronisieren, um sie für alle Ihre AWS-Konten und Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität in Ihrem AWS-Konto mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter [Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Eine [IAM-Gruppe](#) ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche

Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter [Erstellen eines IAM-Benutzers \(anstatt einer Rolle\)](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität in Ihrem AWS-Konto mit bestimmten Berechtigungen. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS Management Console indem Sie die [Rollen wechseln](#). Sie können eine Rolle übernehmen, indem Sie eine AWS CLI - oder AWS -API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter [Verwenden von IAM-Rollen](#) im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- **Verbundbenutzerzugriff** – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter [Erstellen von Rollen für externe Identitätsanbieter](#) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter [Berechtigungssätze](#) im AWS IAM Identity Center -Benutzerhandbuch.
- **Temporäre IAM-Benutzerberechtigungen** – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- **Kontoübergreifender Zugriff** – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können AWS-Services Sie jedoch eine Richtlinie direkt an eine Ressource anfügen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden

Zugriff finden Sie unter [So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien](#) im IAM-Benutzerhandbuch.

- Serviceübergreifender Zugriff – Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon EC2 aus oder speichert Objekte in Amazon S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
- Forward Access Sessions (FAS) – Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen in auszuführen AWS, gelten Sie als Prinzipal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service auslösen. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anforderung AWS-Service , Anfragen an nachgelagerte Services zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Service eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder -Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter [Zugriffssitzungen weiterleiten](#).
- Servicerolle: Eine Servicerolle ist eine [IAM-Rolle](#), die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service](#) im IAM-Benutzerhandbuch.
- Serviceverknüpfte Rolle – Eine serviceverknüpfte Rolle ist eine Art von Servicerolle, die mit einem verknüpft ist AWS-Service. Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Serviceverknüpfte Rollen werden in Ihrem angezeigt AWS-Konto und gehören dem Service. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.
- Anwendungen, die auf Amazon EC2 ausgeführt werden – Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und - AWS CLI oder AWS -API-Anforderungen stellen. Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Um einer EC2-Instance eine - AWS Rolle zuzuweisen und sie für alle ihre Anwendungen verfügbar zu machen, erstellen Sie ein Instance-Profil, das an die Instance angehängt ist. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter [Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon EC2-Instances ausgeführt werden](#) im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter [Erstellen einer IAM-Rolle \(anstatt eines Benutzers\)](#) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie steuern den Zugriff in , AWS indem Sie Richtlinien erstellen und sie an AWS Identitäten oder Ressourcen anfügen. Eine Richtlinie ist ein Objekt in , AWS das, wenn es einer Identität oder Ressource zugeordnet wird, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anforderung stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden in AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Administratoren können AWS JSON-Richtlinien verwenden, um anzugeben, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console, der AWS CLI oder der AWS -API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Erstellen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die

Sie mehreren Benutzern, Gruppen und Rollen in Ihrem anfügen können AWS-Konto. Verwaltete Richtlinien umfassen - AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter [Auswahl zwischen verwalteten und eingebundenen Richtlinien](#) im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#). Prinzipale können Konten, Benutzer, Rollen, Verbundbenutzer oder umfassen AWS-Services.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Services AWS WAF, die ACLs unterstützen. Weitere Informationen zu ACLs finden Sie unter [Zugriffssteuerungsliste \(ACL\) – Übersicht](#) (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger häufig verwendete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- **Berechtigungsgrenzen** – Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze

für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im IAM-Benutzerhandbuch.

- **Service-Kontrollrichtlinien (SCPs)** – SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in angeben AWS Organizations. AWS Organizations ist ein Service zum Gruppieren und zentralen Verwalten mehrerer AWS-Konten, die Ihrem Unternehmen gehören. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. Die SCP beschränkt Berechtigungen für Entitäten in Mitgliedskonten, einschließlich jeder Root-Benutzer des AWS-Kontos. Weitere Informationen zu Organizations und SCPs finden Sie unter [Funktionsweise von SCPs](#) im AWS Organizations -Benutzerhandbuch.
- **Sitzungsrichtlinien** – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. 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. Wie AWS bestimmt, ob eine Anforderung zugelassen werden soll, wenn mehrere Richtlinientypen beteiligt sind, erfahren Sie unter [Logik zur Richtlinienbewertung](#) im IAM-Benutzerhandbuch.

Funktionsweise AWS-Services von mit IAM

Einen Überblick über die AWS-Services Funktionsweise der meisten IAM-Funktionen finden Sie unter [-AWS Services, die mit IAM funktionieren](#) im IAM-Benutzerhandbuch.

Informationen zur Verwendung eines bestimmten AWS-Service mit IAM finden Sie im Abschnitt Sicherheit im Benutzerhandbuch des entsprechenden Services.

Fehlerbehebung für AWS Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die beim Arbeiten mit AWS und IAM auftreten können.

Themen

- [Ich bin nicht autorisiert, eine Aktion in auszuführen AWS](#)
- [Ich bin nicht autorisiert, iam durchzuführen:PassRole](#)
- [Ich möchte Personen außerhalb meines AWS-Konto Zugriff auf meine - AWS Ressourcen gewähren](#)

Ich bin nicht autorisiert, eine Aktion in auszuführen AWS

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer mateojackson versucht, über die Konsole Details zu einer fiktiven *my-example-widget*-Ressource anzuzeigen, jedoch nicht über `aws:GetWidget`-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer mateojackson aktualisiert werden, damit er mit der `aws:GetWidget`-Aktion auf die *my-example-widget*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht autorisiert, iam durchzuführen:PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine vorhandene Rolle an diesen Service zu übergeben, anstatt eine neue Servicerolle oder serviceverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb meines AWS-Konto Zugriff auf meine - AWS Ressourcen gewähren

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. Im Fall von Services, die ressourcenbasierte Richtlinien oder Zugriffssteuerungslisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen dazu, ob diese Funktionen AWS unterstützt, finden Sie unter [Funktionsweise AWS-Services von mit IAM](#).
- Informationen zum Gewähren des Zugriffs auf Ihre AWS-Konten -Ressourcen in Ihrem Besitz finden Sie unter [Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , das Sie besitzen](#) im IAM-Benutzerhandbuch.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre -Ressourcen gewähren AWS-Konten, finden Sie unter [Gewähren von Zugriff auf im AWS-Konten Besitz von Dritten](#) im IAM-Benutzerhandbuch.
- 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 [So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien](#) im IAM-Benutzerhandbuch.

Compliance-Validierung für dieses AWS Produkt oder diesen Service

Informationen darüber, ob ein in den Geltungsbereich bestimmter Compliance-Programme AWS-Service fällt, finden Sie [AWS-Services unter im Geltungsbereich nach Compliance-Programm](#) und wählen Sie das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme](#)

Sie können Auditberichte von Drittanbietern mit herunterladen AWS Artifact. Weitere Informationen finden Sie unter [Herunterladen von Berichten unter AWS Artifact](#) .

Ihre Compliance-Verantwortung bei der Verwendung von AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Unterstützung der Compliance bereit:

- [Schnellstartanleitungen für Sicherheit und Compliance](#) – In diesen Bereitstellungsleitfäden werden Überlegungen zur Architektur erörtert und Schritte für die Bereitstellung von Basisumgebungen in bereitgestellt AWS , die sich auf Sicherheit und Compliance konzentrieren.
- [Architekturerstellung für HIPAA-Sicherheit und -Compliance in Amazon Web Services](#) – In diesem Whitepaper wird beschrieben, wie Unternehmen mithilfe AWS von HIPAA-berechtigte Anwendungen erstellen können.

Note

Nicht alle AWS-Services sind HIPAA-berechtigt. Weitere Informationen finden Sie in der [Referenz für HIPAA-berechtigte Services](#).

- [AWS Compliance-Ressourcen](#) – Diese Sammlung von Arbeitsmappen und Leitfäden könnte für Ihre Branche und Ihren Standort gelten.
- [AWS Kunden-Compliance-Leitfäden](#) – Verstehen Sie das Modell der geteilten Verantwortung anhand der Berücksichtigung der Compliance. Die Leitfäden fassen die bewährten Methoden zur Sicherung zusammen AWS-Services und ordnen die Leitlinien den Sicherheitskontrollen in mehreren Frameworks zu (einschließlich National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Officer (PCI) und International Organization for Standardization (ISO)).

- [Bewertung von Ressourcen mit Regeln](#) im -AWS Config Entwicklerhandbuch – Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#) – Dies AWS-Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus innerhalb von AWS. Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Eine Liste der unterstützten Services und Kontrollen finden Sie in der [Security-Hub-Steuerungsreferenz](#).
- [AWS Audit Manager](#) – Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um den Umgang mit Risiken und die Einhaltung von Branchenstandards zu vereinfachen.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der geteilten Verantwortung](#) über die spezifischen Amazon Web Services (AWS)-Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [AWS Seite zur Servicesicherheit](#) und [AWS in den Services, die im Rahmen von AWS Compliance-Bemühungen nach Compliance-Programmen enthalten sind](#).

Ausfallsicherheit für dieses AWS Produkt oder diesen Service

Die AWS globale -Infrastruktur ist um AWS-Regionen und Availability Zones herum aufgebaut.

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die mit einem Netzwerk mit niedriger Latenz, hohem Durchsatz und hoher Redundanz 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 geteilten Verantwortung](#) durch die spezifischen Amazon Web Services (AWS)-Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [AWS Seite zur Servicesicherheit](#) und [AWS in den Services, die im Rahmen von AWS Compliance-Bemühungen nach Compliance-Programmen enthalten sind](#).

Infrastruktursicherheit für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Service verwendet verwaltete Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsservices und wie die Infrastruktur AWS schützt, finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS Umgebung mit den bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastrukturschutz](#) in Security Pillar AWS Well-Architected Framework.

Sie verwenden durch 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 geteilten Verantwortung](#) über die spezifischen Amazon Web Services (AWS)-Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [AWS Seite mit der Service-Sicherheitsdokumentation und AWS den Services, die im Rahmen von AWS Compliance-Bemühungen nach Compliance-Programmen enthalten sind](#).

Erzwingen einer TLS-Mindestversion in der AWS SDK for .NET

Um die Sicherheit bei der Kommunikation mit - AWS Services AWS SDK for .NET zu erhöhen, sollten Sie die für die Verwendung von TLS 1.2 oder höher konfigurieren.

Die AWS SDK for .NET verwendet die zugrunde liegende .NET-Laufzeit, um zu bestimmen, 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 for .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 unter Windows 7 SP1 oder Windows Server 2008 R2 SP1 ausführen, müssen Sie sicherstellen, 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 v10.9 oder höher](#) unterstützt. .NET Core Version 2.1 und höher erfordert neuere Versionen von macOS, wie unter <https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos> beschrieben.

Bei Verwendung von .NET Core 1.0 nutzt .NET Core [OpenSSL unter macOS](#) eine Abhängigkeit, die separat installiert werden muss. OpenSSL hat Unterstützung für TLS 1.2 in Version 1.0.1 hinzugefügt und Unterstützung für TLS 1.3 in Version 1.1.1 hinzugefügt.

Linux

.NET Core ist unter Linux OpenSSL erforderlich, das mit vielen Linux-Verteilungen geliefert wird. Aber es kann auch separat installiert werden. OpenSSL hat Unterstützung für TLS 1.2 in Version 1.0.1 hinzugefügt und Unterstützung für TLS 1.3 in Version 1.1.1 hinzugefügt. 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-Laufzeit verwenden, die die Betriebssystemeinstellungen (.NET Framework 3.5 bis 4.5.2) nicht verwendet, AWS SDK for .NET versucht die , den unterstützten Protokollen [Unterstützung für TLS 1.1 und TLS 1.2 hinzuzufügen](#). 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 – [KB3154520](#)
- Windows Server 2012 – [KB3154519](#)
- Windows 7 SP1 und Server 2008 R2 SP1 – [KB3154518](#)

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

Wenn Ihre Anwendung auf einem neueren .NET Framework unter Windows 7 SP1 oder Windows Server 2008 R2 SP1 ausgeführt wird, müssen Sie sicherstellen, 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](#).

Ausführliche 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 for PowerShell

[AWS Tools for PowerShell](#) verwenden die AWS SDK for .NET für alle Aufrufe von - AWS Services. Das Verhalten Ihrer Umgebung hängt wie folgt von der Windows-Version ab, die PowerShell Sie ausführen.

Windows PowerShell 2.0 bis 5.x

Windows PowerShell 2.0 bis 5.x werden unter .NET Framework ausgeführt. Mit dem PowerShell folgenden Befehl können Sie überprüfen, welche .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 for .NET und .NET Framework 3.5.

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

- Wenn Sie .NET Runtime 4.0 verwenden, befolgen Sie die obigen Anweisungen in Bezug auf AWS SDK for .NET und .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 und neuer werden auf .NET Core ausgeführt. Mit dem folgenden Befehl können Sie überprüfen, welche Version von .NET Core verwendet wird.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.TargetFrameworkAttribute], $true).FrameworkName
```

Folgen Sie den Anweisungen oben in Bezug auf die AWS SDK for .NET und die entsprechende Version von .NET Core.

Xamarin

Informationen zu Xamarin finden Sie in den 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 : HttpClient Die Implementierung muss auf Android und die SSL/TLS-Implementierung auf Native TLS 1.2+ festgelegt sein.

Für iOS

- iOS 7 oder höher erforderlich.
- Projekteigenschaften , iOS Build : HttpClient Die Implementierung muss auf NSURLSession gesetzt sein.

Für macOS

- macOS 10.9 oder höher erforderlich.
- Projektoptionen , Build , Mac Build : HttpClient implementation müssen auf NSURLSession gesetzt sein.

Unity

Sie müssen Unity 2018.2 oder höher verwenden und die .NET 4.x gleichwertige Skripting-Laufzeit verwenden. Sie können dies in Projekteinstellungen , Konfiguration , Player festlegen, wie unter <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html> beschrieben. Die .NET 4.x gleichwertige Skripting-Laufzeit ermöglicht die TLS 1.2-Unterstützung für alle Unity-Plattformen, auf denen Mono oder IL2CPP ausgeführt wird. Weitere Informationen finden Sie unter <https://blog.unity.com/technology/scripting-runtime-improvements-in-unity-2018-2>.

Browser (für Blazor WebAssembly)

WebAssembly wird im Browser statt auf dem Server ausgeführt und verwendet den Browser für die Verarbeitung von HTTP-Datenverkehr. Daher wird die TLS-Unterstützung durch die Browser-Unterstützung bestimmt.

Blazor wird WebAssembly in der Vorschau für ASP.NET Core 3.1 nur in Browsern unterstützt, die unterstützen WebAssembly, wie unter <https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms> beschrieben. Alle Mainstream-Browser unterstützen TLS 1.2, bevor sie unterstützen 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.

Amazon S3-Verschlüsselungs-Client-Migration

In diesem Thema wird gezeigt, wie Sie Ihre Anwendungen von Version 1 (V1) des Amazon Simple Storage Service (Amazon S3)-Verschlüsselungsclients zu Version 2 (V2) migrieren und die Anwendungsverfügbarkeit während des gesamten Migrationsprozesses sicherstellen.

Objekte, die mit dem V2-Client verschlüsselt sind, können nicht mit dem V1-Client entschlüsselt werden. Um die Migration zum neuen Client zu erleichtern, ohne alle Objekte gleichzeitig erneut verschlüsseln zu müssen, wurde ein „V1-transitionaler“ Client bereitgestellt. Dieser Client kann sowohl V1- als auch V2-encrypted Objekte entschlüsseln, aber Objekte nur im V1-compatible Format. Der V2-Client kann sowohl V1- als auch V2-encrypted Objekte entschlüsseln (wenn er für V1-Objekte aktiviert ist), aber Objekte nur im V2-compatible Format verschlüsseln.

Migrationsübersicht

Diese Migration erfolgt in drei Phasen. Diese Phasen werden hier eingeführt und später ausführlich beschrieben. Jede Phase muss für all eClients abgeschlossen sein, die freigegebene Objekte verwenden, bevor die nächste Phase gestartet wird.

1. Aktualisieren Sie vorhandene Clients auf V1-transitional-Clients, um neue Formate zu lesen. Aktualisieren Sie zunächst Ihre Anwendungen so, dass sie vom V1-transitional-Client und nicht vom V1-Client abhängen. Der V1-transitional-Client ermöglicht es Ihrem vorhandenen Code, Objekte zu entschlüsseln, die von den neuen V2-Clients geschrieben wurden, und Objekte, die im V1-compatible Format geschrieben wurden.

Note

Der V1-transitional-Client wird nur für Migrationszwecke bereitgestellt. Fahren Sie mit dem Upgrade auf den V2-Client fort, nachdem Sie zum V1-transitionalClient gewechselt sind.

2. Migrieren Sie V1-transitional eClients zu V2-Clients, um neue Formate zu schreiben. Ersetzen Sie als Nächstes alle V1-transitional-Clients in Ihren Anwendungen durch V2-Clients und setzen Sie das Sicherheitsprofil auf V2AndLegacy. Durch das Festlegen dieses Sicherheitsprofils auf V2-Clients können diese Clients Objekte entschlüsseln, die im V1-compatible Format verschlüsselt wurden.
3. Aktualisieren Sie V2-Clients, um keine V1-Formate mehr zu lesen. Nachdem alle Clients zu V2 migriert wurden und alle Objekte im V2-compatible Format verschlüsselt oder erneut

verschlüsselt wurden, legen Sie das V2-Sicherheitsprofil V2 auf statt auf `festV2AndLegacy`. Dadurch wird die Entschlüsselung von Objekten im V1-compatible Format verhindert.

Aktualisieren vorhandener Clients auf V1-transitional-Clients zum Lesen neuer Formate

Der V2-Verschlüsselungsclient verwendet Verschlüsselungsalgorithmen, die ältere Versionen des Clients nicht unterstützen. Der erste Schritt bei der Migration besteht darin, Ihre V1-Entschlüsselungsclients zu aktualisieren, damit sie das neue Format lesen können.

Der V1-transitional-Client ermöglicht es Ihren Anwendungen, sowohl V1- als auch V2-encrypted Objekte zu entschlüsseln. Dieser Client ist Teil des [Amazon.Extensions.S3.Encryption](#) NuGet -Pakets. Führen Sie für jede Ihrer Anwendungen die folgenden Schritte aus, um den V1-transitionalClient zu verwenden.

1. Nehmen Sie eine neue Abhängigkeit vom [Amazon.Extensions.S3.Encryption](#)-Paket an. Wenn Ihr Projekt direkt von den `AWSSDK.S3`- oder `AWSSDK.KeyManagementService`-Paketen abhängt, müssen Sie diese Abhängigkeiten entweder aktualisieren oder entfernen, damit ihre aktualisierten Versionen mit diesem neuen Paket abgerufen werden.
2. Ändern Sie die entsprechende `using` Anweisung `Amazon.Extensions.S3.Encryption` wie folgt von `Amazon.S3.Encryption` in :

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. Erstellen Sie Ihre Anwendung neu und stellen Sie sie erneut bereit.

Der V1-transitional-Client ist vollständig API-kompatibel mit dem V1-Client, sodass keine anderen Codeänderungen erforderlich sind.

Migrieren von V1-transitional-Clients zu V2-Clients zum Schreiben neuer Formate

Der V2-Client ist Teil des [Amazon.Extensions.S3.Encryption](#) NuGet -Pakets. Sie ermöglicht es Ihren Anwendungen, sowohl V1- als auch V2-encrypted Objekte zu entschlüsseln (sofern konfiguriert), aber Objekte nur im V2-compatible Format zu verschlüsseln.

Nachdem Sie Ihre vorhandenen Clients aktualisiert haben, um das neue Verschlüsselungsformat zu lesen, können Sie Ihre Anwendungen sicher auf die V2-Verschlüsselungs- und Entschlüsselungsclients 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 verwendete Verschlüsselungsmethode, d. h. `KmsType.KmsContext`.
 - iii. Stellen Sie KMS einen Verschlüsselungskontext bereit, der diesem Datenschlüssel zugeordnet werden soll. Sie können ein leeres Wörterbuch senden (Amazon-Verschlüsselungskontext wird immer noch zusammengeführt), aber die Bereitstellung von zusätzlichem Kontext wird empfohlen.
 - b. Bei Verwendung von vom Benutzer bereitgestellten Schlüsselumbruchmethoden (symmetrische oder asymmetrische Verschlüsselung):
 - i. Stellen Sie eine - AES oder eine RSA -Instance bereit, die die Verschlüsselungsmaterialien enthält.
 - ii. Deklarieren Sie, welcher Verschlüsselungsalgorithmus verwendet werden soll, d. h. `SymmetricAlgorithmType.AesGcm` oder `AsymmetricAlgorithmType.RsaOaepSha1`.
2. Ändern Sie `AmazonS3CryptoConfiguration` in `AmazonS3CryptoConfigurationV2` wobei die Eigenschaft auf `SecurityProfile` gesetzt ist `SecurityProfile.V2AndLegacy`.
3. Ändern Sie `AmazonS3EncryptionClient` zu `AmazonS3EncryptionClientV2`. Dieser Client übernimmt die neu konvertierten `EncryptionMaterialsV2` Objekte `AmazonS3CryptoConfigurationV2` und aus den vorherigen Schritten.

Beispiel: KMS zu KMS+Context

Vormigration

```
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.

Vormigration

```
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 beim Entschlüsseln 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 zu RSA-OAEP-SHA1 Key Wrap)

StorageMode kann ObjectMetadata oder InstructionFile sein.

Vormigration

```
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 von V2-Clients auf V1-Formate, die nicht länger gelesen werden

Schließlich wurden alle Objekte mit einem V2-Client verschlüsselt oder erneut verschlüsselt. Nachdem diese Konvertierung abgeschlossen ist, können Sie die V1-Kompatibilität in den V2-Clients deaktivieren, indem Sie die `-SecurityProfile`Eigenschaft auf `setzenSecurityProfile.V2`, wie im folgenden Ausschnitt gezeigt.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Besondere Überlegungen für AWS SDK for .NET

Dieser Abschnitt enthält Überlegungen für Sonderfälle, in denen die normalen Konfigurationen oder Prozeduren nicht angemessen oder ausreichend sind.

Themen

- [Abrufen von Baugruppen für das AWS SDK for .NET](#)
- [Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung](#)
- [Besondere Überlegungen zur Unity-Unterstützung](#)
- [Besondere Überlegungen zur Xamarin Unterstützung](#)

Abrufen von Baugruppen für das AWS SDK for .NET

In diesem Thema wird beschrieben, wie Sie die AWSSDK Baugruppen abrufen und lokal (oder On-Premises) für die Verwendung in Ihren Projekten speichern können. Dies ist nicht die empfohlene Methode für den Umgang mit SDK-Referenzen, ist aber in einigen Umgebungen erforderlich.

Note

Die empfohlene Methode für die Handhabung von 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.

Zip-Dateien herunterladen und extrahieren

(Beachten Sie, dass dies nicht die [empfohlene Methode](#) für den Umgang mit Verweisen auf die ist AWS SDK for .NET.)

1. Laden Sie eine der folgenden ZIP-Dateien herunter:
 - [aws-sdk-net8.0.zip](#) – Baugruppen, die .NET 8 und höher unterstützen.

- [aws-sdk-netcoreapp3.1.zip](#) – Baugruppen, die .NET Core 3.1 und höher unterstützen.
- [aws-sdk-netstandard2.0.zip](#) – Baugruppen, die .NET Standard 2.0 und 2.1 unterstützen.
- [aws-sdk-net45.zip](#) – Baugruppen, die .NET Framework 4.5 und höher unterstützen.
- [aws-sdk-net35.zip](#) – Baugruppen, die .NET Framework 3.5 unterstützen.

Warning

Ab dem 15. August 2024 AWS SDK for .NET wird die Unterstützung für .NET Framework 3.5 beenden und die minimale .NET Framework-Version auf 4.6.2 ändern. Weitere Informationen finden Sie im Blogbeitrag [Wichtige Änderungen für die Ziele von .NET Framework 3.5 und 4.5 des AWS SDK for .NET](#).

2. Extrahieren Sie die Baugruppen in einen „Herunterladen“-Ordner in Ihrem Dateisystem; es spielt keine Rolle, wo sie gespeichert sind. Notieren Sie sich diesen Ordner.
3. Wenn Sie Ihr Projekt einrichten, erhalten Sie die erforderlichen Komponenten aus diesem Ordner, wie unter beschrieben [Installieren Sie AWSSDK-Baugruppen ohne NuGet](#).

Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung

Die bevorzugte Methode für die Verwendung von Anmeldeinformationen besteht darin, es den Benutzern AWS SDK for .NET zu ermöglichen, sie für Sie zu finden und abzurufen, wie unter beschrieben [Auflösung von Anmeldeinformationen und Profilen](#).

Sie können Ihre Anwendung jedoch auch so konfigurieren, dass sie aktiv Profile und Anmeldeinformationen abrufen 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](#). `CredentialManagement` Namespace.

- Verwenden Sie die [SharedCredentialsFile](#) Klasse, um ein Profil in einer Datei zu finden, die AWS das Dateiformat für [AWSAnmeldeinformationen 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 [CredentialsFileNetSDK-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 [CredentialProfileStoreChain](#) SDK-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 [CredentialProfileStoreChain](#) Klasse abrufen, indem Sie die [TryGetProfile](#) Methoden [TryGetAWSCredentials](#) oder 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 gemeinsamen AWS Anmeldeinformationen am Standardspeicherort stammen.

In diesem Beispiel wird auch [Amazon.Runtime verwendet. AWSCredentials](#) Klasse.

```
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 [CredentialProfile](#) Klasse 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. AWSCredentials](#) Klasse.

```
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 AWSCredentialsFactory

Erstellen Sie einen AmazonS3-Client 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 [CredentialProfile](#)Klasse und die [Amazon.Runtime verwendet. AWSCredentials](#)Klasse.

```
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 [CredentialsFileNetSDK-Klasse](#) kann auf genau die gleiche Weise verwendet werden, außer dass Sie ein neues CredentialsFile NetSDK-Objekt anstelle eines Objekts instanziiieren würden. SharedCredentialsFile

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 for .NET Assemblys (DLL-Dateien) verweisen, anstatt sie zu verwenden NuGet. AWS SDK for .NET Angesichts dieser Anforderung müssen Sie die folgenden wichtigen Aktionen ausführen.

- Sie müssen sich die AWS SDK for .NET Baugruppen besorgen und sie auf Ihr Projekt anwenden. Informationen dazu, wie Sie dies tun können, finden Sie [Zip-Dateien herunterladen und extrahieren](#) im Thema [Abrufen von AWSSDK Baugruppen](#).
- Sie müssen neben den DLLs für AWSSDK.Core und die anderen AWS Dienste, die Sie verwenden, die folgenden DLLs in Ihr Unity-Projekt aufnehmen. Ab Version 3.5.109 von enthält die AWS SDK for .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 Datei hinzufügen, um Code-Stripping zu verhindern. `link.xml` Die `link.xml` Datei muss alle AWSSDK Assemblys auflisten, die Sie verwenden, und jede muss das Attribut enthalten. `preserve="all"` 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-from-unity-xamarin-or-uwp/>.

Zusätzlich zu diesen speziellen Überlegungen finden Sie unter Informationen [Was hat sich für Version 3.5 geändert](#) zur Migration Ihrer Unity-Anwendung auf Version 3.5 von. AWS SDK for .NET

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](#).

Siehe auch die Informationen über [Portable Class Library und Xamarin](#) aus.

API-Referenz für den AWS SDK for .NET

Die AWS SDK for .NET bietet eine API für den Zugriff auf AWS-Services. Um zu sehen, welche Klassen und Methoden in der API verfügbar sind, lesen Sie die [AWS SDK for .NET-API-Referenz](#) aus.

Zusätzlich zu der oben genannten allgemeinen Referenz wird jedes der Beispiele unter der [Codebeispiele mit Anleitungen](#) enthält Verweise auf die spezifischen Methoden und Klassen, die in diesem Beispiel verwendet werden.

Dokumentverlauf

In der folgenden Tabelle werden die wichtigen Änderungen seit der letzten Version des AWS SDK for .NET Entwicklerhandbuchs beschrieben. Für Benachrichtigungen über Aktualisierungen dieser Dokumentation können Sie einen [RSS-Feed](#) abonnieren.

Änderung	Beschreibung	Datum
Was ist neu?	Enthält Informationen zur Unterstützung für .NET 8.	23. Februar 2024
Was ist neu?	Enthält Informationen zu bevorstehenden Änderungen am .NET Framework-Support.	18. Februar 2024
Abrufen von AWSSDK Baugruppen	Enthält Informationen zu Baugruppen, die .NET 8 und höher unterstützen.	8. Januar 2024
AWS Framework zur Nachricht enverarbeitung für .NET	Es wurden Informationen über die Vorschauversion des Message Processing Framework enthalten.	10. Dezember 2023
AWS OpsWorks	Hinweis zum Ende der Lebensdauer für hinzugefügt AWS OpsWorks.	8. Dezember 2023
Verwenden von Amazon-DynamoDB-NoSQL-Datenbanken	Aktualisierte Informationen zu den Modellen für die Protokoll- und Objektpersistenz-Programmierung. Es ist jetzt möglich, bestimmte Latenz- oder Deadlock-Bedingungen aufgrund von Kaltstart- und Thread-Pool-Verhalten zu verhindern.	15. November 2023

Weitere Updates zu bewährten Methoden für IAM enthalten	Aktualisierter Leitfaden, angepasst an die bewährten IAM-Methoden. Weitere Informationen finden Sie unter Bewährte IAM-Methoden .	05. Oktober 2023
Abrufen von AWSSDK Baugruppen	Informationen zur Installation der wurden AWS SDK for .NET mithilfe des AWS Tools for Windows Installationsprogramms (d. h. des MSI) entfernt, das veraltet ist.	25. September 2023
Aktualisierungen der bewährten Methoden für IAM	Aktualisierter Leitfaden, angepasst an die bewährten IAM-Methoden. Weitere Informationen finden Sie unter Bewährte IAM-Methoden .	18. Juli 2023
Lambda-Anmerkungen	Das AWS Lambda Annotations-Framework wurde für die allgemeine Verfügbarkeit veröffentlicht.	17. Juli 2023
Was ist neu?	Es wurden Informationen zur Vorschauversion des Distributed Cache Provider für DynamoDB hinzugefügt.	15. Juli 2023
Inhaltsverzeichnis	Das Inhaltsverzeichnis wurde aktualisiert, um Codebeispiele leichter auffindbar zu machen.	08. Juni 2023
Auflösung der Region	Es wurden Informationen darüber hinzugefügt, wie das SDK eine fehlende Regionsspezifikation auflöst.	14. März 2023

Unterstützung für den MSI	Hinweis zum Beenden des Supports für das AWS Tools for Windows Installationsprogramm hinzugefügt.	6. März 2023
Lambda-Anmerkungen (Vorschau)	Vorschau des AWS Lambda Annotations-Frameworks.	22. September 2022
Bereitstellen von Anwendungen in AWS	Hauptinhalte wurden auf eine GitHub Pages-Website verschoben: https://aws.github.io/aws-dotnet-deploy/	28. Juni 2022
Außerbetriebnahme von EC2-Classic	Hinweise zur Außerbetriebnahme von EC2-Classic hinzugefügt.	13. April 2022
Single Sign-On mit der AWS SDK for .NET	Informationen zu Single Sign-On (SSO) bei Verwendung der hinzugefügt AWS SDK for .NET.	17. März 2022
Erzwingen einer TLS-Mindestversion	Es wurden Informationen zu TLS 1.3 hinzugefügt.	16. März 2022
Arbeiten mit - AWS Services	Enthaltene Listen der Codebeispiele, die auf verfügbar sind GitHub.	28. Februar 2022
Aktivieren von SDK-Metriken	Informationen zum Aktivieren von SDK-Metriken, die veraltet sind, wurden entfernt.	20. Januar 2022
Bereitstellen von Anwendungen in AWS	Es wurde ein Verweis auf das AWS Toolkit for Visual Studio hinzugefügt, das Bereitstellungs-funktionen bietet, die dem AWS Deploy Tool ähneln.	26. Oktober 2021

AWS SDK for .NET Konsolidierung des -Leitfadens für Version 3	Die beiden Entwicklerhandbücher der AWS SDK for .NET Version 3, „V3“ und „neueste“, wurden in einem Leitfaden unter der „v3“-URL zusammengefasst.	18. August 2021
Migrieren von .NET Standard 1.3	Die Unterstützung für .NET Standard 1.3 in der AWS SDK for .NET hat das Ende ihrer Lebensdauer erreicht.	25. März 2021
Bereitstellen von Anwendungen in AWS (Vorschau)	Vorschauinformationen zum AWS Deploy Tool hinzugefügt, mit denen Sie eine Anwendung über die .NET CLI bereitstellen können.	15. März 2021
Version 3.5 des AWS SDK for .NET	Version 3.5 des AWS SDK for .NET wurde veröffentlicht.	25. August 2020
Paginatoren	Paginatoren wurden zu vielen Service-Clients hinzugefügt, was die Paginierung von API-Ergebnissen bequemer macht.	24. August 2020
Wiederholungen und Timeouts	Es wurden Informationen zu Wiederholungsmodi hinzugefügt.	20. August 2020
Migration des S3-Verschlüsselungsclients	Es wurden Informationen zur Migration Ihrer Amazon S3-Verschlüsselungsclients von V1 zu V2 hinzugefügt.	7. August 2020

Verwenden von KMS-Schlüsseln für die S3-Verschlüsselung	Aktualisiertes Beispiel für die Verwendung von Version 2 des S3-Verschlüsselungsclients.	6. August 2020
Migrieren von .NET Standard 1.3	Informationen zur Beendigung der Unterstützung für .NET-Standard 1.3 Ende 2020 wurden hinzugefügt.	18. Mai 2020
Schnellstart	Es wurde ein Quick Start-Abschnitt mit grundlegenden Einstellungen und Tutorials hinzugefügt, um dem Leser das AWS SDK for .NET vorzustellen.	27. März 2020
Erzwingen von TLS 1.2	Es wurden Informationen zum Erzwingen von TLS 1.2 im SDK hinzugefügt.	10. März 2020

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.