

Guida per gli sviluppatori

AWS SDK for .NET



AWS SDK for .NET: Guida per gli sviluppatori

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e il trade dress di Amazon non possono essere utilizzati in relazione ad alcun prodotto o servizio che non sia di Amazon, in alcun modo che possa causare confusione tra i clienti, né in alcun modo che possa denigrare o screditare Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Che cos'è il AWS SDK for .NET	1
Informazioni su questa versione	1
Manutenzione e supporto per le versioni principali dell'SDK	2
Casi di utilizzo comune	2
Argomenti aggiuntivi in questa sezione	2
correlatiAWSutensili	3
Strumenti per Windows PowerShell e Tools for PowerShell Core	3
Toolkit for VS Code	3
Toolkit for Visual Studio	3
Toolkit for Azure DevOps	4
SDK e strumenti di riferimento per	4
Risorse aggiuntive	4
Inizia a usare	7
Installa e configura la tua toolchain	7
Sviluppo multiplatforma	8
Windows con Visual Studio e .NET Core	8
Approfondimenti	9
Configura l'autenticazione SDK	9
Abilitazione e configurazione di IAM Identity Center	9
Configura l'SDK per utilizzare IAM Identity Center.	9
Avvio di una sessione del portale di accesso AWS	11
Informazioni aggiuntive	11
Fai un breve tour	12
Semplice app multiplatforma	12
Semplice app basata su Windows	18
Fasi successive	24
Iniziare un nuovo progetto	24
Configura la AWS regione	25
Crea un client di servizio con una regione particolare	26
Specificare una regione per tutti i client di servizio	27
Risoluzione della regione	28
Informazioni speciali sulla regione Cina (Pechino)	28
Informazioni speciali sui nuovi servizi AWS	29
Installa AWSSDK pacchetti con NuGet	29

Utilizzo NuGet dal prompt dei comandi o dal terminale	30
Utilizzo NuGet da Visual Studio Solution Explorer	30
Utilizzo NuGet dalla console di Package Manager	31
Installa gli assembly AWSSDK senza NuGet	32
Risoluzione di credenziali e profili	33
Risoluzione del profilo	34
Utilizzo delle credenziali dell'account utente federato	35
Specificare ruoli o credenziali temporanee	35
Utilizzo di credenziali proxy	36
Utenti e ruoli	36
Utenti e set di autorizzazioni	36
●Ruoli di servizio	37
Configurazione avanzata	38
AWSSDK.extensions.netcore.setup e IConfiguration	38
Configurazione di altri parametri dell'applicazione	43
Riferimento per i file di configurazione per AWS SDK for .NET	51
Uso delle credenziali legacy	63
Avvertenze e linee guida importanti per le credenziali	64
Utilizzo del file di AWS credenziali condivise	65
Utilizzo dell'SDK Store (solo Windows)	68
Funzionalità SDK	73
API asincrone	73
Tentativi e timeout	75
Tentativi	75
Timeout	77
Esempio	78
Impaginatori	78
Dove posso trovare gli impaginatori?	79
Cosa mi danno gli impaginatori?	79
Impaginazione sincrona e asincrona	79
Esempio	80
Considerazioni aggiuntive per gli impaginatori	83
Strumenti aggiuntivi	84
AWSStrumento di distribuzione	84
AWSMessage Processing Framework per.NET	85
Autenticazione avanzata	86

Autenticazione unica	86
Prerequisiti	87
Configurazione di un profilo SSO	87
Generazione e utilizzo di token SSO	89
Risorse aggiuntive	94
Tutorial	94
Tutorial: solo applicazione.NET	94
Tutorial: AWS CLI e applicazione.NET	103
Distribuzione su AWS	113
Esegui la distribuzione da.NET CLI	113
Esegui la distribuzione dai toolkit IDE	113
Casi d'uso	114
App ASP.NET Core	114
App.NET Console	115
App Blazor WebAssembly	116
Progetti AWS Lambda	116
Prerequisiti	117
Comandi Lambda disponibili	117
Passaggi per implementare	118
Migra il tuo progetto	120
Cosa c'è di nuovo	120
Piattaforme supportate	122
.NET Core	122
.NET Standard 2.0	122
.NET Framework 4.5	122
.NET Framework 3.5	122
Portable Class Library e Xamarin	123
Supporto Unity	123
Ulteriori informazioni	123
Migrazione alla versione 3	123
Informazioni sulle versioni di AWS SDK for .NET	124
Riprogettazione dell'architettura per l'SDK	124
Modifiche importanti	124
Migaglia alla versione 3.5	126
Cosa è cambiato per la versione 3.5	126
Migrazione del codice sincrono	128

Migrazione alla versione 3.7	129
Migrazione da .NET Standard 1.3	129
Lavora con AWS i servizi	130
Esempi di codice con indicazioni	130
AWS CloudFormation	131
Amazon Cognito	135
DynamoDB	143
Amazon EC2	173
IAM	235
Amazon S3	255
Amazon SNS	265
Amazon SQS	269
AWS Lambda	302
API	302
Prerequisiti	302
Argomenti	303
Annotazioni Lambda	303
Librerie e framework di alto livello	304
Framework di elaborazione dei messaggi	305
AWS OpsWorks	326
API	327
Prerequisiti	327
Altri servizi e configurazione	327
Esempi di codice	328
Azioni e scenari	328
ACM	330
Aurora	335
Auto Scaling	377
Amazon Bedrock	458
Runtime di Amazon Bedrock	462
AWS CloudFormation	512
CloudWatch	515
CloudWatch Registri	570
Provider di identità Amazon Cognito	584
Amazon Comprehend	609
DynamoDB	620

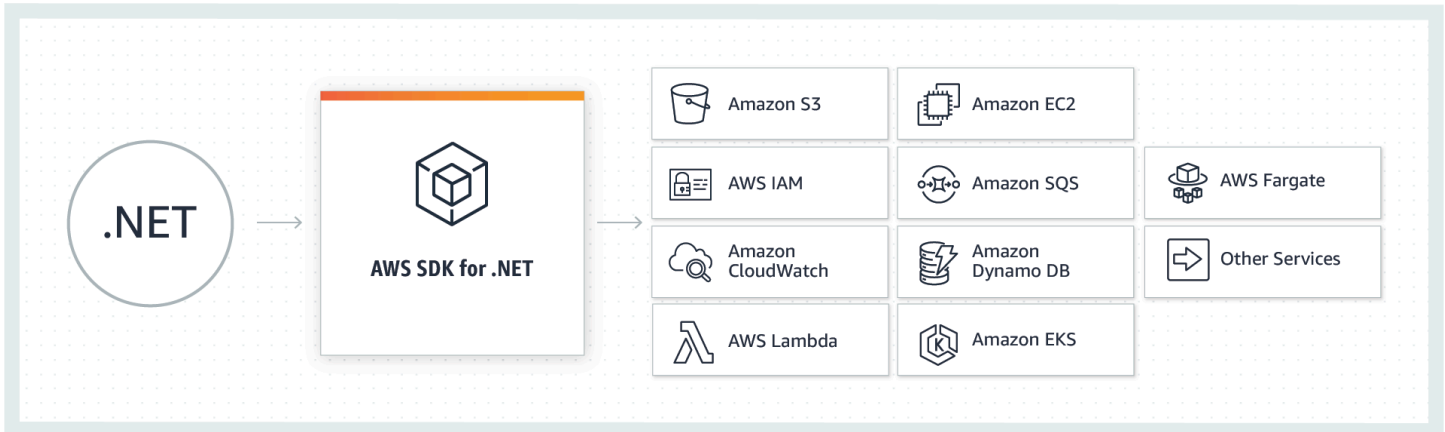
Amazon EC2	714
Amazon ECS	813
Elastic Load Balancing - Versione 2	826
EventBridge	879
AWS Glue	920
IAM	951
Amazon Keyspaces	1075
Kinesis	1102
AWS KMS	1120
Lambda	1132
MediaConvert	1173
Organizations	1184
Amazon Pinpoint	1202
Amazon Polly	1209
Amazon RDS	1220
Amazon Rekognition	1255
Registrazione del dominio Route 53	1285
Amazon S3	1312
S3 Glacier	1440
SageMaker	1450
Secrets Manager	1485
Amazon SES	1488
API Amazon SES v2	1500
Amazon SNS	1539
Amazon SQS	1584
Step Functions	1627
AWS STS	1655
AWS Support	1657
Amazon Transcribe	1684
Amazon Translate	1696
Esempi di servizi incrociati	1708
Costruzione di un'applicazione Amazon SNS	1708
Creazione di un'applicazione serverless per gestire foto	1709
Creazione di un'applicazione Web per tracciare i dati DynamoDB	1709
Creazione di un tracciatore di elementi di lavoro di Aurora Serverless	1710
Crea un'applicazione per analizzare il feedback dei clienti	1710

Rilevamento di oggetti nelle immagini	1711
Trasforma i dati con S3 Object Lambda	1711
Usa il framework di elaborazione dei AWS messaggi per.NET con Amazon SQS	1712
Sicurezza	1713
Protezione dei dati	1713
Identity and Access Management	1714
Destinatari	1715
Autenticazione con identità	1716
Gestione dell'accesso con policy	1719
Come Servizi AWS lavorare con IAM	1722
Risoluzione dei problemi di AWS identità e accesso	1722
Convalida della conformità	1724
Resilienza	1725
Sicurezza dell'infrastruttura	1726
Applicazione di una versione minima di TLS	1726
.NET Core	1727
.NET Framework	1728
AWS Tools for PowerShell	1729
Xamarin	1730
Unità	1730
Browser (per Blazor) WebAssembly	1730
Migrazione del client di crittografia S3	1731
Panoramica sulla migrazione	1731
Aggiorna i client esistenti ai client di transizione V1 per leggere nuovi formati	1732
Esegui la migrazione dei client di transizione V1 ai client V2 per scrivere nuovi formati	1733
Aggiorna i client V2 in modo che non leggano più i formati V1	1736
Considerazioni speciali	1737
Ottenere assieme AWSSDK	1737
Scarica ed estrai i file ZIP	1737
Accesso a credenziali e profili in un'applicazione	1738
Esempi di classe CredentialProfileStoreChain	1739
Esempi di classi SharedCredentialsFile e AWSCredentialsFactory	1740
Supporto Unity	1741
Supporto Xamarin	1742
Documentazione di riferimento dell'API	1743
Cronologia dei documenti	1744

..... **mdccclix**

Che cos'è il AWS SDK for .NET

AWS SDK for .NET Semplifica la creazione di applicazioni.NET che attingono a AWS servizi convenienti, scalabili e affidabili come Amazon Simple Storage Service (Amazon S3) e Amazon Elastic Compute Cloud (Amazon EC2). L'SDK semplifica l'uso dei AWS servizi fornendo un set di librerie coerenti e familiari agli sviluppatori.NET.



(OK, capito! Sono [pronto per prepararmi](#) e [fare un breve tour.](#))

Informazioni su questa versione

Note

Questa documentazione si riferisce alla versione 3.0 e successive di AWS SDK for .NET. È principalmente incentrata su .NET Core e ASP.NET Core, ma contiene anche informazioni su .NET Framework e ASP.NET 4. x. Oltre a Windows e Visual Studio, dà pari considerazione allo sviluppo multiplatforma.

Per informazioni sulla migrazione, consulta [Migra il tuo progetto](#)

Per trovare contenuti obsoleti per le versioni precedenti di AWS SDK for .NET, consulta i seguenti elementi:

- [AWS SDK for .NET \(versione 2, obsoleta\) Guida per gli sviluppatori](#)

Manutenzione e supporto per le versioni principali dell'SDK

Per informazioni sulla manutenzione e sul supporto per le versioni principali dell'SDK e le relative dipendenze sottostanti, consulta quanto segue nella [Guida di riferimento degli strumenti e degli SDK AWS](#):

- [AWS Politica di manutenzione degli SDK e degli strumenti](#)
- [AWS Matrice di supporto delle versioni degli SDK e degli strumenti](#)

Casi di utilizzo comune

Ti AWS SDK for .NET aiuta a realizzare diversi casi d'uso interessanti, tra cui:

- Gestisci utenti e ruoli con [AWS Identity and Access Management \(IAM\)](#).
- Accedi ad [Amazon Simple Storage Service \(Amazon S3\) Simple Storage Service \(Amazon S3\)](#) per creare bucket e archiviare oggetti.
- Gestisci gli abbonamenti HTTP di [Amazon Simple Notification Service \(Amazon SNS\)](#) agli argomenti.
- Usa l'[utilità di trasferimento S3](#) per trasferire file su Amazon S3 dalle tue applicazioni Xamarin.
- Usa [Amazon Simple Queue Service \(Amazon SQS\)](#) per elaborare messaggi e flussi di lavoro tra i componenti di un sistema.
- Esegui trasferimenti Amazon S3 efficienti inviando istruzioni SQL ad [Amazon S3 Select](#).
- [Crea e avvia istanze Amazon EC2 e configura e richiedi istanze spot Amazon EC2](#).

Argomenti aggiuntivi in questa sezione

- [AWSstrumenti correlati aAWS SDK for .NET](#)
- [AWSSDK e strumenti di riferimento per](#)
- [Risorse aggiuntive](#)

AWS strumenti correlati a AWS SDK for .NET

Strumenti per Windows PowerShell e Tools for PowerShell Core

I moduli di PowerShell AWS Tools for Windows PowerShell e AWS Tools for PowerShell Core sono basati sulle funzionalità esposte da AWS SDK for .NET. Gli strumenti PowerShell consentono di eseguire lo script di operazioni sul server dal prompt di PowerShell. Anche se i cmdlet sono implementati utilizzando i client del servizio e i metodi dall'SDK, forniscono un'esperienza idiomatica della PowerShell per specificare i parametri e gestire i risultati.

Per iniziare, consulta [AWS Tools for Windows PowerShell](#).

Toolkit for VS Code

[AWS Toolkit for Visual Studio Code](#) è un plugin per l'editor di codice di Visual Studio (VS Code). Il toolkit semplifica lo sviluppo, il debug e la distribuzione di applicazioni che utilizzano AWS.

Con il toolkit, puoi fare cose come le seguenti:

- Puoi creare applicazioni serverless che contengono funzioni AWS Lambda e quindi distribuire le applicazioni in uno stack AWS CloudFormation.
- Utilizzo di schemi Amazon EventBridge.
- Utilizzare IntelliSense quando si lavora con i file di definizione delle attività Amazon ECS.
- Visualizzazione di un'applicazione AWS Cloud Development Kit (AWS CDK)

Toolkit for Visual Studio

AWS Toolkit for Visual Studio è un plug-in per Visual Studio IDE che semplifica le attività di sviluppo, debug e distribuzione di applicazioni .NET che utilizzano Amazon Web Services. Il Toolkit for Visual Studio fornisce modelli Visual Studio per i servizi come Lambda e procedure guidate per la distribuzione di applicazioni Web e applicazioni serverless. Puoi utilizzare il plugin AWSExplorer per gestire le istanze Amazon EC2, lavorare con le tabelle Amazon DynamoDB, pubblicare messaggi sulle code Amazon Simple Notification Service (Amazon SNS) e molto altro, il tutto all'interno di Visual Studio.

Per iniziare, consulta [Configurazione di AWS Toolkit for Visual Studio](#).

Toolkit for Azure DevOps

AWS Toolkit for Microsoft Azure DevOps aggiunge attività che consentono di creare e rilasciare facilmente le pipeline in Azure DevOps e Azure DevOps Server per utilizzare i servizi AWS. Puoi lavorare con Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, Lambda, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) e Amazon SNS. Puoi inoltre eseguire comandi utilizzando il modulo di Windows PowerShell e AWS Command Line Interface (AWS CLI).

Per iniziare a utilizzare AWS Toolkit for Azure DevOps, consulta [AWS Toolkit for Microsoft Azure DevOps Guida per l'utente di](#).

AWSSDK e strumenti di riferimento per

La [AWSSDK e strumenti di riferimento per](#) contiene informazioni rilevanti e importanti per molti dei AWSSDK e toolkit e il AWS CLI. Di seguito sono riportati alcuni esempi di informazioni che il riferimento contiene:

- Informazioni sul [condiviso AWS config credentials file](#) e la loro [posizione](#).
- [Configurazione di AWS account, utenti e ruoli](#)
- [Riferimento alle impostazioni di configurazione e autenticazione](#)
- [AWS Librerie Common Runtime \(CRT\)](#)
- [Policy di manutenzione degli SDK AWS e degli strumenti](#)
- [Matrice di supporto delle versioni degli SDK AWS e degli strumenti](#)

Risorse aggiuntive

Servizi supportati

AWS SDK for .NET supporta la maggior parte dei prodotti infrastrutturali AWS e di frequente vengono aggiunti ulteriori servizi. Per un elenco dei servizi AWS supportati dall'SDK, consulta l'apposito [file SDK README](#).

Cronologia delle revisioni

Per scoprire cosa è cambiato nelle varie versioni, consulta quanto segue:

- [Registro delle modifiche SDK](#)

- [Cosa c'è di nuovo nel AWS SDK for .NET](#)
- [Cronologia dei documenti](#)

Pagina iniziale per AWS SDK for .NET

Per ulteriori informazioni su AWS SDK for .NET, consulta la home page dell'SDK all'indirizzo <https://aws.amazon.com/sdk-for-net/>.

Documentazione di riferimento SDK

La documentazione di riferimento dell'SDK offre la possibilità di sfogliare e cercare in tutto il codice incluso nell'SDK. Fornisce documentazione completa ed esempi di utilizzo. Per ulteriori informazioni, consulta la [Documentazione di riferimento delle API di AWS SDK for .NET](#).

AWSRe:post (in precedenza AWSforum)

Visita [AWSRe: post](#), in particolare [il argomento per il AWS SDK for .NET](#), per porre domande o fornire feedback su AWS. Ogni pagina di documentazione ha un [Prova AWSRe: post](#) link nella parte inferiore della pagina che porta all'argomento relativo a Re:post.AWS. Gli ingegneri monitorano gli argomenti e rispondono a domande, feedback e problemi.

Se hai effettuato l'accesso a re:POST, puoi anche seguire un argomento. Per seguire l'argomento relativo al AWS SDK for .NET, vai al [Tutti gli argomenti pagina](#), trova «.NET su AWS» e seleziona il [Seguire pulsante](#).

Kit di strumenti

- AWS Toolkit for Visual Studio: Se utilizzi l'IDE di Microsoft Visual Studio, dovresti controllare il [AWS Toolkit for Visual Studio Guida per l'utente](#).
- AWS Toolkit for Visual Studio Code: Se utilizzi l'IDE di Microsoft Visual Studio, dovresti consultare il [AWS Toolkit for Visual Studio Code Guida per l'utente](#).

Librerie, estensioni e strumenti utili

Visita il [aws/dotneteseghe/aws-sdk-net](#) repository su GitHub sito Web per i collegamenti a librerie, strumenti e risorse che è possibile utilizzare per creare applicazioni e servizi.NET su AWS.

Di seguito vengono mostrati alcuni esempi:

- [AWS Estensione di configurazione.NET per Systems Manager](#)

- [AWSEstensioni.NET Core Setup](#)
- [AWSRegistrazione di.NET](#)
- [Libreria di estensioni di autenticazione Amazon Cognito](#)
- [SDK AWS X-Ray per .NET](#)

Altre risorse

Di seguito sono riportate altre risorse che potrebbero rivelarsi utili:

- [Rete per sviluppatori](#)
- [Ambiente di sviluppo.NET suAWSCloud: distribuzione di riferimento con avvio rapido](#)
- [Ciao, Cloud! blog](#)
- AWSWhite paper:[Sviluppo e distribuzione di applicazioni.NET suAWS](#)
- [AWS Microservice Extractor for .NET](#)
- [Porting Assistant per.NET](#)
- [AWSGuida di riferimento agli SDK e agli strumenti](#)

Nozioni di base su AWS SDK for .NET

Per utilizzare il AWS SDK for .NET, è necessario installare la toolchain e configurare una serie di elementi essenziali di cui l'applicazione ha bisogno per accedere ai AWS servizi. Ciò include:

- Un account o un ruolo utente appropriato
- Informazioni di autenticazione per quell'account utente o per assumere quel ruolo
- Specificazione della AWS regione
- AWSSDK pacchetti o assieme

Alcuni argomenti di questa sezione forniscono informazioni su come configurare questi elementi essenziali.

In altri argomenti di questa sezione e in altre sezioni vengono fornite informazioni su metodi più avanzati di configurazione del progetto.

Argomenti

- [Installa e configura la tua toolchain](#)
- [Configura l'autenticazione SDK con AWS](#)
- [Fai un rapido tour del AWS SDK for .NET](#)
- [Iniziare un nuovo progetto](#)
- [Configura la AWS regione](#)
- [Installa AWSSDK pacchetti con NuGet](#)
- [Installa gli assembly AWSSDK senza NuGet](#)
- [Risoluzione di credenziali e profili](#)
- [Ulteriori informazioni su utenti e ruoli](#)
- [Configurazione avanzata per il tuo AWS SDK for .NET progetto](#)
- [Uso delle credenziali legacy](#)

Installa e configura la tua toolchain

Per utilizzare AWS SDK for .NET, è necessario che siano installati determinati strumenti di sviluppo.

Sviluppo multiplatforma

Per lo sviluppo multiplatforma su Windows, Linux o macOS sono necessari i seguenti elementi:

- Microsoft [.NET Core SDK](#), versione 2.1, 3.1 o versioni successive, che include l'interfaccia a riga di comando (CLI) .NET (**dotnet**) e il .NET Core Runtime.
- Un editor di codice o un ambiente di sviluppo integrato (IDE) appropriato per il sistema operativo e i requisiti in uso. Si tratta in genere di un programma che fornisce un certo supporto per .NET Core.

Gli esempi includono [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) e [Microsoft Visual Studio](#).

- (Facoltativo) Un AWS kit di strumenti, se disponibile, per l'editor scelto e il sistema operativo in uso.

Gli esempi includono [AWS Toolkit for Visual Studio Code](#), [AWS Toolkit for JetBrains](#), e [AWS Toolkit for Visual Studio](#).

Windows con Visual Studio e .NET Core

Per lo sviluppo su Windows con Visual Studio e .NET Core sono necessari i seguenti elementi:

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1, 3.1 o versioni successive

Questo è in genere incluso per impostazione predefinita quando si installa una versione recente di Visual Studio.

- (Facoltativo) [AWS Toolkit for Visual Studio](#), che è un plug-in che fornisce un'interfaccia utente per la gestione AWS delle risorse e dei profili locali da Visual Studio. Per installare il toolkit, vedi [Configurazione di AWS Toolkit for Visual Studio](#).

Per ulteriori informazioni, consulta la [AWS Toolkit for Visual Studio Guida per l'utente](#).

Approfondimenti

[Configura l'autenticazione SDK con AWS](#)

Configura l'autenticazione SDK con AWS

È necessario definire la modalità di autenticazione del codice con AWS durante lo sviluppo con i Servizi AWS. Esistono diversi modi per configurare l'accesso programmatico alle risorse AWS, a seconda dell'ambiente e dell'accesso AWS disponibili.

Per visualizzare i vari metodi di autenticazione per l'SDK, consulta [Autenticazione e accesso nella Guida](#) di riferimento agli AWSSDK e agli strumenti.

Questo argomento presuppone che un nuovo utente stia sviluppando in locale, non disponga di un metodo di autenticazione fornito dal datore di lavoro e utilizzi AWS IAM Identity Center per ottenere credenziali temporanee. Se il tuo ambiente non rientra in questi presupposti, alcune delle informazioni contenute in questo argomento potrebbero non riguardarti o alcune potrebbero essere già state fornite.

La configurazione di questo ambiente richiede diversi passaggi, riassunti di seguito:

1. [Abilitazione e configurazione di IAM Identity Center](#)
2. [Configura l'SDK per utilizzare IAM Identity Center.](#)
3. [Avvio di una sessione del portale di accesso AWS](#)

Abilitazione e configurazione di IAM Identity Center

Per utilizzare IAM Identity Center, deve prima essere abilitato e configurato. Per maggiori dettagli su come eseguire questa operazione per l'SDK, consulta la Fase 1 dell'argomento relativo all'[autenticazione di IAM Identity Center nella Guida](#) di riferimento agli AWS SDK and Tools. In particolare, segui le istruzioni necessarie riportate nella sezione I do not have established access through IAM Identity Center (Non ho stabilito l'accesso tramite IAM Identity Center).

Configura l'SDK per utilizzare IAM Identity Center.

Le informazioni su come configurare l'SDK per l'utilizzo di IAM Identity Center si trovano nella fase 2 dell'argomento relativo all'[autenticazione di IAM Identity Center nella Guida](#) di riferimento agli AWSSDK e agli strumenti. Dopo aver completato questa configurazione, il sistema deve contenere i seguenti elementi:

- La AWS CLI, che viene utilizzata per avviare una sessione del portale di accesso AWS prima di eseguire l'applicazione.
- Il AWS config file condiviso che contiene un [\[default\]profilo](#) con un set di valori di configurazione a cui è possibile fare riferimento dall'SDK. Per trovare la posizione di questo file, consulta l'argomento relativo alla [posizione dei file condivisi](#) nella Guida di riferimento per SDK e strumenti AWS. AWS SDK for .NET utilizza il provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il valore `sso_role_name`, che è un ruolo IAM connesso a un set di autorizzazioni di IAM Identity Center, dovrebbe consentire l'accesso ai Servizi AWS utilizzati nell'applicazione.

Il seguente file config di esempio mostra un profilo predefinito impostato con il provider di token SSO. L'impostazione `sso_session` del profilo si riferisce alla sezione `sso-session` denominata. La sezione `sso-session` contiene le impostazioni per avviare una sessione del portale di accesso AWS.

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

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

Important

Se si utilizza AWS IAM Identity Center per l'autenticazione, l'applicazione deve fare riferimento ai seguenti NuGet pacchetti in modo che la risoluzione SSO possa funzionare:

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Il mancato riferimento a questi pacchetti comporterà un'eccezione di runtime.

Avvio di una sessione del portale di accesso AWS

Prima di eseguire un'applicazione con accesso Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché l'SDK utilizzi l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e l'SDK risconterà un errore di autenticazione. Per accedere al portale di accesso AWS, esegui il seguente comando nella AWS CLI.

```
aws sso login
```

Poiché disponi di una configurazione predefinita del profilo, non devi chiamare il comando con un'opzione `--profile`. Se la configurazione del provider di token SSO utilizza un profilo denominato, il comando è `aws sso login --profile named-profile`.

Per verificare se hai già una sessione attiva, esegui il seguente comando AWS CLI.

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

La risposta a questo comando dovrebbe restituire l'account IAM Identity Center e il set di autorizzazioni configurati nel file `config` condiviso.

Note

Se hai già una sessione attiva del portale di accesso AWS e hai già eseguito `aws sso login`, non ti verrà richiesto di fornire credenziali.

La procedura di accesso potrebbe chiederti di autorizzare l'accesso ai dati da parte della AWS CLI. Poiché la AWS CLI è basata sull'SDK per Python, i messaggi di autorizzazione possono contenere variazioni del nome `botocore`.

Informazioni aggiuntive

- Per ulteriori informazioni sull'utilizzo di IAM Identity Center e SSO in un ambiente di sviluppo, consulta [Autenticazione unica](#) la sezione. [Autenticazione avanzata](#) Queste informazioni includono metodi alternativi e più avanzati, oltre a tutorial che mostrano come utilizzare questi metodi.
- Per ulteriori opzioni sull'autenticazione per l'SDK, come l'uso di profili e variabili di ambiente, consulta il capitolo sulla [configurazione](#) nella Guida di riferimento agli AWSSDK e agli strumenti.

- Per ulteriori informazioni sulle best practice, consulta [Best practice per la sicurezza in IAM](#) nella Guida per l'utente di IAM.
- Per creare credenziali AWS a breve termine, consulta [Credenziali di sicurezza temporanee in IAM](#) nella Guida per l'utente di IAM.
- Per ulteriori informazioni su altri provider di credenziali, consulta l'argomento relativo ai [provider di credenziali standard](#) nella Guida di riferimento per SDK e strumenti AWS.

Fai un rapido tour del AWS SDK for .NET

Questa sezione fornisce tutorial di base per gli sviluppatori che non conoscono. AWS SDK for .NET

Note

[Prima di utilizzare questi tutorial, è necessario aver installato la toolchain e configurato l'autenticazione SDK.](#)

Per informazioni sullo sviluppo di software per AWS servizi specifici e esempi di codice, consulta [Lavora con AWS i servizi](#) Per ulteriori esempi di codice, vedere [AWS SDK for .NET esempi di codice](#).

Argomenti

- [Semplice applicazione multiplatforma con AWS SDK for .NET](#)
- [Semplice applicazione basata su Windows con AWS SDK for .NET](#)
- [Fasi successive](#)

Semplice applicazione multiplatforma con AWS SDK for .NET

Questo tutorial utilizza e.NET Core per lo sviluppo multiplatforma. AWS SDK for .NET II tutorial mostra come utilizzare l'SDK per elencare i bucket [Amazon S3](#) di cui sei proprietario e, facoltativamente, creare un bucket.

Eseguirai questo tutorial utilizzando strumenti multiplatforma come l'interfaccia a riga di comando (CLI) .NET. Per altri modi di configurare il tuo ambiente di sviluppo, consulta [Installa e configura la tua toolchain](#)

Necessario per lo sviluppo multiplatforma in Windows, Linux o macOS:

- Microsoft [.NET Core SDK](#), versione 2.1, 3.1 o versioni successive, che include l'interfaccia a riga di comando (CLI) .NET (**dotnet**) e il .NET Core Runtime.
- Un editor di codice o un ambiente di sviluppo integrato (IDE) appropriato per il sistema operativo e i requisiti in uso. Si tratta in genere di un programma che fornisce un certo supporto per .NET Core.

Gli esempi includono [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) e [Microsoft Visual Studio](#).

Note

Prima di utilizzare questi tutorial, è necessario aver installato la toolchain e configurato l'autenticazione SDK.

Fasi

- [Creazione del progetto](#)
- [Creazione del codice](#)
- [Esecuzione dell'applicazione.](#)
- [Rimozione](#)

Creazione del progetto

1. Apri il prompt dei comandi o il terminale. Trovare o creare una cartella del sistema operativo in cui è possibile creare un progetto.NET.
2. In tale cartella, eseguire il seguente comando per creare il progetto.NET.

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

3. Vai alla `S3CreateAndList` cartella appena creata ed esegui i seguenti comandi:

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

I comandi precedenti installano i NuGet pacchetti dal [gestore di NuGet pacchetti](#). Poiché sappiamo esattamente di quali NuGet pacchetti abbiamo bisogno per questo tutorial, ora possiamo eseguire questo passaggio. È anche comune che i pacchetti richiesti vengano resi noti durante lo sviluppo. Quando ciò accade, un comando simile può essere eseguito in quel momento.

Creazione del codice

1. Nella cartella `S3CreateAndList`, trovare e aprire `Program.cs` nell'editor del codice.
2. Sostituire il contenuto con il seguente codice e salvare il file.

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

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

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

Esecuzione dell'applicazione.

1. Esegui il comando seguente.

```
dotnet run
```

2. Esamina l'output per vedere il numero di bucket Amazon S3 che possiedi, se ce ne sono, e i relativi nomi.
3. Scegli un nome per un nuovo bucket Amazon S3. Usa "dotnet-quicktour-s3-1-cross->" come base e aggiungi qualcosa di unico, come un GUID o il tuo nome. Assicurati di seguire le regole per i nomi dei bucket, come descritto in [Regole per la denominazione dei bucket](#) nella [Amazon S3 User Guide](#).
4. Eseguire il comando seguente, sostituendo *BUCKET_NAME* con il nome del bucket scelto.

```
dotnet run BUCKET-NAME
```

5. Esaminare l'output per visualizzare il nuovo bucket creato.

Rimozione

Durante l'esecuzione di questo tutorial, hai creato alcune risorse che puoi scegliere di pulire in questo momento.

- [Se non desideri conservare il bucket creato dall'applicazione in un passaggio precedente, eliminalo utilizzando la console Amazon S3 all'indirizzo `https://console.aws.amazon.com/s3/`.](#)
- Se non desideri mantenere il progetto.NET, rimuovi la cartella `S3CreateAndList` dall'ambiente di sviluppo.

Fasi successive

Torna al [menu del tour rapido](#) o vai direttamente alla [fine di](#) questo tour rapido.

Semplice applicazione basata su Windows con AWS SDK for .NET

Questo tutorial utilizza Windows con Visual Studio e .NET Core. AWS SDK for .NET Il tutorial mostra come utilizzare l'SDK per elencare i bucket [Amazon S3](#) di cui sei proprietario e, facoltativamente, creare un bucket.

Esegui questo tutorial su Windows utilizzando Visual Studio e .NET Core. Per altri modi di configurare il tuo ambiente di sviluppo, consulta. [Installa e configura la tua toolchain](#)

Necessario per lo sviluppo su Windows con Visual Studio e .NET Core:

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1, 3.1 o versioni successive

Questo è in genere incluso per impostazione predefinita quando si installa una versione recente di Visual Studio.

Note

Prima di utilizzare questi tutorial, è necessario aver [installato la toolchain e configurato l'autenticazione SDK](#).

Fasi

- [Creazione del progetto](#)
- [Creazione del codice](#)

- [Esecuzione dell'applicazione.](#)
- [Rimozione](#)

Creazione del progetto

1. Apri Visual Studio e crea un nuovo progetto che utilizzi la versione C# del modello dell'app Console, ovvero con la descrizione: «... per creare un'applicazione da riga di comando che può essere eseguita su.NET...». Assegnare un nome al progetto `S3CreateAndList`.

Note

Non scegliere la versione.NET Framework del modello di app per console o, in tal caso, assicurati di utilizzare.NET Framework 4.6.2 o versione successiva.

2. Con il progetto appena creato caricato, scegli Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Cerca i seguenti NuGet pacchetti e installali nel progetto:
`AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO`, e `AWSSDK.SSO0IDC`

Questo processo installa i NuGet pacchetti dal [gestore di NuGet pacchetti](#). Poiché sappiamo esattamente di quali NuGet pacchetti abbiamo bisogno per questo tutorial, ora possiamo eseguire questo passaggio. È anche comune che i pacchetti richiesti vengano resi noti durante lo sviluppo. Quando ciò accade, seguire un processo simile per installarli in quel momento.

4. Se intendi eseguire l'applicazione dal prompt dei comandi, apri subito un prompt dei comandi e vai alla cartella che conterrà l'output della build. In genere si tratta di qualcosa di simile `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, ma dipenderà dall'ambiente in uso.

Creazione del codice

1. Nel progetto `S3CreateAndList`, trovare e aprire `Program.cs` nell'IDE.
2. Sostituire il contenuto con il seguente codice e salvare il file.

```
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($"
SSO Profile:
{await
ssoProfileClient.GetCallerIdentityArn()}");

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

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

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

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

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

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

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

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

3. Per compilare l'applicazione

Note

Se usi una versione precedente di Visual Studio, potresti ricevere un errore di compilazione simile al seguente:

«La funzionalità 'async main' non è disponibile in C# 7.0. Utilizza la versione linguistica 7.1 o successiva.»

Se ricevi questo errore, configura il progetto per utilizzare una versione successiva della lingua. Questa operazione viene in genere eseguita nelle proprietà del progetto, Build, Advanced.

Esecuzione dell'applicazione.

1. Eseguire l'applicazione senza argomenti della riga di comando. Fatelo nel prompt dei comandi (se ne avete aperto uno in precedenza) o dall'IDE.
2. Esamina l'output per vedere il numero di bucket Amazon S3 che possiedi, se ce ne sono, e i relativi nomi.
3. Scegli un nome per un nuovo bucket Amazon S3. Usa "dotnet-quicktour-s3-1-winv5-" come base e aggiungi qualcosa di unico, ad esempio un GUID o il tuo nome. Assicurati di seguire le regole per i nomi dei bucket, come descritto in [Rules for Bucket Naming](#) nella [Amazon S3 User Guide](#).
4. Eseguire di nuovo l'applicazione, questa volta fornendo il nome del bucket.

Nella riga di comando, sostituisci *BUCKET-NAME* nel comando seguente con il nome del bucket che hai scelto.

```
S3CreateAndList BUCKET-NAME
```

Oppure, se stai eseguendo l'applicazione nell'IDE, scegli Project, S3 CreateAndList Properties, Debug e inserisci lì il nome del bucket.

5. Esaminare l'output per visualizzare il nuovo bucket creato.

Rimozione

Durante l'esecuzione di questo tutorial, hai creato alcune risorse che puoi scegliere di pulire in questo momento.

- [Se non desideri conservare il bucket creato dall'applicazione in un passaggio precedente, eliminalo utilizzando la console Amazon S3 all'indirizzo `https://console.aws.amazon.com/s3/`.](#)
- Se non desideri mantenere il progetto.NET, rimuovi la cartella `S3CreateAndList` dall'ambiente di sviluppo.

Fasi successive

Torna al [menu del tour rapido](#) o vai direttamente alla [fine di](#) questo tour rapido.

Fasi successive

Assicurarti di eliminare tutte le risorse residue create durante l'esecuzione di questi tutorial. Potrebbero trattarsi di AWS risorse o risorse presenti nell'ambiente di sviluppo, ad esempio file e cartelle.

Ora che hai visitato ilAWS SDK for .NET, potresti voler [iniziare il tuo progetto](#).

Iniziare un nuovo progetto

Esistono diverse tecniche che è possibile utilizzare per avviare un nuovo progetto di accesso ai AWS servizi. Di seguito sono riportate alcune di queste tecniche:

- Se non conosci lo sviluppo su.NET AWS o almeno non lo seiAWS SDK for .NET, puoi vedere esempi completi in[Fai un breve tour](#). Fornisce un'introduzione all'SDK.
- È possibile avviare un progetto di base utilizzando l'interfaccia della riga di comando .NET. Per vederne un esempio, apri un prompt dei comandi o un terminale, create una cartella o una directory e accedete ad essa, quindi immettete quanto segue.

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

Viene creato un progetto vuoto al quale è possibile aggiungere codice e NuGet pacchetti. Per ulteriori informazioni, vedere la [guida di .NET Core](#).

Per visualizzare un elenco di modelli di progetto, utilizza quanto segue: `dotnet new --list`

- AWS Toolkit for Visual Studio include modelli di progetto C# per diversi servizi AWS. Dopo aver [installato il toolkit](#) in Visual Studio, puoi accedere ai modelli durante la creazione di un nuovo progetto.

Per visualizzarlo, vai a [Lavorare con AWS i servizi](#) nella [Guida per l'AWS Toolkit for Visual Studio utente](#). Molti degli esempi in quella sezione creano nuovi progetti.

- Se sviluppi con Visual Studio su Windows ma senza il AWS Toolkit for Visual Studio, usa le tecniche tipiche per creare un nuovo progetto.

Per vedere un esempio, apri Visual Studio e scegli File, Nuovo, Progetto. Cerca «.net core» e scegli la versione C# del modello Console App (.NET Core) o WPF App (.NET Core). Viene creato un progetto vuoto a cui puoi aggiungere codice e NuGet pacchetti.

Puoi trovare alcuni esempi di come lavorare con AWS i servizi in [Esempi di codice con indicazioni](#).

Important

Se si utilizza AWS IAM Identity Center per l'autenticazione, l'applicazione deve fare riferimento ai seguenti NuGet pacchetti in modo che la risoluzione SSO possa funzionare:

- AWSSDK.SSO
- AWSSDK.SSO0IDC

Il mancato riferimento a questi pacchetti comporterà un'eccezione di runtime.

Configura la AWS regione

AWSLe regioni consentono di accedere ai AWS servizi che risiedono fisicamente in un'area geografica specifica. Questa funzionalità può essere utile per la ridondanza e per mantenere i dati e le applicazioni in esecuzione vicino ai punti di accesso ai servizi stessi.

Per visualizzare l'elenco corrente di tutte le regioni e gli endpoint supportati per ogni AWS servizio, consulta [Endpoint e quote del servizio](#) in. Riferimenti generali di AWS [Per visualizzare un elenco degli endpoint regionali esistenti, consulta Endpoint di servizio. AWS](#) Per visualizzare informazioni dettagliate sulle regioni, consulta [Specificare le AWS regioni che il tuo account può utilizzare](#).

Puoi creare un client AWS di servizio destinato a una [particolare regione](#). È inoltre possibile configurare l'applicazione con una regione che verrà utilizzata per [tutti i client AWS di servizio](#). Questi due casi vengono illustrati di seguito.

Crea un client di servizio con una regione particolare

È possibile specificare la regione per qualsiasi client di AWS servizio nell'applicazione. L'impostazione della regione in questo modo ha la precedenza su qualsiasi impostazione globale per quel particolare client di servizio.

Regione esistente

Questo esempio mostra come creare un'istanza di un client [Amazon EC2](#) in una regione esistente. Utilizza campi definiti. [RegionEndpoint](#)

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

Nuova regione che utilizza la RegionEndpoint classe

[Questo esempio mostra come costruire un nuovo endpoint Region utilizzando. 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
}
```

Nuova regione che utilizza la classe di configurazione del client di servizio

Questo esempio mostra come utilizzare la ServiceURL proprietà della classe di configurazione del client di servizio per specificare la regione; in questo caso, utilizzando la classe [AmazonEC2Config](#).

Questa tecnica funziona anche se l'endpoint Region non segue il normale modello di endpoint Region.

```
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
}
```

Specificare una regione per tutti i client di servizio

Esistono diversi modi per specificare una regione per tutti i client di AWS servizio creati dall'applicazione. Questa regione viene utilizzata per i client di servizio che non sono stati creati con una regione particolare.

AWS SDK for .NET Cerca un valore Region nell'ordine seguente.

Profili

Imposta un profilo caricato dall'applicazione o dall'SDK. Per ulteriori informazioni, consulta [Risoluzione di credenziali e profili](#).

Variabili di ambiente

Impostato nella variabile di AWS_REGION ambiente.

Su Linux o macOS

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

In Windows:

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

Note

Se imposti questa variabile di ambiente per l'intero sistema (usando `export` o `setx`), avrà effetto su tutti gli SDK e i toolkit, non solo su AWS SDK for .NET

AWSConfigs classe

Impostato come [AWSConfigs.AWSRegion](#) proprietà.

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

Risoluzione della regione

Se nessuno dei metodi sopra descritti viene utilizzato per specificare un'AWS Regione, i AWS SDK for .NET tentativi di trovare una regione in cui il client del AWS servizio possa operare.

Ordine di risoluzione della regione

1. File di configurazione dell'applicazione come `app.config` e `web.config`.
2. Variabili di ambiente (`AWS_REGION` e `AWS_DEFAULT_REGION`).
3. Un profilo con il nome specificato da un valore in `AWSConfigs.AWSProfileName`.
4. Un profilo con il nome specificato dalla variabile di `AWS_PROFILE` ambiente.
5. Il `[default]` profilo.
6. Metadati dell'istanza Amazon EC2 (se in esecuzione su un'istanza EC2).

Se non viene trovata alcuna regione, l'SDK genera un'eccezione che indica che il client del AWS servizio non ha una regione configurata.

Informazioni speciali sulla regione Cina (Pechino)

Per utilizzare i servizi nella Regione Cina (Pechino), devi avere un account e le credenziali specifiche per tale Regione. Gli account e le credenziali di altre Regioni AWS non funzioneranno per la Regione Cina (Pechino). Analogamente, gli account e le credenziali della Regione Cina (Pechino) non funzioneranno per altre Regioni AWS. Per informazioni sugli endpoint e i protocolli disponibili nella regione Cina (Pechino), vedi [Beijing Region Endpoints](#).

Informazioni speciali sui nuovi servizi AWS

AWSI nuovi servizi possono essere lanciati inizialmente in alcune regioni e poi supportati in altre regioni. In questi casi non è necessario installare l'SDK più recente per accedere alle nuove regioni per quel servizio. È possibile specificare le nuove regioni aggiunte per cliente o a livello globale, come illustrato in precedenza.

Installa AWSSDK pacchetti con NuGet

[NuGet](#) è un sistema di gestione dei pacchetti per la piattaforma .NET. Con NuGet, puoi installare i [AWSSDKpacchetti](#), oltre a diverse altre estensioni, nel tuo progetto. Per ulteriori informazioni, consulta il repository [aws/dotnet](#) sul sito Web. GitHub

NuGet ha sempre le versioni più recenti dei AWSSDK pacchetti, oltre alle versioni precedenti. NuGet è consapevole delle dipendenze tra i pacchetti e installa automaticamente tutti i pacchetti richiesti.

Warning

L'elenco dei NuGet pacchetti potrebbe includerne uno chiamato semplicemente "AWSSDK" (senza identificatore aggiunto). NON installate questo NuGet pacchetto; è obsoleto e non dovrebbe essere usato per nuovi progetti.

I pacchetti installati con NuGet vengono archiviati con il progetto anziché in una posizione centrale. In questo modo puoi installare le versioni delle assembly specifiche per una determinata applicazione senza creare problemi di compatibilità alle altre applicazioni. Per ulteriori informazioni in merito NuGet, consulta la [NuGet documentazione](#).

Note

Se non potete o non siete autorizzati a scaricare e installare NuGet pacchetti per progetto, potete ottenere gli AWSSDK assembly e archivarli localmente (o in locale).

Se questo è il vostro caso e non avete ancora ottenuto gli assembly, vedete. AWSSDK [Ottenere assieme AWSSDK](#) Per informazioni su come utilizzare gli assieme memorizzati localmente, vedere. [Installa gli assembly AWSSDK senza NuGet](#)

Utilizzo NuGet dal prompt dei comandi o dal terminale

1. Vai ai [AWSSDK pacchetti NuGet](#) e determina quali pacchetti ti servono nel tuo progetto, ad esempio [AWSSDK.S3](#).
2. Copiare il comando.NET CLI dalla pagina Web del pacchetto, come illustrato nell'esempio seguente.

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

3. Nella directory del progetto, esegui il comando.NET CLI. NuGet [installa anche eventuali dipendenze, ad esempio .Core. AWSSDK](#)

Note

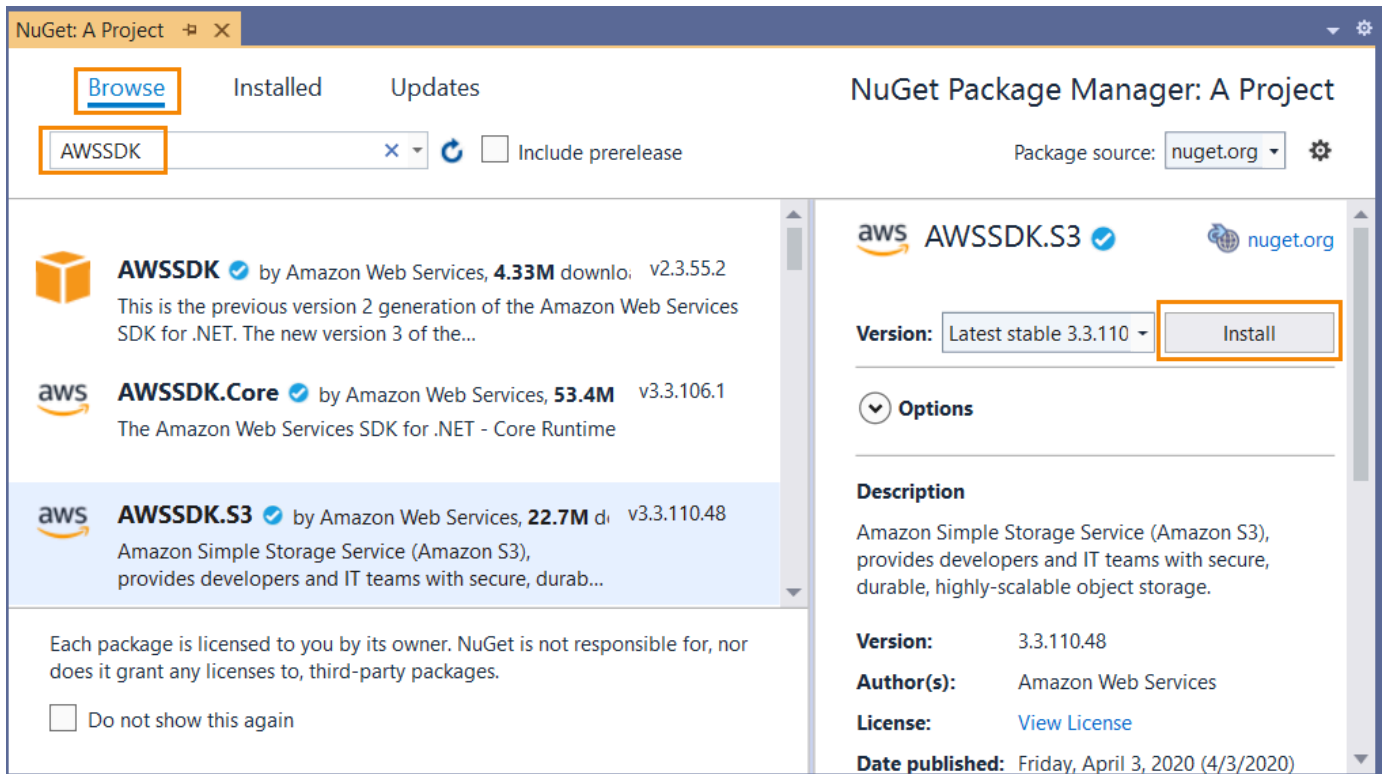
Se desideri solo la versione più recente di un NuGet pacchetto, puoi escludere le informazioni sulla versione dal comando, come mostrato nell'esempio seguente.

```
dotnet add package AWSSDK.S3
```

Utilizzo NuGet da Visual Studio Solution Explorer

1. In Solution Explorer, fai clic con il pulsante destro del mouse sul progetto, quindi scegli Gestisci NuGet pacchetti dal menu contestuale.
2. Nel riquadro sinistro del NuGet Package Manager, scegli Browse. È quindi possibile utilizzare la casella di ricerca per cercare il pacchetto che si desidera installare. NuGet [installa anche tutte le dipendenze, come AWSSDK .Core.](#)

La figura seguente mostra l'installazione del pacchetto.S3AWSSDK.



Utilizzo NuGet dalla console di Package Manager

In Visual Studio, scegli Tools, NuGet Package Manager, Package Manager Console.

È possibile installare i AWSSDK pacchetti desiderati dalla console di Package Manager utilizzando il **Install-Package** comando. Ad esempio, per installare [AWSSDK.S3](#), utilizzare il comando seguente.

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

NuGet [installa anche tutte le dipendenze, ad esempio .Core. AWSSDK](#)

Se è necessario installare una versione precedente di un pacchetto, utilizzare l'-Version'opzione e specificare la versione del pacchetto desiderata, come illustrato nell'esempio seguente.

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

Per ulteriori informazioni sui comandi della console di Package Manager, vedere il [PowerShellriferimento](#) nella [NuGetdocumentazione](#) di Microsoft.

Installa gli assembly AWSSDK senza NuGet

In questo argomento viene descritto come utilizzare gli assembly AWSSDK ottenuti e archiviati localmente (o in locale) come descritto in [Ottenere assieme AWSSDK](#). Questo è non il metodo consigliato per la gestione dei riferimenti SDK, ma è richiesto in alcuni ambienti.

Note

Il metodo consigliato per la gestione dei riferimenti SDK consiste nel scaricare e installare solo i pacchetti NuGet di cui ogni progetto ha bisogno. Questo metodo è descritto in [Installa AWSSDK pacchetti con NuGet](#).

Per installare gli assembly AWSSDK

1. Crea una cartella nell'area di progetto per gli assembly AWSSDK richiesti. È ad esempio possibile chiamare questa cartella `AwsAssemblies`.
2. Se non è già stato fatto, [ottenere gli assieme AWSSDK](#), che colloca gli assembly in alcune cartelle di download o installazione locali. Copiare i file DLL per gli assembly richiesti da quella cartella di download nel progetto (nella `AwsAssemblies` cartella, nel nostro esempio).

Assicurati di copiare anche qualsiasi dipendenza. Puoi trovare informazioni sulle dipendenze sul [GitHub](#) sito Web.

3. Fare riferimento agli assieme richiesti come segue.

Cross-platform development

1. Aprire il progetto `.csproj` file e aggiungi un `<ItemGroup>` elemento.
2. Nella `<ItemGroup>` elemento, aggiungi un `<Reference>` elemento con un `Include` attributo per ogni assieme richiesto.

Ad esempio, per Amazon S3, puoi aggiungere le seguenti righe al tuo progetto `.csproj` file.

Su Linux e macOS:

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

```
</ItemGroup>
```

In Windows:

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

3. Salva il tuo progetto .csprojfile.

Windows with Visual Studio and .NET Core

1. In Visual Studio carica il progetto e apriProgetto,Aggiungi riferimento.
2. SelezionaSfogliapulsante nella parte inferiore della finestra di dialogo. Passare alla cartella del progetto e alla sottocartella in cui sono stati copiati i file DLL richiesti (AwsAssemblies, ad esempio).
3. Seleziona tutti i file DLL, scegliInseriscie scegliOK.
4. Salvare il progetto.

Risoluzione di credenziali e profili

AWS SDK for .NETCerca le credenziali in un determinato ordine e utilizza il primo set disponibile per l'applicazione corrente.

Ordine di ricerca delle credenziali

1. Credenziali impostate esplicitamente sul client del AWS servizio, come descritto in. [Accesso a credenziali e profili in un'applicazione](#)

Note

Questo argomento è incluso nella [Considerazioni speciali](#) sezione perché non è il metodo preferito per specificare le credenziali.

2. [Un profilo di credenziali con il nome specificato da un valore in. AWSConfigs AWSProfileName.](#)
3. Un profilo di credenziali con il nome specificato dalla variabile di AWS_PROFILE ambiente.

4. Il profilo delle credenziali [default].
5. [Sessione AWSCredentials](#) creata dalle variabili di `AWS_SESSION_TOKEN` ambiente `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY`, e, se non sono tutte vuote.
6. [Di base AWSCredentials](#) che vengono create dalle variabili `AWS_ACCESS_KEY_ID` di `AWS_SECRET_ACCESS_KEY` ambiente, se entrambe non sono vuote.
7. [IAM Roles for Tasks](#) for Amazon ECS task.
8. Metadati delle istanze Amazon EC2.

Se l'applicazione è in esecuzione su un'istanza Amazon EC2, ad esempio in un ambiente di produzione, utilizza un ruolo IAM come descritto in [Concessione dell'accesso utilizzando un ruolo IAM](#). Altrimenti, ad esempio nei test preliminari alla release, archivia le credenziali in un file che utilizza il formato di file delle AWS credenziali a cui l'applicazione Web ha accesso sul server.

Risoluzione del profilo

Con due diversi meccanismi di archiviazione delle credenziali, è importante capire come configurarle AWS SDK for .NET per utilizzarle. [IAWSConfigs. AWSProfilesLocation](#) la proprietà controlla il modo in cui AWS SDK for .NET trova i profili di credenziali.

AWSProfilesLocation	Comportamento di risoluzione del profilo
null (non impostato) o vuoto	Cerca nell'SDK Store se la piattaforma lo supporta, quindi cerca il file delle AWS credenziali condivise nella posizione predefinita . Se il profilo non si trova in nessuna di queste posizioni, cerca <code>~/ .aws/config</code> (Linux o macOS) o <code>%USERPROFILE%\ .aws \config</code> (Windows).
Il percorso di un file nel formato di file delle AWS credenziali	Cerca solo un profilo con uno specifico nome nel file specificato.

Utilizzo delle credenziali dell'account utente federato

Le applicazioni che utilizzano il AWS SDK for .NET ([AWSSDK.Core](#) versione 3.1.6.0 e successive) possono utilizzare account utente federati tramite Active Directory Federation Services (ADFS) per accedere ai servizi utilizzando Security Assertion Markup AWS Language (SAML).

Il supporto degli accessi federati indica che gli utenti possono autenticarsi utilizzando Active Directory. Le credenziali temporanee vengono concesse automaticamente all'utente. Queste credenziali temporanee, valide per un'ora, vengono utilizzate quando l'applicazione richiama i servizi. AWS L'SDK gestisce le credenziali temporanee. Per gli account degli utenti aggiunti al dominio, se l'applicazione effettua una chiamata, ma le credenziali sono scadute, l'utente viene riautenticato automaticamente e vengono concesse nuove credenziali. (Per non-domain-joined gli account, all'utente viene richiesto di inserire le credenziali prima della riautenticazione.)

Per utilizzare questo supporto nell'applicazione.NET, è necessario innanzitutto configurare il profilo del ruolo utilizzando un cmdlet. PowerShell [Per informazioni su come, consulta la AWS Tools for Windows PowerShell documentazione.](#)

Dopo aver impostato il profilo del ruolo, fai riferimento al profilo nella tua candidatura. Esistono diversi modi per eseguire questa operazione, uno dei quali consiste nell'utilizzare il [AWSConfigs.AWSProfileName](#) proprietà nello stesso modo in cui lo faresti con altri profili di credenziali.

[L'AWS Security Token Serviceassemblaggio \(AWSSDK.SecurityToken\)](#) fornisce il supporto SAML per ottenere AWS le credenziali. Per utilizzare le credenziali degli account utente federati, assicurati che questo assembly sia disponibile per l'applicazione.

Specificare ruoli o credenziali temporanee

Per le applicazioni eseguite su istanze Amazon EC2, il modo più sicuro per gestire le credenziali consiste nell'utilizzare i ruoli IAM, come descritto in [Concessione dell'accesso utilizzando un ruolo IAM](#)

Per gli scenari applicativi in cui l'eseguibile del software è disponibile per gli utenti esterni all'organizzazione, consigliamo di progettare il software in modo da utilizzare credenziali di sicurezza temporanee. Oltre a fornire un accesso limitato alle AWS risorse, queste credenziali hanno il vantaggio di scadere dopo un determinato periodo di tempo. Per ulteriori informazioni sull'utilizzo delle credenziali di sicurezza temporanee, vedi quanto segue:

- [Credenziali di sicurezza temporanee](#)
- [Pool di identità Amazon Cognito](#)

Utilizzo di credenziali proxy

Se il software comunica con AWS il proxy, è possibile specificare le credenziali per il proxy utilizzando la `ProxyCredentials` proprietà della `Config` classe di un servizio. La `Config` classe di un servizio fa in genere parte del namespace principale del servizio. Gli esempi includono quanto segue: [AmazonCloudDirectoryConfig](#) in [Amazon.CloudDirectory](#) namespace e [AmazonGameLiftConfig](#) in [Amazon.GameLift](#) namespace.

Per [Amazon S3](#), ad esempio, è possibile utilizzare un codice simile al seguente, dove `SecurelyStoredUserName` e `SecurelyStoredPassword` sono il nome utente e la password del proxy specificati in un [NetworkCredential](#) oggetto.

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

Note

Le versioni precedenti dell'SDK utilizzavano `ProxyUsername` e `ProxyPassword`, ma queste proprietà sono obsolete.

Ulteriori informazioni su utenti e ruoli

Per lo sviluppo AWS o l'esecuzione di applicazioni.NETAWS, è necessario disporre di una combinazione di utenti, set di autorizzazioni e ruoli di servizio appropriata per queste attività.

Gli utenti, i set di autorizzazioni e i ruoli di servizio specifici creati e il modo in cui li si utilizza dipenderanno dai requisiti delle applicazioni. Di seguito sono riportate alcune informazioni aggiuntive sul motivo per cui potrebbero essere utilizzati e su come crearli.

Utenti e set di autorizzazioni

Sebbene sia possibile utilizzare un account utente IAM con credenziali a lungo termine per accedere ai servizi AWS, questa non è più una best practice e dovrebbe essere evitata. Anche durante lo sviluppo, è consigliabile creare utenti e set di autorizzazioni in AWS IAM Identity Center e utilizzare credenziali temporanee fornite da un'origine di identità.

Per lo sviluppo, puoi utilizzare l'utente che hai creato o assegnato in [Configura l'autenticazione SDK](#). Se disponi delle autorizzazioni AWS Management Console appropriate, puoi anche creare diversi set di autorizzazioni con il privilegio minimo per quell'utente o creare nuovi utenti specificamente per progetti di sviluppo, fornendo set di autorizzazioni con il privilegio minimo. L'eventuale operazione scelta dipenderà dalle circostanze.

Per ulteriori informazioni su questi utenti e set di autorizzazioni e su come crearli, consulta l'argomento relativo [all'autenticazione e all'accesso](#) nella Guida di riferimento relativa a SDK e strumenti AWS e l'argomento relativo [alle nozioni di base](#) nella Guida per l'utente AWS IAM Identity Center.

☉ Ruoli di servizio

Puoi impostare un ruolo di servizio AWS per accedere ai servizi AWS per conto degli utenti. Questo tipo di accesso è appropriato se più persone eseguiranno l'applicazione in remoto, ad esempio su un'istanza Amazon EC2 creata a tale scopo.

Il processo di creazione di un ruolo di servizio varia a seconda della situazione, ma è essenzialmente simile a quanto indicato di seguito.

1. Accedi a AWS Management Console e apri la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Selezionare Roles (Ruoli), quindi selezionare Create role (Crea ruolo).
3. Scegli il servizio AWS, quindi come esempio trova e seleziona EC2 e scegli il caso d'uso EC2.
4. Scegliete Avanti: Autorizzazioni e selezionate le [politiche appropriate](#) per i AWS servizi che l'applicazione utilizzerà.

Warning

NON scegliete la AdministratorAccess politica perché tale politica consente le autorizzazioni di lettura e scrittura per quasi tutto il contenuto del vostro account.

5. Scegli Avanti: Tag e inserisci i tag che desideri.

Puoi trovare informazioni sui tag in [Control access using AWS resource tags](#) nella [IAM User Guide](#).

6. Scegli Avanti: esamina e fornisci un nome e una descrizione del ruolo. Quindi seleziona Create role (Crea ruolo).

Puoi trovare informazioni di alto livello sui ruoli IAM nelle [identità \(utenti, gruppi e ruoli\)](#) nella [IAM User Guide](#). Trova informazioni dettagliate sui ruoli nell'argomento sui [ruoli IAM](#) di quella guida.

Informazioni aggiuntive sui ruoli

- Usa [ruoli IAM](#) per le attività di Amazon Elastic Container Service (Amazon ECS).
- Usa [ruoli IAM](#) per le applicazioni in esecuzione sulle istanze Amazon EC2.

Configurazione avanzata per il tuo AWS SDK for .NET progetto

Gli argomenti di questa sezione contengono informazioni su attività e metodi di configurazione aggiuntivi che potrebbero interessarti.

Argomenti

- [Utilizzo di AWSSDK .extensions.netcore.setup e dell'interfaccia IConfiguration](#)
- [Configurazione di altri parametri dell'applicazione](#)
- [Riferimento per i file di configurazione per AWS SDK for .NET](#)

Utilizzo di AWSSDK .extensions.netcore.setup e dell'interfaccia IConfiguration

(Questo argomento era precedentemente intitolato «Configurazione di con.NET Core») AWS SDK for .NET

Una delle principali modifiche apportate a .NET Core è la rimozione degli standard `ConfigurationManager` e dei `web.config` file utilizzati con le `app.config` applicazioni .NET Framework e ASP.NET.

La configurazione in .NET Core si basa su coppie chiave-valore stabilite dai provider di configurazione. I provider di configurazione leggono i dati di configurazione nelle coppie chiave-valore da una serie di origini di configurazione, inclusi gli argomenti della riga comando, file directory, variabili d'ambiente e file di impostazioni.

Note

Per ulteriori informazioni, consulta [Configuration in ASP.NET Core](#).

Per semplificare l'utilizzo AWS SDK for .NET con .NET Core, è possibile utilizzare il pacchetto [AWSSDK.extensions.NETCORE.setup](#). NuGet Come molte librerie.NET Core, aggiunge metodi di estensione all'IConfigurationinterfaccia per semplificare la configurazione. AWS

Utilizzando .extensions.netcore.setup AWSSDK

Si supponga di creare un'applicazione ASP.NET Core Model-View-Controller (MVC), che può essere eseguita con il modello di applicazione Web ASP.NET Core in Visual Studio o eseguendo in.NET Core CLI. `dotnet new mvc ...` Quando si crea un'applicazione di questo tipo, il costruttore di `Startup.cs` gestisce la configurazione leggendo varie fonti di input da provider di configurazione come `appsettings.json`

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

Per utilizzare l'Configurationoggetto per ottenere le AWSopzioni, aggiungete prima il `AWSSDK.Extensions.NETCore.Setup` NuGet pacchetto. Quindi, aggiungete le vostre opzioni al file di configurazione come descritto di seguito.

Nota che uno dei file aggiunti al tuo progetto è `appsettings.Development.json`. Ciò corrisponde a un `EnvironmentName` set di `Development`. Durante lo sviluppo, si inserisce la configurazione in questo file, che viene letto solo durante i test locali. Quando si distribuisce un'istanza Amazon EC2 **EnvironmentName** impostata su `Production`, questo file viene ignorato e torna alle credenziali e AWS SDK for .NET alla regione IAM configurate per l'istanza Amazon EC2.

Le seguenti impostazioni di configurazione mostrano esempi dei valori che puoi aggiungere al `appsettings.Development.json` file nel tuo progetto per fornire le impostazioni. AWS

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

Per accedere a un'impostazione in un file CSHTML, utilizzate la direttiva. `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>
```

Per accedere alle AWS opzioni impostate nel file dal codice, chiamate il metodo di `GetAWSSOptions` estensione aggiunto a `IConfiguration`

Per costruire un client di servizio da queste opzioni, chiama il codice `CreateServiceClient`. L'esempio seguente mostra come creare un client di servizio Amazon S3. (Assicurati di aggiungere il [AWSSDK NuGet pacchetto.S3](#) al tuo progetto.)

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

È inoltre possibile creare più client di servizio con impostazioni incompatibili utilizzando più voci nel `appsettings.Development.json` file, come illustrato negli esempi seguenti, in cui la configurazione per `service1` include la `us-west-2` regione e la configurazione per include l'URL speciale dell'endpoint. `service2`

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

È possibile ottenere le opzioni per un servizio specifico utilizzando la voce nel file JSON. Ad esempio, per ottenere le impostazioni `service1` utilizzate quanto segue.

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

Valori consentiti nel file appsettings

I seguenti valori di configurazione delle app possono essere impostate nel file `appsettings.Development.json`. I nomi dei campi devono utilizzare la maiuscola mostrata. Per i dettagli su queste impostazioni, consultate la [AWS.Runtime.ClientConfig](#) classe.

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

Iniezione delle dipendenze di ASP.NET Core

Il NuGet pacchetto `AWSSDK.extensions.netcore.setup` si integra anche con un nuovo sistema di iniezione delle dipendenze in ASP.NET Core. Il `ConfigureServices` metodo nella classe dell'applicazione è quello in cui vengono aggiunti i servizi MVC. `Startup` Se l'applicazione utilizza Entity Framework, è qui che viene inizializzata.

```
public void ConfigureServices(IServiceCollection services)
```

```
{
    // Add framework services.
    services.AddMvc();
}
```

Note

Le informazioni sull'iniezione delle dipendenze in .NET Core sono disponibili nel sito di [documentazione di .NET Core](#).

Il `AWSSDK.Extensions.NETCore.Setup` NuGet pacchetto aggiunge nuovi metodi di estensione `IServiceCollection` che è possibile utilizzare per aggiungere AWS servizi all'iniezione di dipendenze. Il codice seguente mostra come aggiungere le AWS opzioni da cui vengono lette `IConfiguration` per aggiungere Amazon S3 e DynamoDB all'elenco dei servizi. (Assicurati di aggiungere i [AWSSDKpacchetti.S3](#) e [AWSSDK NuGet .DynamoDBv2](#) al tuo progetto.)

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

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

Ora, se i controller MVC utilizzano `IAmazonS3` o `IAmazonDynamoDB` come parametri nei costruttori, il sistema di inserimento delle dipendenze trasferisce questi servizi.

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

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

    ...
}
```

```
}
```

Configurazione di altri parametri dell'applicazione

Note

Le informazioni in questo argomento sono specifiche per i progetti basati su .NET Framework. I Web.config file App.config and non sono presenti per impostazione predefinita nei progetti basati su .NET Core.

Apri per visualizzare il contenuto di .NET Framework

Esistono diversi parametri dell'applicazione che è possibile configurare:

- [AWSLogging](#)
- [AWSLogMetrics](#)
- [AWSRegion](#)
- [AWSResponseLogging](#)
- [AWS.DynamoDBContext.TableNamePrefix](#)
- [AWS.S3.UseSignatureVersion4](#)
- [AWSEndpointDefinition](#)
- [AWSEndpoint generati dal servizio](#)

Questi parametri possono essere configurati nel file App.config o Web.config dell'applicazione. Anche se puoi configurarli anche con l'API di AWS SDK for .NET, consigliamo di utilizzare il file .config dell'applicazione. Entrambi gli approcci sono descritti in questo articolo.

Per ulteriori informazioni sull'uso dell'<aws>elemento, come descritto più avanti in questo argomento, vedere [Configuration Files Reference for .NET](#) AWS SDK for .NET

AWSLogging

Configura se e come l'SDK deve registrare gli eventi. Ad esempio, l'approccio consigliato prevede di utilizzare l'elemento <logging>, che è un elemento figlio di <aws>:

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

```
</aws>
```

In alternativa:

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

I valori possibili sono:

None

Disattiva la registrazione degli eventi . Questa è l'impostazione predefinita.

log4net

Accedi utilizzando log4net.

SystemDiagnostics

Accedi utilizzando la classe `System.Diagnostics`.

Puoi impostare più valori per l'attributo `logTo`, separandoli con virgole. L'esempio seguente imposta la registrazione sia di `log4net` che di `System.Diagnostics` nel file `.config`:

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

In alternativa:

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

[In alternativa, utilizzando l'AWS SDK for .NET API, combinate i valori dell'enum `LoggingOptions` e impostate la proprietà `.Logging`: \[AWSConfigs\]\(#\)](#)

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

Le modifiche a questa impostazione vengono applicate solo alle nuove istanze del client AWS.

AWSLogMetrics

Specifica se l'SDK debba registrare i parametri delle prestazioni. Per impostare la configurazione della registrazione dei parametri nel file `.config`, imposta il valore dell'attributo `logMetrics` nell'elemento `<logging>`, che è un elemento figlio dell'elemento `<aws>`:

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

In alternativa, imposta la chiave `AWSLogMetrics` nella sezione `<appSettings>`:

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

[In alternativa, per impostare la registrazione delle metriche con l'API, imposta `AWS SDK for .NET AWSConfigs LogMetrics` proprietà:](#)

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

Questa impostazione configura la proprietà `LogMetrics` predefinita per tutti i client e le configurazioni. Le modifiche a questa impostazione vengono applicate solo alle nuove istanze del client AWS.

AWSRegion

Configura l'AWSarea predefinita per i client che non hanno specificato esplicitamente una regione. Per impostare la regione nel file `.config`, l'approccio consigliato prevede di impostare il valore dell'attributo `region` nell'elemento `aws`:

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

In alternativa, imposta la chiave `AWSRegion` nella sezione `<appSettings>`:

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

[In alternativa, per impostare la regione con l'AWS SDK for .NETAPI, imposta il `AWSConfigs AWSRegion` proprietà:](#)

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

Per ulteriori informazioni sulla creazione di un AWS client per una regione specifica, vedere [Selezione AWS della regione](#). Le modifiche a questa impostazione vengono applicate solo alle nuove istanze del client AWS.

AWSResponseLogging

Configura quando l'SDK deve registrare le risposte di servizio. I valori possibili sono:

Never

Non registrare mai le risposte di servizio. Questa è l'impostazione predefinita.

Always

Registra sempre le risposte di servizio.

OnError

Registra le risposte di servizio solo quando si verifica un errore.

Per impostare la configurazione della registrazione dei servizi nel file `.config`, l'approccio consigliato prevede di impostare il valore dell'attributo `logResponses` nell'elemento `<logging>`, che è un elemento figlio dell'elemento `<aws>`:

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

In alternativa, imposta la chiave `AWSResponseLogging` nella sezione `<appSettings>`:

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

[In alternativa, per impostare la registrazione del servizio con l'AWS SDK for .NETAPI, imposta il `AWSConfigs.ResponseLogging` proprietà su uno dei valori dell'`ResponseLoggingOption` enumerazione:](#)

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

Le modifiche a questa impostazione vengono applicate immediatamente.

AWS.DynamoDBContext.TableNamePrefix

Configura l'elemento `TableNamePrefix` predefinito utilizzato da `DynamoDBContext` se non viene configurato manualmente.

Per impostare il prefisso del nome della tabella nel file `.config`, l'approccio consigliato prevede di impostare il valore dell'attributo `tableNamePrefix` nell'elemento `<dynamoDBContext>`, che è un elemento figlio dell'elemento `<dynamoDB>`, a sua volta elemento figlio dell'elemento `<aws>`:

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

In alternativa, imposta la chiave `AWS.DynamoDBContext.TableNamePrefix` nella sezione `<appSettings>`:

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

In alternativa, per impostare il prefisso del nome della tabella con l'AWS SDK for .NET API, imposta la [AWSConfig proprietà DynamoDB Context TableNamePrefix](#):

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

Le modifiche a questa impostazione vengono applicate solo alle istanze di recente costruzione di `DynamoDBContextConfig` e `DynamoDBContext`.

AWS.S3.UseSignatureVersion4

Configura se il client Amazon S3 deve utilizzare o meno la firma con la versione 4 della firma con le richieste.

Per impostare la firma della versione 4 della firma per Amazon S3 nel `.config` file, l'approccio consigliato consiste nell'impostare l'`useSignatureVersion4` attributo dell'`<s3>` elemento, che è un elemento figlio dell'`<aws>` elemento:

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

In alternativa, imposta la `AWS.S3.UseSignatureVersion4` chiave su `true` nella `<appSettings>` sezione:

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

In alternativa, per impostare la firma della versione 4 della firma con l'AWS SDK for .NET API, imposta la proprietà [AWSConfigs.S3 UseSignatureVersion 4](#) su: `true`


```
AWSSignatureVersion4 = true;
```

Per impostazione predefinita, questa impostazione è `false`, ma in alcuni casi o regioni la firma Signature Version 4 può essere utilizzata per impostazione predefinita. Quando l'impostazione è `true`, la firma Signature Version 4 verrà utilizzata per tutte le richieste. Le modifiche a questa impostazione hanno effetto solo per le nuove istanze client Amazon S3.

AWSEndpointDefinition

Configura se l'SDK debba utilizzare un file di configurazione personalizzato che definisce le regioni e gli endpoint.

Per impostare il file di definizione degli endpoint nel file `.config`, consigliamo di impostare il valore dell'attributo `endpointDefinition` nell'elemento `<aws>`.

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

In alternativa, puoi impostare la `AWSEndpointDefinition` chiave nella `<appSettings>` sezione:

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

[In alternativa, per impostare il file di definizione dell'endpoint con l'AWS SDK for .NET API, imposta il `AWSSignatureVersion4` proprietà:](#)

```
AWSSignatureVersion4 = @"c:\config\endpoints.json";
```

Se non viene fornito il nome del file, il file di configurazione personalizzato non verrà utilizzato. Le modifiche a questa impostazione vengono applicate solo alle nuove istanze del client AWS. Il file `endpoint.json` è disponibile da <https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json>

AWSEndpoint generati dal servizio

Alcuni AWS servizi generano i propri endpoint invece di utilizzare un endpoint regionale. I client per questi servizi utilizzano un URL specifico per il servizio e le tue risorse. Due esempi di questi servizi sono Amazon CloudSearch e AWS IoT. Gli esempi seguenti mostrano come ottenere gli endpoint per questi servizi.

Esempio di Amazon CloudSearch Endpoints

Il CloudSearch client Amazon viene utilizzato per accedere al servizio di CloudSearch configurazione Amazon. Utilizzi il servizio di CloudSearch configurazione Amazon per creare, configurare e gestire i domini di ricerca. Per creare un dominio di ricerca, crea un [CreateDomainRequest](#) oggetto e fornisci la DomainName proprietà. Crea un [AmazonCloudSearchClient](#) oggetto utilizzando l'oggetto di richiesta. Chiama il metodo [CreateDomain](#). L'[CreateDomainResponse](#) oggetto restituito dalla chiamata contiene una DomainStatus proprietà che ha sia l'estremità che DocService l'SearchService estremità. Crea un [AmazonCloudSearchDomainConfig](#) oggetto e utilizzatelo per inizializzare DocService e inizializzare SearchService le istanze della classe. [AmazonCloudSearchDomainClient](#)

```
// 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);
}

```

Esempio di endpoint AWS IoT

Per ottenere l'endpoint per AWS IoT, crea un oggetto [AmazonIoTClient](#) e chiama il metodo.

[DescribeEndPoint](#) L'[DescribeEndPointResponse](#) oggetto restituito contiene il `EndpointAddress`

Create un [AmazonIoTDataConfig](#) oggetto, impostate la `ServiceURL` proprietà e utilizzate l'oggetto per creare un'istanza della [AmazonIoTDataClient](#) classe.

```

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");  
}
```

Riferimento per i file di configurazione per AWS SDK for .NET

Note

Le informazioni in questo argomento sono specifiche per i progetti basati su .NET Framework. I `Web.config` file `App.config` and non sono presenti per impostazione predefinita nei progetti basati su .NET Core.

Apri per visualizzare il contenuto di .NET Framework

Puoi utilizzare un progetto `App.config` o un `Web.config` file .NET per specificare AWS impostazioni, come AWS credenziali, opzioni di registrazione, endpoint di AWS servizio e AWS regioni, nonché alcune impostazioni per AWS servizi come Amazon DynamoDB, Amazon EC2 e Amazon S3. Le informazioni seguenti descrivono come formattare correttamente un file `App.config` o `Web.config` per specificare questi tipi di impostazioni.

Note

Sebbene sia possibile continuare a utilizzare l'<appSettings> elemento in un `Web.config` file `App.config` or per specificare AWS le impostazioni, si consiglia di utilizzare <aws> gli elementi <configSections> and come descritto più avanti in questo argomento. Per ulteriori informazioni sull'<appSettings> elemento, consultate gli esempi di <appSettings> elementi in [Configurazione AWS SDK for .NET dell'applicazione](#).

Note

Sebbene sia possibile continuare a utilizzare le seguenti proprietà di [AWSConfigs](#) classe in un file di codice per specificare AWS le impostazioni, le seguenti proprietà sono obsolete e potrebbero non essere supportate nelle versioni future:

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

In generale, invece di utilizzare le proprietà di `AWSConfigs` classe in un file di codice per specificare AWS le impostazioni, si consiglia di utilizzare gli `<aws>` elementi `<configSections>` and in un `Web.config` file `App.config` or per specificare AWS le impostazioni, come descritto più avanti in questo argomento. Per ulteriori informazioni sulle proprietà precedenti, consultate gli esempi di `AWSConfigs` codice in [Configurazione dell'applicazione AWS SDK for .NET](#).

Argomenti

- [Dichiarazione di una sezione delle impostazioni AWS](#)
- [Elementi consentiti](#)
- [Riferimento per gli elementi elemento](#)

Dichiarazione di una sezione delle impostazioni AWS

AWSLe impostazioni vengono specificate in un `Web.config` file `App.config` or dall'interno dell'`<aws>`elemento. Prima di iniziare a utilizzare l'elemento `<aws>`, devi creare un elemento `<section>`, ossia un elemento figlio dell'elemento `<configSections>`, e impostare l'attributo `name` su `aws` e l'attributo `type` su `Amazon.AWSSection, AWSSDK.Core`, come nell'esempio seguente:

```
<?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>
```

Visual Studio Editor non fornisce il completamento automatico del codice (IntelliSense) per l'<aws>elemento o i relativi elementi secondari.

Se serve assistenza per creare correttamente una versione formattata dell'elemento <aws>, chiama il metodo `Amazon.AWSConfigs.GenerateConfigTemplate`. Viene restituita una versione canonica dell'elemento <aws> sotto forma di stringa formattata che puoi adattare alle tue esigenze. Le sezioni seguenti descrivono gli attributi dell'elemento <aws> e i relativi elementi figlio.

Elementi consentiti

Di seguito è riportato un elenco delle relazioni logiche tra gli elementi consentiti in una sezione AWS delle impostazioni. Puoi generare la versione più recente di questo elenco chiamando il metodo `Amazon.AWSConfigs.GenerateConfigTemplate`, che fornisce una versione canonica dell'elemento <aws> sotto forma di stringa che puoi adattare alle tue esigenze.

```
<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>
```

Riferimento per gli elementi elemento

Di seguito è riportato un elenco degli elementi consentiti in una sezione AWS delle impostazioni. Per ogni elemento sono elencati gli attributi e gli elementi padre-figlio consentiti.

Argomenti

- [alias](#)
- [aws](#)
- [dynamoDB](#)
- [dynamoDBContext](#)
- [ec2](#)
- [logging](#)
- [map](#)
- [property](#)
- [proxy](#)
- [s3](#)

alias

L'elemento `<alias>` rappresenta un singolo elemento in una raccolta di una o più mappature da from-table a to-table che specifica una tabella diversa rispetto a quella configurata per un tipo. Questo elemento viene mappato a un'istanza della classe `Amazon.Util.TableAlias` dalla proprietà `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` in AWS SDK for .NET. La ri-mappatura viene eseguita prima di applicare un prefisso al nome della tabella.

Questo elemento può includere gli attributi seguenti:

fromTable

La parte from-table della mappatura da from-table a to-table. Questo attributo viene mappato alla proprietà `Amazon.Util.TableAlias.FromTable` in AWS SDK for .NET.

toTable

La parte to-table della mappatura da from-table a to-table. Questo attributo viene mappato alla proprietà `Amazon.Util.TableAlias.ToTable` in AWS SDK for .NET.

Il padre dell'elemento `<alias>` è l'elemento `<tableAliases>`.

L'elemento `<alias>` non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento `<alias>` in uso:

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

aws

L'`<aws>`elemento rappresenta l'elemento più in alto in una sezione AWS delle impostazioni. Questo elemento può includere gli attributi seguenti:

endpointDefinition

Il percorso assoluto di un file di configurazione personalizzato che definisce le AWS regioni e gli endpoint da utilizzare. Questo attributo viene mappato alla proprietà `Amazon.AWSConfigs.EndpointDefinition` in AWS SDK for .NET.

profileName

Il nome del profilo per AWS le credenziali archiviate che verranno utilizzate per effettuare chiamate di servizio. Questo attributo viene mappato alla proprietà `Amazon.AWSConfigs.AWSProfileName` in AWS SDK for .NET.

profilesLocation

Il percorso assoluto della posizione del file delle credenziali condiviso con altri AWS SDK. Per impostazione predefinita, il file delle credenziali viene archiviato nella directory `.aws` nella directory principale dell'utente corrente. Questo attributo viene mappato alla proprietà `Amazon.AWSConfigs.AWSProfilesLocation` in AWS SDK for .NET.

region

L'ID di AWS regione predefinito per i client che non hanno specificato esplicitamente una regione. Questo attributo viene mappato alla proprietà `Amazon.AWSConfigs.AWSRegion` in AWS SDK for .NET.

L'elemento `<aws>` non ha elementi padre.

L'elemento `<aws>` può contenere i seguenti elementi figlio:

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

Di seguito è illustrato un esempio dell'elemento `<aws>` in uso:

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

dynamoDB

L'elemento <dynamoDB> rappresenta una raccolta di impostazioni per Amazon DynamoDB. Questo elemento può includere l'attributo `conversionSchema`, che rappresenta la versione da utilizzare per la conversione di oggetti .NET e DynamoDB. I valori consentiti includono V1 e V2. Questo attributo viene mappato alla classe `Amazon.DynamoDBv2.DynamoDBEntryConversion` in AWS SDK for .NET. Per ulteriori informazioni, vedi [DynamoDB Series - Schemi di conversione](#).

Il padre dell'elemento <dynamoDB> è l'elemento <aws>.

L'elemento <dynamoDB> può contenere l'elemento figlio <dynamoDBContext>.

Di seguito è illustrato un esempio dell'elemento <dynamoDB> in uso:

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

dynamoDBContext

L'elemento <dynamoDBContext> rappresenta una raccolta di impostazioni specifiche per il contesto di Amazon DynamoDB. Questo elemento può includere l'attributo `tableNamePrefix`, che rappresenta il prefisso del nome di tabella predefinito che il contesto DynamoDB utilizzerà se non è configurato manualmente. Questo attributo viene mappato alla proprietà `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` dalla proprietà `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` in AWS SDK for .NET. Per ulteriori informazioni, vedi [Miglioramenti all'SDK DynamoDB](#).

Il padre dell'elemento <dynamoDBContext> è l'elemento <dynamoDB>.

L'elemento <dynamoDBContext> può contenere i seguenti elementi figlio:

- <alias> (una o più istanze)
- <map> (una o più istanze)

Di seguito è illustrato un esempio dell'elemento <dynamoDBContext> in uso:

```
<dynamoDBContext
```

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

ec2

L'elemento `<ec2>` rappresenta una raccolta di impostazioni di Amazon EC2. Questo elemento può includere l'attributo `useSignatureVersion4`, che specifica se la firma della versione 4 della firma verrà utilizzata per tutte le richieste (`true`) o se la firma della versione 4 della firma non verrà utilizzata per tutte le richieste (`false`, l'impostazione predefinita). Questo attributo viene mappato alla proprietà `Amazon.Util.EC2Config.UseSignatureVersion4` dalla proprietà `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` in AWS SDK for .NET.

Il padre dell'elemento `<ec2>` è l'elemento.

L'elemento `<ec2>` non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento `<ec2>` in uso:

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

logging

L'elemento `<logging>` rappresenta una raccolta di impostazioni per la registrazione delle risposte e la registrazione dei parametri delle prestazioni. Questo elemento può includere gli attributi seguenti:

logMetrics

Indica se i parametri delle prestazioni sono registrati per tutti i client e le configurazioni (`true`) o meno (`false`). Questo attributo viene mappato alla proprietà `Amazon.Util.LoggingConfig.LogMetrics` dalla proprietà `Amazon.AWSConfigs.LoggingConfig.LogMetrics` in AWS SDK for .NET.

logMetricsCustomFormatter

Il tipo di dati e il nome dell'assembly di un formattatore personalizzato per i parametri di registrazione. Questo attributo viene mappato alla proprietà `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` dalla proprietà `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` in AWS SDK for .NET.

logMetricsFormat

Il formato in cui vengono visualizzati i parametri di registrazione (mappato alla proprietà `Amazon.Util.LoggingConfig.LogMetricsFormat` dalla proprietà `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` in AWS SDK for .NET).

I valori autorizzati includono:

JSON

Utilizza il formato JSON.

Standard

Utilizza formato predefinito.

logResponses

Quando registrare le risposte del servizio (mappato alla proprietà `Amazon.Util.LoggingConfig.LogResponses` dalla proprietà `Amazon.AWSConfigs.LoggingConfig.LogResponses` in AWS SDK for .NET).

I valori autorizzati includono:

Always

Registra sempre le risposte di servizio.

Never

Non registrare mai le risposte di servizio.

OnError

Registra le risposte del servizio solo quando ci sono errori.

logTo

Dove eseguire la registrazione (mappato alla proprietà `LogTo` dalla proprietà `Amazon.AWSConfigs.LoggingConfig.LogTo` in AWS SDK for .NET).

I valori consentiti includono uno o più:

Log4Net

Accedi a `log4net`.

None

Disabilita la registrazione.

SystemDiagnostics

Accedi a System.Diagnostics.

Il padre dell'elemento <logging> è l'elemento <aws>.

L'elemento <logging> non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento <logging> in uso:

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

map

L'<map>elemento rappresenta un singolo elemento in una raccolta di type-to-table mappature dai tipi.NET alle tabelle DynamoDB (esegue il mapping a un'istanza della TypeMapping classe dalla proprietà in). Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings AWS SDK for .NET Per ulteriori informazioni, vedi [Miglioramenti all'SDK DynamoDB](#).

Questo elemento può includere gli attributi seguenti:

targetTable

La tabella DynamoDB a cui si applica la mappatura. Questo attributo viene mappato alla proprietà Amazon.Util.TypeMapping.TargetTable in AWS SDK for .NET.

type

Il tipo e il nome dell'assembly a cui si applica la mappatura. Questo attributo viene mappato alla proprietà Amazon.Util.TypeMapping.Type in AWS SDK for .NET.

Il padre dell'elemento <map> è l'elemento <dynamoDBContext>.

L'elemento <map> può includere una o più istanze dell'elemento figlio <property>.

Di seguito è illustrato un esempio dell'elemento <map> in uso:

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

property

L'elemento `<property>` rappresenta una proprietà di DynamoDB. (Questo elemento è mappato a un'istanza di `Amazon.Util.PropertyConfig` [classe dal `AddProperty` metodo in AWS SDK for .NET](#)) [Per ulteriori informazioni, consulta Miglioramenti all'SDK di DynamoDB e agli attributi di DynamoDB.](#)

Questo elemento può includere gli attributi seguenti:

attribute

Il nome di un attributo per la proprietà, ad esempio il nome di una chiave di intervallo. Questo attributo viene mappato alla proprietà `Amazon.Util.PropertyConfig.Attribute` in AWS SDK for .NET.

converter

Il tipo di convertitore da utilizzare per questa proprietà. Questo attributo viene mappato alla proprietà `Amazon.Util.PropertyConfig.Converter` in AWS SDK for .NET.

ignore

Indica se proprietà associata debba essere ignorata (`true`) o meno (`false`). Questo attributo viene mappato alla proprietà `Amazon.Util.PropertyConfig.Ignore` in AWS SDK for .NET.

name

Il nome della proprietà. Questo attributo viene mappato alla proprietà `Amazon.Util.PropertyConfig.Name` in AWS SDK for .NET.

version

Indica se la proprietà debba archiviare il numero di versione dell'elemento (`true`) o meno (`false`). Questo attributo viene mappato alla proprietà `Amazon.Util.PropertyConfig.Version` in AWS SDK for .NET.

Il padre dell'elemento `<property>` è l'elemento `<map>`.

L'elemento `<property>` non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento `<property>` in uso:

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

proxy

L'elemento `<proxy>` rappresenta le impostazioni per la configurazione di un proxy da utilizzare con AWS SDK for .NET. Questo elemento può includere gli attributi seguenti:

host

L'indirizzo IP o il nome host del server proxy. Questo attributo viene mappato alla proprietà `Amazon.Util.ProxyConfig.Host` dalla proprietà `Amazon.AWSConfigs.ProxyConfig.Host` in AWS SDK for .NET.

password

La password per eseguire l'autenticazione con il server proxy. Questo attributo viene mappato alla proprietà `Amazon.Util.ProxyConfig.Password` dalla proprietà `Amazon.AWSConfigs.ProxyConfig.Password` in AWS SDK for .NET.

port

Il numero di porta del proxy. Questo attributo viene mappato alla proprietà `Amazon.Util.ProxyConfig.Port` dalla proprietà `Amazon.AWSConfigs.ProxyConfig.Port` in AWS SDK for .NET.

username

Il nome utente per eseguire l'autenticazione con il server proxy. Questo attributo viene mappato alla proprietà `Amazon.Util.ProxyConfig.Username` dalla proprietà `Amazon.AWSConfigs.ProxyConfig.Username` in AWS SDK for .NET.

Il padre dell'elemento `<proxy>` è l'elemento `<aws>`.

L'elemento `<proxy>` non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento `<proxy>` in uso:

```
<proxy
```

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

s3

L'elemento `<s3>` rappresenta una raccolta di impostazioni di Amazon S3. Questo elemento può includere l'attributo `useSignatureVersion4`, che specifica se la firma della versione 4 della firma verrà utilizzata per tutte le richieste (`true`) o se la firma della versione 4 della firma non verrà utilizzata per tutte le richieste (`false`, l'impostazione predefinita). Questo attributo viene mappato alla proprietà `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` in AWS SDK for .NET.

Il padre dell'elemento `<s3>` è l'elemento `<aws>`.

L'elemento `<s3>` non contiene elementi figlio.

Di seguito è illustrato un esempio dell'elemento `<s3>` in uso:

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

Uso delle credenziali legacy

Negli argomenti di questa sezione vengono fornite informazioni sull'utilizzo di credenziali a lungo termine o a breve termine senza utilizzare AWS IAM Identity Center.

Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

Note

Le informazioni contenute in questo argomento riguardano le circostanze in cui è necessario ottenere e gestire manualmente le credenziali a breve o lungo termine. Per ulteriori informazioni sulle credenziali a breve e lungo termine, consulta l'argomento relativo agli [altri modi di autenticazione disponibili](#) nella Guida di riferimento per SDK e strumenti AWS.

Per le best practice relative alla sicurezza, usa AWS IAM Identity Center, come descritto in [Configura l'autenticazione SDK](#).

Avvertenze e linee guida importanti per le credenziali

Avvertenze per le credenziali

- NON utilizzare le credenziali root dell'account per accedere alle risorse AWS. Queste credenziali forniscono un accesso illimitato all'account e sono difficili da revocare.
- NON inserire chiavi di accesso letterali o informazioni sulle credenziali nei file dell'applicazione. In caso contrario, rischi di esporre accidentalmente le credenziali se, per esempio, carichi il progetto in repository pubblici.
- NON includete file che contengono credenziali nell'area del progetto.
- Si noti che qualsiasi credenziale archiviata nel file `AWS credentials` condiviso viene archiviata in testo normale.

Linee guida aggiuntive per la gestione sicura delle credenziali

[Per una discussione generale su come gestire in modo sicuro le credenziali, consulta AWS le credenziali di AWSsicurezza nella sezione Credenziali di sicurezza e le best practice Riferimenti generali di AWS e i casi d'uso di sicurezza nella IAM User Guide](#). Oltre a queste discussioni, considera quanto segue:

- Crea utenti aggiuntivi, come gli utenti in IAM Identity Center, e utilizza le loro credenziali invece di utilizzare le credenziali dell'utente root AWS. Le credenziali per altri utenti possono essere revocate, se necessario, o sono temporanee per natura. Inoltre, puoi applicare una policy a ciascun utente per consentire l'accesso solo a determinate risorse e azioni e quindi adottare autorizzazioni basate sul con privilegio minimo.
- Usa [ruoli IAM](#) per le attività di Amazon Elastic Container Service (Amazon ECS).
- Usa [ruoli IAM](#) per le applicazioni in esecuzione sulle istanze Amazon EC2.
- Utilizzate [credenziali temporanee](#) o variabili di ambiente per le applicazioni disponibili per gli utenti esterni all'organizzazione.

Argomenti

- [Utilizzo del file di AWS credenziali condivise](#)
- [Utilizzo dell'SDK Store \(solo Windows\)](#)

Utilizzo del file di AWS credenziali condivise

(Assicurati di leggere gli [avvisi e le linee guida importanti per le credenziali](#).)

Un modo per fornire le credenziali per le applicazioni consiste nel creare profili nel file delle AWS credenziali condivise e quindi archiviare le credenziali in tali profili. Questo file può essere utilizzato dagli altri SDK. AWS [Può essere utilizzato anche da AWS CLI/AWS Tools for Windows PowerShell, the e dai AWS toolkit per Visual Studio e VS JetBrainsCode.](#)

Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

Note

Le informazioni contenute in questo argomento riguardano le circostanze in cui è necessario ottenere e gestire manualmente le credenziali a breve o lungo termine. Per ulteriori informazioni sulle credenziali a breve e lungo termine, consulta l'argomento relativo agli [altri modi di autenticazione disponibili](#) nella Guida di riferimento per SDK e strumenti AWS. Per le best practice relative alla sicurezza, usa AWS IAM Identity Center, come descritto in [Configura l'autenticazione SDK](#).

Informazioni generali

Per impostazione predefinita, il file AWS delle credenziali condivise si trova nella `.aws` directory all'interno della home directory ed è denominato, `credentials` ovvero `~/.aws/credentials` (Linux o macOS) `%USERPROFILE%\.aws\credentials` o (Windows). Per informazioni sulle posizioni alternative, consulta [Posizione dei file condivisi nella Guida](#) di riferimento agli [AWSSDK e agli strumenti](#). Consultare anche [Accesso a credenziali e profili in un'applicazione](#).

Il file delle AWS credenziali condivise è un file di testo semplice e segue un determinato formato. Per informazioni sul formato dei file di AWS credenziali, consulta [Formato del file delle credenziali nella Guida di riferimento agli AWSSDK e agli strumenti](#).

Puoi gestire i profili nel file delle AWS credenziali condivise in diversi modi.

- Utilizza qualsiasi editor di testo per creare e aggiornare il file di AWS credenziali condivise.
- Usa [Amazon.Runtime.CredentialManagement](#) namespace dell'AWS SDK for .NET API, come illustrato più avanti in questo argomento.
- [Usa i comandi e le procedure per AWS Tools for PowerShell e i AWS toolkit per Visual Studio e VS Code JetBrains.](#)
- Usa [AWS CLI](#) i comandi, ad esempio `eaws configure set aws_access_key_id. aws configure set aws_secret_access_key`

Esempi di gestione dei profili

Le seguenti sezioni mostrano esempi di profili nel file di AWS credenziali condivise. Alcuni esempi mostrano il risultato, che può essere ottenuto tramite uno qualsiasi dei metodi di gestione delle credenziali descritti in precedenza. Altri esempi mostrano come utilizzare un metodo particolare.

Il profilo predefinito

Il file AWS delle credenziali condivise avrà quasi sempre un profilo denominato default. È qui AWS SDK for .NET che cerca le credenziali se non sono definiti altri profili.

Il [default] profilo ha in genere un aspetto simile al seguente.

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

Crea un profilo a livello di codice

Questo esempio mostra come creare un profilo e salvarlo nel file delle AWS credenziali condivise a livello di codice. [Utilizza le seguenti classi di Amazon.Runtime.CredentialManagement](#) namespace:, e. [CredentialProfileOptionsCredentialProfileSharedCredentialsFile](#)

```
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

Un codice come questo in genere non dovrebbe essere presente nell'applicazione. Se lo includete nella vostra applicazione, prendete le precauzioni appropriate per assicurarvi che le chiavi in chiaro non siano visibili nel codice, sulla rete e nemmeno nella memoria del computer.

Di seguito è riportato il profilo creato da questo esempio.

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

Aggiorna un profilo esistente a livello di codice

Questo esempio mostra come aggiornare a livello di codice il profilo creato in precedenza.

[Utilizza le seguenti classi di Amazon.Runtime.CredentialManagement](#) namespace: e.

[CredentialProfileSharedCredentialsFile](#) Utilizza anche la [RegionEndpoint](#) classe dello spazio dei nomi [Amazon](#).

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

```
...  
AddRegion("my_new_profile", RegionEndpoint.USWest2);  
...  
void AddRegion(string profileName, RegionEndpoint region)  
{  
    var sharedFile = new SharedCredentialsFile();  
    CredentialProfile profile;  
    if (sharedFile.TryGetProfile(profileName, out profile))  
    {  
        profile.Region = region;  
        sharedFile.RegisterProfile(profile);  
    }  
}
```

Di seguito è riportato il profilo aggiornato.

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

Note

È inoltre possibile impostare la AWS regione in altre posizioni e utilizzando altri metodi. Per ulteriori informazioni, consulta [Configura la AWS regione](#).

Utilizzo dell'SDK Store (solo Windows)

(Assicurati di leggere le [avvertenze e le linee guida importanti](#)).

In Windows, l'SDK Store è un altro posto per creare profili e archiviare credenziali crittografate per l'applicazione. AWS SDK for .NET Si trova in. %USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json Puoi utilizzare l'SDK Store durante lo sviluppo come alternativa al file di [AWS credenziali condivise](#).

Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

Note

Le informazioni contenute in questo argomento riguardano le circostanze in cui è necessario ottenere e gestire manualmente le credenziali a breve o lungo termine. Per ulteriori informazioni sulle credenziali a breve e lungo termine, consulta l'argomento relativo agli [altri modi di autenticazione disponibili](#) nella Guida di riferimento per SDK e strumenti AWS.

Per le best practice relative alla sicurezza, usa AWS IAM Identity Center, come descritto in [Configura l'autenticazione SDK](#).

Informazioni generali

L'SDK Store offre i seguenti vantaggi:

- Le credenziali nell'SDK Store sono crittografate e l'SDK Store si trova nella home directory dell'utente. Ciò limita il rischio di esporre accidentalmente le proprie credenziali.
- L'SDK Store fornisce anche le credenziali per e per. [AWS Tools for Windows PowerShell](#)
[AWS Toolkit for Visual Studio](#)

I profili SDK Store sono specifici per un particolare utente su un determinato host. Non puoi copiarli in altri host o per altri utenti. Ciò significa che non puoi riutilizzare i profili SDK Store presenti sulla tua macchina di sviluppo per altri host o macchine per sviluppatori. Significa anche che non puoi utilizzare i profili SDK Store nelle applicazioni di produzione.

Puoi gestire i profili nell'SDK Store nei seguenti modi:

- Utilizzate l'interfaccia utente grafica (GUI) in. [AWS Toolkit for Visual Studio](#)
- [Usa Amazon.Runtime.CredentialManagement](#) namespace dell'AWS SDK for .NET API, come illustrato più avanti in questo argomento.

- Utilizzate i comandi di [AWS Tools for Windows PowerShell](#); ad esempio, `Set-AWSCredential`.
`Remove-AWSCredentialProfile`

Esempi di gestione dei profili

I seguenti esempi mostrano come creare e aggiornare a livello di codice un profilo nell'SDK Store.

Crea un profilo a livello di codice

Questo esempio mostra come creare un profilo e salvarlo nell'SDK Store a livello di codice. [Utilizza le seguenti classi di Amazon.Runtime.CredentialManagement namespace: CredentialProfileOptions, e NetSDK.CredentialProfileCredentialsFile](#)

```
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

Un codice come questo in genere non dovrebbe essere presente nell'applicazione. Se è incluso nell'applicazione, prendete le precauzioni appropriate per assicurarvi che le chiavi in chiaro non siano visibili nel codice, sulla rete e nemmeno nella memoria del computer.

Di seguito è riportato il profilo creato da questo esempio.

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

Aggiorna un profilo esistente a livello di codice

Questo esempio mostra come aggiornare a livello di codice il profilo creato in precedenza.

[Utilizza le seguenti classi di Amazon.Runtime.CredentialManagementnamespace: e NetSDK.CredentialProfileCredentialsFile](#) Utilizza anche la [RegionEndpoint](#) classe dello spazio dei nomi [Amazon](#).


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

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

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

Di seguito è riportato il profilo aggiornato.

```
"[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

È inoltre possibile impostare la AWS regione in altre posizioni e utilizzando altri metodi. Per ulteriori informazioni, consulta [Configura la AWS regione](#).

Caratteristiche della AWS SDK for .NET

Questa sezione fornisce informazioni sulle funzionalità AWS SDK for .NET che potresti dover prendere in considerazione durante la creazione delle tue applicazioni.

Assicurati di aver prima [configurato il tuo progetto](#).

Per informazioni sullo sviluppo di software per AWS servizi specifici e esempi di codice, consulta [Lavora con AWS i servizi](#). Per ulteriori esempi di codice, vedere [AWS SDK for .NET esempi di codice](#).

Argomenti

- [API asincrone di AWS per .NET](#)
- [Tentativi e timeout](#)
- [Impaginatori](#)
- [Strumenti aggiuntivi](#)

API asincrone di AWS per .NET

AWS SDK for .NET utilizza il Task-based Asynchronous Pattern (TAP) per la sua implementazione asincrona. Per ulteriori informazioni sul TAP, vedere [Task-based Asynchronous Pattern \(TAP\)](#) su docs.microsoft.com.

Questo argomento offre una panoramica su come utilizzare TAP nelle chiamate ai clienti di assistenza. AWS

I metodi asincroni nell'AWS SDK for .NET API sono operazioni basate sulla Task classe o sulla classe. Task<TResult> [Vedi docs.microsoft.com per informazioni su queste classi: Task class, Task class.](#) <TResult>

Quando questi metodi API vengono chiamati nel codice, devono essere chiamati all'interno di una funzione dichiarata con la async parola chiave, come illustrato nell'esempio seguente.

```
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;
}
```

Come illustrato nel frammento di codice precedente, l'ambito preferito per la `async` dichiarazione è la funzione. `Main` L'impostazione di questo `async` ambito garantisce che tutte le chiamate ai client di AWS servizio debbano essere asincrone. Se per qualche motivo non puoi `Main` dichiarare di essere asincrono, puoi utilizzare la `async` parola chiave su funzioni diverse da `Main` e quindi chiamare i metodi API da lì, come mostrato nell'esempio seguente.

```
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;
}
```

Notate la `Task<>` sintassi speciale necessaria `Main` quando utilizzate questo modello. Inoltre, è necessario utilizzare il **Result** membro della risposta per ottenere i dati.

È possibile visualizzare esempi completi di chiamate asincrone ai client di AWS assistenza nella [Fai un breve tour](#) sezione ([Semplice app multiplatformae](#)[Semplice app basata su Windows](#)) e in [Esempi di codice con indicazioni](#)

Tentativi e timeout

AWS SDK for .NET consente di configurare il numero di tentativi e i valori di timeout per le richieste HTTP ai servizi. AWS Se i valori predefiniti per i nuovi tentativi e il timeout non sono appropriati per la tua applicazione, puoi regolarli in base alle tue esigenze, tuttavia è importante comprendere che questa modifica può influire sul comportamento dell'applicazione.

Per determinare i valori da utilizzare per i nuovi tentativi e i timeout, considera quanto segue:

- Come devono reagire l'applicazione AWS SDK for .NET e la vostra applicazione quando la connettività di rete peggiora o un AWS servizio non è raggiungibile? Preferisci che i tentativi di chiamata si interrompano presto o che continuino automaticamente?
- La tua applicazione è o sito Web o un'applicazione rivolta agli utenti che deve essere reattiva o è un processo di elaborazione in background con una maggiore tolleranza rispetto alle latenze più lunghe?
- L'applicazione è distribuita su una rete affidabile con bassa latenza o viene distribuita in una posizione remota con connettività inaffidabile?

Tentativi

Panoramica

AWS SDK for .NET Possono riprovare le richieste che non vanno a buon fine a causa di limitazioni sul lato server o interruzioni delle connessioni. Esistono due proprietà delle classi di configurazione del servizio che è possibile utilizzare per specificare il comportamento di ripetizione dei tentativi di un client di servizio. [Le classi di configurazione del servizio ereditano queste proprietà dall'astratto Amazon.Runtime.ClientConfig classe dell'API Reference: AWS SDK for .NET](#)

- `RetryMode` [specifica una delle tre modalità di riprova, definite in Amazon.Runtime.RequestRetryMode](#) enumerazione.

Il valore predefinito per l'applicazione può essere controllato utilizzando la variabile di ambiente `AWS_RETRY_MODE` o l'impostazione `retry_mode` nel file di configurazione condiviso. AWS

- `MaxErrorRetrys` specifica il numero di tentativi consentiti a livello di client di servizio; l'SDK riprova l'operazione il numero di volte specificato prima di fallire e generare un'eccezione.

Il valore predefinito per l'applicazione può essere controllato utilizzando la variabile di ambiente `AWS_MAX_ATTEMPTS` o l'impostazione `max_attempts` nel file di configurazione condiviso. `AWS`

[Le descrizioni dettagliate di queste proprietà sono disponibili nell'estratto Amazon.Runtime.](#)

[ClientConfig](#) classe dell'API Reference. [AWS SDK for .NET](#) Ogni valore di `RetryMode` corrisponde per impostazione predefinita a un particolare valore di `MaxErrorRetry`, come illustrato nella tabella seguente.

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

Comportamento

All'avvio dell'applicazione

All'avvio dell'applicazione, i valori predefiniti per `RetryMode` e `MaxErrorRetry` vengono configurati dall'SDK. Questi valori predefiniti vengono utilizzati quando si crea un client di servizio, a meno che non si specifichino altri valori.

- Se le proprietà non sono impostate nell'ambiente, l'impostazione predefinita per `RetryMode` è configurata come `Legacy` e l'impostazione predefinita per `MaxErrorRetry` è configurata con il valore corrispondente della tabella precedente.
- Se la modalità di riprova è stata impostata nell'ambiente in uso, tale valore viene utilizzato come valore predefinito per `RetryMode`. Il valore predefinito per `MaxErrorRetry` è configurato con il valore corrispondente della tabella precedente, a meno che nell'ambiente in uso non sia stato impostato anche il valore per il numero massimo di errori (descritto di seguito).
- Se il valore per il numero massimo di errori è stato impostato nell'ambiente, tale valore viene utilizzato come valore predefinito per `MaxErrorRetry`. Amazon DynamoDB è l'eccezione a questa

regola; il valore DynamoDB predefinito `MaxErrorRetry` per è sempre il valore della tabella precedente.

Durante l'esecuzione dell'applicazione

Quando si crea un client di servizio, è possibile utilizzare i valori predefiniti per `RetryMode` e `MaxErrorRetry`, come descritto in precedenza, oppure specificare altri valori. [Per specificare altri valori, crea e includi un oggetto di configurazione del servizio come `AmazonDynamoDbConfig` o `AmazonSQSConfig` quando crei il client del servizio.](#)

Questi valori non possono essere modificati per un client di servizio dopo la sua creazione.

Considerazioni

Quando si verifica un nuovo tentativo, la latenza della richiesta aumenta. Devi configurare i nuovi tentativi in base ai limiti dell'applicazione in termini di latenza della richiesta totale e percentuali di errore.

Timeout

AWS SDK for .NET consente di configurare il timeout della richiesta e i valori di timeout di lettura/scrittura del socket a livello del client di servizio. [Questi valori sono specificati nelle proprietà `ReadWriteTimeout`, `Timeout` e dell'oggetto astratto `Amazon.Runtime.ClientConfig` classe.](#) Questi valori vengono trasmessi come `ReadWriteTimeout` proprietà `Timeout` e degli `HttpRequest` oggetti creati dall'oggetto client del AWS servizio. Per impostazione predefinita, il valore `Timeout` è di 100 secondi e il valore `ReadWriteTimeout` è di 300 secondi.

Se la rete ha una latenza elevata o se esistono condizioni che causano la ripetizione di un'operazione, l'utilizzo di valori lunghi per il timeout e di un elevato numero di tentativi può far sì che alcune operazioni dell'SDK sembrino non rispondere.

Note

La versione di destinata alla AWS SDK for .NET Portable Class Library (PCL) utilizza la [Httpclient](#) classe anziché la `HttpRequest` classe e supporta solo la proprietà [Timeout](#).

Di seguito sono elencate le eccezioni per i valori di timeout predefiniti. Questi valori vengono sovrascritti quando imposti esplicitamente i valori di timeout.

- [Timeoute ReadWriteTimeout](#) sono impostati sui valori massimi se il metodo chiamato carica uno stream, come [AmazonS3Client.PutObjectAsync\(\)](#), [client Amazon S3.UploadPartAsync\(\)](#), [AmazonGlacierClient UploadArchiveAsync\(\)](#) e così via.
- Le versioni di destinazione .NET Framework impostate Timeout e ReadWriteTimeout impostano i valori massimi per tutti gli [AmazonS3Client](#) e gli oggetti. AWS SDK for .NET [AmazonGlacierClient](#)
- [Le versioni destinate alla AWS SDK for .NET Portable Class Library \(PCL\) e .NET Core](#) sono impostate sul valore massimo per tutti gli Timeout [AmazonS3Client](#) e gli oggetti. [AmazonGlacierClient](#)

Esempio

L'esempio seguente mostra come specificare la modalità di riprova standard, un massimo di 3 tentativi, un timeout di 10 secondi e un timeout di lettura/scrittura di 10 secondi (se applicabile). [Al costruttore AmazonS3Client viene assegnato un oggetto AmazonS3Config.](#)

```
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  
    });
```

Impaginatori

Alcuni AWS servizi raccolgono e archiviano una grande quantità di dati, che è possibile recuperare utilizzando le chiamate API di AWS SDK for .NET. Se la quantità di dati che desideri recuperare diventa troppo grande per una singola chiamata API, puoi suddividere i risultati in parti più gestibili utilizzando la paginazione.

Per consentirvi di eseguire l'impaginazione, gli oggetti di richiesta e risposta per molti client di servizio nell'SDK forniscono un token di continuazione (in genere denominato NextToken). Alcuni di questi client di servizio forniscono anche impaginatori.

Gli impaginatori consentono di evitare il sovraccarico del token di continuazione, che potrebbe comportare loop, variabili di stato, più chiamate API e così via. Quando si utilizza un impaginatore, è possibile recuperare i dati da un AWS servizio tramite una singola riga di codice, una dichiarazione di ciclo. `foreach` Se sono necessarie più chiamate API per recuperare i dati, l'impaginatore se ne occupa per te.

Dove posso trovare gli impaginatori?

Non tutti i servizi forniscono impaginatori. [Un modo per determinare se un servizio fornisce un impaginatore per una particolare API consiste nell'esaminare la definizione di una classe client di servizio nell'API Reference. AWS SDK for .NET](#)

Ad esempio, se si esamina la definizione della [AmazonCloudWatchLogsClient](#) classe, viene visualizzata una `Paginators` proprietà. Questa è la proprietà che fornisce un impaginatore per Amazon CloudWatch Logs.

Cosa mi danno gli impaginatori?

Gli impaginatori contengono proprietà che consentono di visualizzare le risposte complete. In genere contengono anche una o più proprietà che consentono di accedere alle parti più interessanti delle risposte, che chiameremo risultati chiave.

Ad esempio, in `AmazonCloudWatchLogsClient` precedenza, l'`Paginator` oggetto contiene una `Responses` proprietà con l'[DescribeLogGroupsResponse](#) oggetto completo della chiamata API. Questa `Responses` proprietà contiene, tra le altre cose, una raccolta dei gruppi di log.

L'oggetto `Paginator` contiene anche un risultato chiave denominato `LogGroups`. Questa proprietà contiene solo la parte della risposta relativa ai gruppi di log. La disponibilità di questo risultato chiave consente di ridurre e semplificare il codice in molte circostanze.

Impaginazione sincrona e asincrona

Gli impaginatori forniscono meccanismi sincroni e asincroni per l'impaginazione. L'impaginazione sincrona è disponibile nei progetti .NET Framework 4.5 (o versioni successive). L'impaginazione asincrona è disponibile nei progetti .NET Core (.NET Core 3.1, .NET 5 e così via).

Poiché sono consigliate le operazioni asincrone e .NET Core, l'esempio che segue mostra l'impaginazione asincrona. Le informazioni su come eseguire le stesse attività utilizzando

l'impaginazione sincrona e.NET Framework 4.5 (o versione successiva) vengono mostrate dopo l'esempio in [Considerazioni aggiuntive per gli impaginatori](#)

Esempio

L'esempio seguente mostra come utilizzare per visualizzare un elenco di gruppi di log. AWS SDK for .NET Per contro, l'esempio mostra come eseguire questa operazione con e senza impaginatori. Prima di esaminare il codice completo, mostrato più avanti, considera i seguenti frammenti.

Ottenere gruppi di CloudWatch log senza impaginatori

```
// 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));
```

Ottenere gruppi di CloudWatch log utilizzando gli impaginatori

```
// 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);
}
```

I risultati di questi due frammenti sono esattamente gli stessi, quindi è possibile vedere chiaramente il vantaggio dell'uso degli impaginatori.

Note

Prima di provare a compilare ed eseguire il codice completo, assicuratevi di aver [configurato l'ambiente e il progetto](#).

Potresti anche aver bisogno di [Microsoft.Bcl.AsyncInterfaces](#) NuGet pacchetto perché gli impaginatori asincroni utilizzano l'interfaccia. `IAsyncEnumerable`

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.CloudWatch](#)

Elementi di programmazione:

- [Namespace Amazon. CloudWatch](#)
Classe [AmazonCloudWatchLogsClient](#)
- [Namespace Amazon. CloudWatchLogs.Modello](#)

Classe [DescribeLogGroupsRequest](#)

Classe [DescribeLogGroupsResponse](#)

Classe [LogGroup](#)

Codice completo

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

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

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

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

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

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

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

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

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

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

Considerazioni aggiuntive per gli impaginatori

- Gli impaginatori non possono essere usati più di una volta

Se avete bisogno dei risultati di un particolare AWS impaginatore in più posizioni del codice, non dovete usare un oggetto paginator più di una volta. Invece, create un nuovo impaginatore ogni

volta che ne avete bisogno. Questo concetto è illustrato nel codice di esempio precedente del metodo `DisplayLogGroupsWithPaginators`

- Impaginazione sincrona

L'impaginazione sincrona è disponibile per i progetti .NET Framework 4.5 (o versioni successive).

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e la versione minima di .NET Framework passerà alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

Per vedere ciò, crea un progetto .NET Framework 4.5 (o successivo) e copia il codice precedente su di esso. Quindi rimuovi semplicemente la `await` parola chiave dalle due chiamate al `foreach` paginatore, come mostrato nell'esempio seguente.

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

Crea ed esegui il progetto per ottenere gli stessi risultati che hai ottenuto con l'impaginazione asincrona.

Strumenti aggiuntivi

Di seguito sono riportati alcuni strumenti aggiuntivi che è possibile utilizzare per semplificare lo sviluppo, la distribuzione e la manutenzione delle applicazioni .NET.

AWSStrumento di distribuzione

Dopo aver sviluppato l'applicazione .NET Core nativa per il cloud su una macchina di sviluppo, puoi utilizzare AWS Deploy Tool for the .NET CLI per distribuire più facilmente l'applicazione su AWS.

Per ulteriori informazioni, consulta [Distribuisci le applicazioni su AWS](#).

AWSMessage Processing Framework per.NET

Questa è la documentazione non definitiva per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Se utilizzi servizi come Amazon SQS, Amazon SNS o EventBridge Amazon, potresti sfruttare AWS il Message Processing Framework per .NET. Per ulteriori informazioni, consulta [AWS Framework di elaborazione dei messaggi per.NET](#).

Autenticazione e autorizzazione avanzata con AWS SDK for .NET

Gli argomenti di questa sezione forniscono informazioni sulle tecniche avanzate di autenticazione e autorizzazione dell'utente AWS SDK for .NET applicazione.

Argomenti

- [Single sign-on con AWS SDK for .NET](#)

Single sign-on con AWS SDK for .NET

AWS IAM Identity Center è un servizio Single Sign-On (SSO) basato sul cloud che semplifica la gestione centralizzata dell'accesso SSO a tutte le tue applicazioni e al cloud. Account AWS [Per tutti i dettagli, consulta la Guida per l'utente di IAM Identity Center.](#)

Se non conosci il modo in cui un SDK interagisce con IAM Identity Center, consulta le seguenti informazioni.

Modello di interazione di alto livello

Ad alto livello, gli SDK interagiscono con IAM Identity Center in modo simile al seguente schema:

1. IAM Identity Center è configurato, in genere tramite la [console IAM Identity Center](#), e un utente SSO è invitato a partecipare.
2. Il AWS config file condiviso sul computer dell'utente viene aggiornato con informazioni SSO.
3. L'utente accede tramite IAM Identity Center e riceve credenziali a breve termine per le autorizzazioni AWS Identity and Access Management (IAM) che sono state configurate per lui. Questo accesso può essere avviato tramite uno strumento non SDK come o programmaticamente tramite un'AWS CLI applicazione.NET.
4. L'utente procede a svolgere il proprio lavoro. Quando eseguono altre applicazioni che utilizzano SSO, non è necessario accedere nuovamente per aprire le applicazioni.

Il resto di questo argomento fornisce informazioni di riferimento per la configurazione e l'utilizzo AWS IAM Identity Center. Fornisce informazioni supplementari e più avanzate rispetto alla configurazione SSO di base in. [Configura l'autenticazione SDK](#) Se non conosci l'SSO AWS, potresti voler dare

un'occhiata prima a questo argomento per ottenere informazioni fondamentali e poi ai seguenti tutorial per vedere l'SSO in azione:

- [Tutorial: solo applicazione.NET](#)
- [Tutorial: AWS CLI e applicazione.NET](#)

Questo argomento contiene le sezioni seguenti:

- [Prerequisiti](#)
- [Configurazione di un profilo SSO](#)
- [Generazione e utilizzo di token SSO](#)
- [Risorse aggiuntive](#)
- [Tutorial](#)

Prerequisiti

Prima di utilizzare IAM Identity Center, devi eseguire determinate attività, come la scelta di una fonte di identità e la configurazione delle relative applicazioni. Account AWS Per ulteriori informazioni, consulta la seguente documentazione:

- Per informazioni generali su queste attività, consulta la Guida [introduttiva](#) alla IAM Identity Center User Guide.
- Per esempi di attività specifiche, consulta l'elenco dei tutorial alla fine di questo argomento. Tuttavia, assicurati di leggere le informazioni contenute in questo argomento prima di provare i tutorial.

Configurazione di un profilo SSO

Dopo aver [configurato](#) IAM Identity Center nel file pertinente Account AWS, è necessario aggiungere un profilo denominato per SSO al file condiviso AWS config dell'utente. Questo profilo viene utilizzato per connettersi al [portale di AWS accesso](#), che restituisce credenziali a breve termine per le autorizzazioni IAM configurate per l'utente.

Il config file condiviso viene in genere denominato %USERPROFILE%\aws\config in Windows e ~/.aws/config in Linux e macOS. Puoi usare il tuo editor di testo preferito per aggiungere un

nuovo profilo per SSO. In alternativa, puoi usare il `aws configure sso` comando. Per ulteriori informazioni su questo comando, consulta [Configurazione della AWS CLI per l'utilizzo di IAM Identity Center nella Guida per AWS Command Line Interface](#) l'utente.

Il nuovo profilo è simile al seguente:

```
[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
```

Le impostazioni per il nuovo profilo sono definite di seguito. Le prime due impostazioni definiscono il portale di AWS accesso. Le altre due impostazioni sono una coppia che, prese insieme, definisce le autorizzazioni configurate per un utente. Tutte e quattro le impostazioni sono obbligatorie.

sso_start_url

L'URL che rimanda al [portale di AWS accesso](#) dell'organizzazione. Per trovare questo valore, apri la [console IAM Identity Center](#), scegli Impostazioni e trova l'URL del portale.

sso_region

Il Regione AWS che contiene l'host del portale di accesso. Questa è la regione selezionata quando hai abilitato IAM Identity Center. Può essere diversa dalle regioni utilizzate per altre attività.

Per un elenco completo di Regioni AWS e dei relativi codici, consulta [Endpoint regionali](#) in Riferimenti generali di Amazon Web Services

sso_account_id

L'ID di un utente Account AWS che è stato aggiunto tramite il AWS Organizations servizio. Per visualizzare l'elenco degli account disponibili, vai alla [console IAM Identity Center](#) e apri la Account AWS pagina. L'ID dell'account che scegli per questa impostazione corrisponderà al valore che intendi assegnare all'`sso_role_name` impostazione, mostrato di seguito.

sso_role_name

Il nome di un set di autorizzazioni IAM Identity Center. Questo set di autorizzazioni definisce le autorizzazioni concesse a un utente tramite IAM Identity Center.

La procedura seguente è un modo per trovare il valore per questa impostazione.

1. Vai alla [console IAM Identity Center](#) e apri la Account AWS pagina.
2. Scegli un account per visualizzarne i dettagli. L'account che scegli sarà quello che contiene l'utente o il gruppo SSO a cui desideri concedere le autorizzazioni SSO.
3. Guarda l'elenco di utenti e gruppi assegnati all'account e trova l'utente o il gruppo di interesse. Il set di autorizzazioni specificato nell'`sso_role_name` impostazione è uno dei set associati a questo utente o gruppo.

Quando dai un valore a questa impostazione, usa il nome del set di autorizzazioni, non l'Amazon Resource Name (ARN).

Ai set di autorizzazioni sono associate politiche IAM e politiche di autorizzazioni personalizzate. Per ulteriori informazioni, consulta i [set di autorizzazioni nella Guida](#) per l'utente di IAM Identity Center.

Generazione e utilizzo di token SSO

Per utilizzare SSO, un utente deve prima generare un token temporaneo e quindi utilizzare tale token per accedere alle AWS applicazioni e alle risorse appropriate. Per le applicazioni.NET, è possibile utilizzare i seguenti metodi per generare e utilizzare questi token temporanei:

- Crea applicazioni.NET che generano prima un token, se necessario, e poi utilizza il token.
- Genera un token con AWS CLI e poi usa il token nelle applicazioni.NET.

Questi metodi sono descritti nelle sezioni seguenti e dimostrati nei [tutorial](#).

Important

L'applicazione deve fare riferimento ai seguenti NuGet pacchetti in modo che la risoluzione SSO possa funzionare:

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Il mancato riferimento a questi pacchetti comporterà un'eccezione di runtime.

Solo applicazione.NET

In questa sezione viene illustrato come creare un'applicazione.NET che generi un token SSO temporaneo, se necessario, e quindi utilizzi tale token. Per un tutorial completo di questo processo, vedi [Tutorial per SSO che utilizza solo applicazioni.NET](#).

Genera e usa un token SSO a livello di codice

Oltre a utilizzare ilAWS CLI, puoi anche generare un token SSO a livello di codice.

A tale scopo, l'applicazione crea un [AWSCredentials](#) soggetto per il profilo SSO, che carica le credenziali temporanee, se disponibili. L'applicazione deve quindi eseguire il cast dell'[AWSCredentials](#) soggetto su un [SSOAWSCredentials](#) soggetto e impostare alcune proprietà [Options](#), incluso un metodo di callback utilizzato per richiedere all'utente le informazioni di accesso, se necessario.

Questo metodo è illustrato nel seguente frammento di codice.

Important

L'applicazione deve fare riferimento ai seguenti NuGet pacchetti in modo che la risoluzione SSO possa funzionare:

- AWSSDK.SSO
- AWSSDK.SSOIDC

Il mancato riferimento a questi pacchetti comporterà un'eccezione di runtime.

```
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");

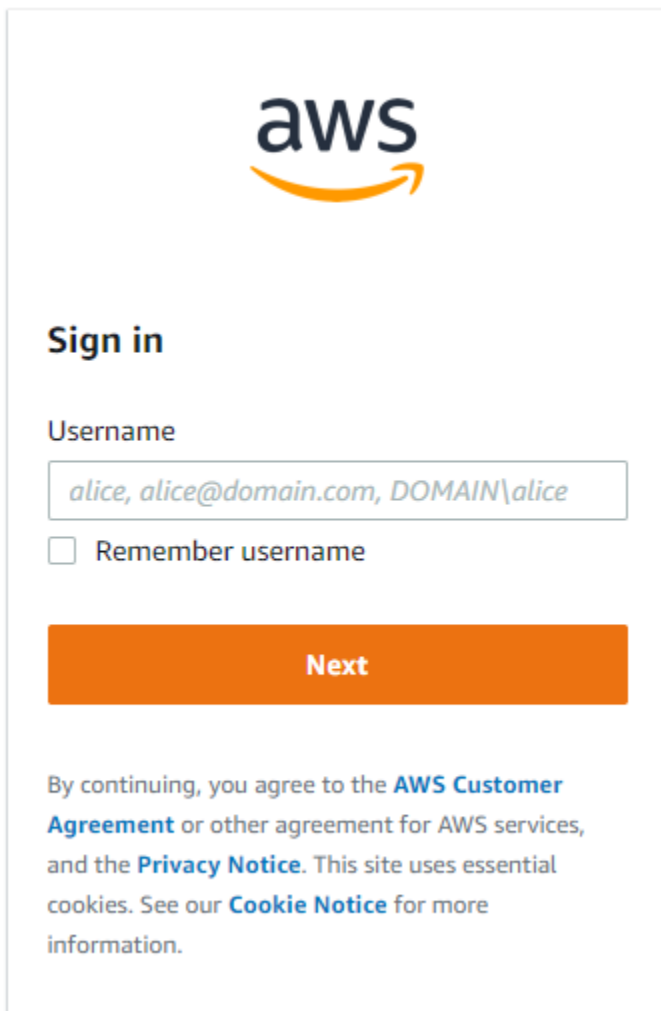
    var ssoCredentials = credentials as SSOAWSCredentials;

    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;
}
```

Se non è disponibile un token SSO appropriato, viene avviata la finestra del browser predefinita e viene aperta la pagina di accesso appropriata. Ad esempio, se utilizzi IAM Identity Center come origine dell'identità, l'utente visualizza una pagina di accesso simile alla seguente:



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

La stringa di testo fornita non `SSOAWSCredentials.Options.ClientName` può contenere spazi. Se la stringa contiene spazi, otterrai un'eccezione di runtime.

[Tutorial per SSO che utilizza solo applicazioni.NET](#)

AWS CLI e applicazione.NET

Questa sezione mostra come generare un token SSO temporaneo utilizzando e come utilizzare tale token in un'applicazione. AWS CLI Per un tutorial completo di questo processo, vedi [Tutorial per l'utilizzo delle AWS CLI applicazioni SSO e.NET](#).

Genera un token SSO utilizzando il AWS CLI

Oltre a generare un token SSO temporaneo a livello di codice, si utilizza il AWS CLI per generare il token. Le seguenti informazioni mostrano come fare.

Dopo che l'utente ha creato un profilo abilitato per l'SSO, come illustrato [nella sezione precedente](#), esegue il `aws sso login` comando da AWS CLI. Devono assicurarsi di includere il `--profile` parametro con il nome del profilo abilitato all'SSO. Questo viene mostrato nell'esempio seguente:

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

Se l'utente desidera generare un nuovo token temporaneo dopo la scadenza di quello corrente, può eseguire nuovamente lo stesso comando.

Usa il token SSO generato in un'applicazione.NET

Le seguenti informazioni mostrano come utilizzare un token temporaneo che è già stato generato.

Important

L'applicazione deve fare riferimento ai seguenti NuGet pacchetti in modo che la risoluzione SSO possa funzionare:

- `AWSSDK.SSO`
- `AWSSDK.SSOIDC`

Il mancato riferimento a questi pacchetti comporterà un'eccezione di runtime.

L'applicazione crea un [AWSCredentials](#) oggetto per il profilo SSO, che carica le credenziali temporanee generate in precedenza da AWS CLI. È simile ai metodi mostrati in [Accesso a credenziali e profili in un'applicazione](#) e ha la forma seguente:

```
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;
}
```

L'`AWSCredentials` soggetto viene quindi passato al costruttore per un client di servizio. Per esempio:

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

Note

L'utilizzo `AWSCredentials` per caricare le credenziali temporanee non è necessario se l'applicazione è stata creata per utilizzare il [default] profilo per SSO. In tal caso, l'applicazione può creare client AWS di servizio senza parametri, simili a "»`var client = new AmazonS3Client();`

[Tutorial per l'utilizzo delle AWS CLI applicazioni SSO e.NET](#)

Risorse aggiuntive

Per ulteriore assistenza, consulta le seguenti risorse.

- [Cos'è IAM Identity Center?](#)
- [Configurazione di AWS CLI IAM Identity Center](#)
- [Utilizzo delle credenziali IAM Identity Center nel AWS Toolkit for Visual Studio](#)

Tutorial

Argomenti

- [Tutorial per SSO che utilizza solo applicazioni.NET](#)
- [Tutorial per l'utilizzo delle AWS CLI applicazioni SSO e.NET](#)

Tutorial per SSO che utilizza solo applicazioni.NET

Questo tutorial mostra come abilitare l'SSO per un'applicazione di base e un utente SSO di prova.

[Configura l'applicazione per generare un token SSO temporaneo a livello di codice anziché utilizzare il. AWS CLI](#)

Questo tutorial mostra una piccola parte della funzionalità SSO in AWS SDK for .NET. Per tutti i dettagli sull'utilizzo di IAM Identity Center con AWS SDK for .NET, consulta l'argomento con [informazioni di base](#). In tale argomento, vedi in particolare la descrizione di alto livello di questo scenario nella sottosezione chiamata [Solo applicazione.NET](#).

Note

Diversi passaggi di questo tutorial consentono di configurare servizi come AWS Organizations IAM Identity Center. Se hai già eseguito quella configurazione o se sei interessato solo al codice, puoi passare alla sezione con il [codice di esempio](#).

Prerequisiti

- Configura il tuo ambiente di sviluppo se non l'hai già fatto. Questo è descritto in sezioni come [Installa e configura la tua toolchain e inizia a usare](#).
- Identificane o creane almeno uno Account AWS che puoi usare per testare l'SSO. Ai fini di questo tutorial, questo è chiamato test Account AWS o semplicemente account di test.
- Identifica un utente SSO che possa testare l'SSO per te. Si tratta di una persona che utilizzerà l'SSO e le applicazioni di base che creerai. Per questo tutorial, quella persona potresti essere tu (lo sviluppatore) o qualcun altro. Consigliamo inoltre una configurazione in cui l'utente SSO lavori su un computer che non si trova nel tuo ambiente di sviluppo. Tuttavia, ciò non è strettamente necessario.
- Sul computer dell'utente SSO deve essere installato un framework .NET compatibile con quello utilizzato per configurare l'ambiente di sviluppo.

Configurazione di AWS

Questa sezione mostra come configurare vari AWS servizi per questo tutorial.

Per eseguire questa configurazione, accedi prima al test Account AWS come amministratore. Quindi, procedi come segue:

Amazon S3

Vai alla [console Amazon S3](#) e aggiungi alcuni bucket innocui. Più avanti in questo tutorial, l'utente SSO recupererà un elenco di questi bucket.

AWSIAM

Vai alla [console IAM](#) e aggiungi alcuni utenti IAM. Se concedi le autorizzazioni agli utenti IAM, limita le autorizzazioni a poche innocue autorizzazioni di sola lettura. Più avanti in questo tutorial, l'utente SSO recupererà un elenco di questi utenti IAM.

AWS Organizations

Vai alla [AWS Organizationsconsole](#) e abilita Organizations. Per ulteriori informazioni sulla configurazione di un'organizzazione, consulta [Creazione di un'organizzazione](#) nella [Guida per l'utente di AWS Organizations](#).

Questa azione aggiunge il test Account AWS all'organizzazione come account di gestione. Se disponi di account di prova aggiuntivi, puoi invitarli a far parte dell'organizzazione, ma non è necessario farlo per questo tutorial.

IAM Identity Center

Vai alla [console IAM Identity Center](#) e abilita l'SSO. Esegui la verifica via e-mail, se necessario. Per ulteriori informazioni, consulta [Enable IAM Identity Center](#) nella [Guida per l'utente di IAM Identity Center](#).

Quindi, esegui la seguente configurazione.

Configura IAM Identity Center

1. Vai alla pagina Impostazioni. Cerca «l'URL del portale di accesso» e registra il valore per un uso successivo nell'`sso_start_url` impostazione.
2. Nel banner di AWS Management Console, cerca Regione AWS quello che è stato impostato quando hai abilitato l'SSO. Questo è il menu a discesa a sinistra dell'Account AWSID. Registra il codice regionale per utilizzarlo successivamente nell'`sso_region` impostazione. Questo codice sarà simile `us-east-1`.
3. Crea un utente SSO come segue:
 - a. Vai alla pagina Utenti.
 - b. Scegli Aggiungi utente e inserisci il nome utente, l'indirizzo e-mail, il nome e il cognome dell'utente. Quindi, seleziona Next (Successivo).
 - c. Scegli Avanti nella pagina per i gruppi, quindi rivedi le informazioni e scegli Aggiungi utente.
4. Crea un gruppo come segue:

- a. Vai alla pagina Gruppi.
 - b. Scegli Crea gruppo e inserisci il nome e la descrizione del gruppo.
 - c. Nella sezione Aggiungi utenti al gruppo, seleziona l'utente SSO di prova che hai creato in precedenza. Quindi, seleziona Crea gruppo.
5. Crea un set di autorizzazioni come segue:
- a. Vai alla pagina Set di autorizzazioni e scegli Crea set di autorizzazioni.
 - b. In Tipo di set di autorizzazioni, seleziona Set di autorizzazioni personalizzato e scegli Avanti.
 - c. Apri Inline policy e inserisci la seguente policy:

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

- d. Per questo tutorial, inserisci `SSOReadOnlyRole` come nome del set di autorizzazioni. Aggiungi una descrizione se lo desideri, quindi scegli Avanti.
 - e. Controlla le informazioni, quindi scegli Crea.
 - f. Registra il nome del set di autorizzazioni per utilizzarlo successivamente nell'`sso_role_name` impostazione.
6. Vai alla pagina degli AWSAccount e scegli l'AWSAccount che hai aggiunto all'organizzazione in precedenza.
7. Nella sezione Panoramica di quella pagina, trova l'ID dell'account e registralo per un uso successivo nell'`sso_account_id` impostazione.
8. Scegli la scheda Utenti e gruppi, quindi scegli Assegna utenti o gruppi.
9. Nella pagina Assegna utenti e gruppi, scegli la scheda Gruppi, seleziona il gruppo creato in precedenza e scegli Avanti.

10. Seleziona il set di autorizzazioni creato in precedenza e scegli Avanti, quindi scegli Invia. La configurazione richiede alcuni istanti.

Crea applicazioni di esempio

Create le seguenti applicazioni. Verranno eseguiti sul computer dell'utente SSO.

Elenca i bucket Amazon S3

Includi NuGet i pacchetti `AWSSDK.S3` e `AWSSDK.SecurityToken` in aggiunta a `AWSSDK.S3` e `AWSSDK.SecurityToken`

```
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;
    }
}
}
```

Elenca gli utenti IAM

Includi NuGet pacchetti `AWSSDK.SSO` e `AWSSDK.SSO0IDC` in aggiunta a `AWSSDK.IdentityManagement` e `AWSSDK.SecurityToken`.

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

// 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.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;
    }
}
}
```

Oltre a visualizzare gli elenchi dei bucket Amazon S3 e degli utenti IAM, queste applicazioni visualizzano l'ARN dell'identità utente per il profilo abilitato all'SSO, illustrato in questo tutorial. `my-sso-profile`

[Queste applicazioni eseguono attività di accesso SSO fornendo un metodo di callback nella proprietà Options di un oggetto SSO. AWSCredentials](#)

Istruisci l'utente SSO

Chiedi all'utente SSO di controllare la sua e-mail e accettare l'invito SSO. Viene richiesto loro di impostare una password. Il messaggio potrebbe impiegare alcuni minuti per arrivare nella casella di posta dell'utente SSO.

Fornisci all'utente SSO le applicazioni che hai creato in precedenza.

Quindi, chiedi all'utente SSO di eseguire le seguenti operazioni:

1. Se la cartella che contiene il AWS config file condiviso non esiste, creala. Se la cartella esiste e contiene una sottocartella chiamata `.sso`, elimina quella sottocartella.

La posizione di questa cartella è `%USERPROFILE%\ .aws` in genere in Windows, Linux e macOS.
`~/ .aws`

2. Crea un AWS config file condiviso in quella cartella, se necessario, e aggiungi un profilo come segue:

```
[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. Esegui l'applicazione Amazon S3.
4. Nella pagina di accesso Web risultante, accedi. Utilizza il nome utente contenuto nel messaggio di invito e la password creata in risposta al messaggio.
5. Una volta completato l'accesso, l'applicazione visualizza l'elenco dei bucket S3.
6. Esegui l'applicazione IAM. L'applicazione visualizza l'elenco degli utenti IAM. Questo è vero anche se non è stato eseguito un secondo accesso. L'applicazione IAM utilizza il token temporaneo creato in precedenza.

Pulizia

Se non vuoi conservare le risorse che hai creato durante questo tutorial, puliscile. Queste potrebbero essere AWS risorse o risorse presenti nel tuo ambiente di sviluppo, ad esempio file e cartelle.

Tutorial per l'utilizzo delle AWS CLI applicazioni SSO e .NET

Questo tutorial mostra come abilitare l'SSO per un'applicazione .NET di base e un utente SSO di prova. Utilizza il AWS CLI per generare un token SSO temporaneo invece di [generarlo](#) a livello di codice.

Questo tutorial mostra una piccola parte della funzionalità SSO in AWS SDK for .NET. Per tutti i dettagli sull'utilizzo di IAM Identity Center con AWS SDK for .NET, consulta l'argomento con [informazioni di base](#). In tale argomento, vedi in particolare la descrizione di alto livello di questo scenario nella sottosezione chiamata [AWS CLI e applicazione.NET](#).

Note

Diversi passaggi di questo tutorial consentono di configurare servizi come AWS Organizations IAM Identity Center. Se hai già eseguito queste configurazioni o se sei interessato solo al codice, puoi passare alla sezione con il codice di [esempio](#).

Prerequisiti

- Configura il tuo ambiente di sviluppo se non l'hai già fatto. Questo è descritto in sezioni come [Installa e configura la tua toolchain](#) e [Inizia a usare](#).
- Identificane o creane almeno uno Account AWS che puoi usare per testare l'SSO. Ai fini di questo tutorial, questo è chiamato test Account AWS o semplicemente account di test.
- Identifica un utente SSO che possa testare l'SSO per te. Si tratta di una persona che utilizzerà l'SSO e le applicazioni di base che creerai. Per questo tutorial, quella persona potresti essere tu (lo sviluppatore) o qualcun altro. Consigliamo inoltre una configurazione in cui l'utente SSO lavori su un computer che non si trova nel tuo ambiente di sviluppo. Tuttavia, ciò non è strettamente necessario.
- Sul computer dell'utente SSO deve essere installato un framework.NET compatibile con quello utilizzato per configurare l'ambiente di sviluppo.
- Assicurati che la AWS CLI versione 2 sia [installata sul computer](#) dell'utente SSO. Puoi verificarlo eseguendo `aws --version` in un prompt dei comandi o in un terminale.

Configurazione di AWS

Questa sezione mostra come configurare vari AWS servizi per questo tutorial.

Per eseguire questa configurazione, accedi prima al test Account AWS come amministratore. Quindi, procedi come segue:

Amazon S3

Vai alla [console Amazon S3](#) e aggiungi alcuni bucket innocui. Più avanti in questo tutorial, l'utente SSO recupererà un elenco di questi bucket.

AWSIAM

Vai alla [console IAM](#) e aggiungi alcuni utenti IAM. Se concedi le autorizzazioni agli utenti IAM, limita le autorizzazioni a poche innocue autorizzazioni di sola lettura. Più avanti in questo tutorial, l'utente SSO recupererà un elenco di questi utenti IAM.

AWS Organizations

Vai alla [AWS Organizationsconsole](#) e abilita Organizations. Per ulteriori informazioni sulla configurazione di un'organizzazione, consulta [Creazione di un'organizzazione](#) nella [Guida per l'utente di AWS Organizations](#).

Questa azione aggiunge il test Account AWS all'organizzazione come account di gestione. Se disponi di account di prova aggiuntivi, puoi invitarli a far parte dell'organizzazione, ma non è necessario farlo per questo tutorial.

IAM Identity Center

Vai alla [console IAM Identity Center](#) e abilita l'SSO. Esegui la verifica via e-mail, se necessario. Per ulteriori informazioni, consulta [Enable IAM Identity Center](#) nella [Guida per l'utente di IAM Identity Center](#).

Quindi, esegui la seguente configurazione.

Configura IAM Identity Center

1. Vai alla pagina Impostazioni. Cerca «l'URL del portale di accesso» e registra il valore per un uso successivo nell'`sso_start_url` impostazione.
2. Nel banner diAWS Management Console, cerca Regione AWS quello che è stato impostato quando hai abilitato l'SSO. Questo è il menu a discesa a sinistra dell'Account AWSID. Registra il codice regionale per utilizzarlo successivamente nell'`sso_region` impostazione. Questo codice sarà simile `us-east-1`.
3. Crea un utente SSO come segue:
 - a. Vai alla pagina Utenti.

- b. Scegli Aggiungi utente e inserisci il nome utente, l'indirizzo e-mail, il nome e il cognome dell'utente. Quindi, seleziona Next (Successivo).
 - c. Scegli Avanti nella pagina per i gruppi, quindi rivedi le informazioni e scegli Aggiungi utente.
4. Crea un gruppo come segue:
- a. Vai alla pagina Gruppi.
 - b. Scegli Crea gruppo e inserisci il nome e la descrizione del gruppo.
 - c. Nella sezione Aggiungi utenti al gruppo, seleziona l'utente SSO di prova che hai creato in precedenza. Quindi, seleziona Crea gruppo.
5. Crea un set di autorizzazioni come segue:
- a. Vai alla pagina Set di autorizzazioni e scegli Crea set di autorizzazioni.
 - b. In Tipo di set di autorizzazioni, seleziona Set di autorizzazioni personalizzato e scegli Avanti.
 - c. Apri Inline policy e inserisci la seguente policy:

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

- d. Per questo tutorial, inserisci `SSOReadOnlyRole` come nome del set di autorizzazioni. Aggiungi una descrizione se lo desideri, quindi scegli Avanti.
 - e. Controlla le informazioni, quindi scegli Crea.
 - f. Registra il nome del set di autorizzazioni per utilizzarlo successivamente nell'`sso_role_name` impostazione.
6. Vai alla pagina degli AWSAccount e scegli l'AWSAccount che hai aggiunto all'organizzazione in precedenza.

7. Nella sezione Panoramica di quella pagina, trova l'ID dell'account e registralo per un uso successivo nell'`sso_account_id` impostazione.
8. Scegli la scheda Utenti e gruppi, quindi scegli Assegna utenti o gruppi.
9. Nella pagina Assegna utenti e gruppi, scegli la scheda Gruppi, seleziona il gruppo creato in precedenza e scegli Avanti.
10. Seleziona il set di autorizzazioni creato in precedenza e scegli Avanti, quindi scegli Invia. La configurazione richiede alcuni istanti.

Crea applicazioni di esempio

Create le seguenti applicazioni. Verranno eseguiti sul computer dell'utente SSO.

Elenca i bucket Amazon S3

Include NuGet pacchetti `AWSSDK.S3` e `AWSSDK.SSO0IDC` in aggiunta a `AWSSDK.S3` e `AWSSDK.SecurityToken`

```
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 SS0Example.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;
    }
}
```

```
    }  
  }  
}
```

Elenca gli utenti IAM

Includi NuGet pacchetti `AWSSDK.SSO` e `AWSSDK.SSO0IDC` in aggiunta a `AWSSDK.IdentityManagement` e `AWSSDK.SecurityToken`.

```
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;
        }
    }
}
```

Oltre a visualizzare gli elenchi dei bucket Amazon S3 e degli utenti IAM, queste applicazioni visualizzano l'ARN dell'identità utente per il profilo abilitato all'SSO, illustrato in questo tutorial. `my-sso-profile`

Istruisci l'utente SSO

Chiedi all'utente SSO di controllare la sua e-mail e accettare l'invito SSO. Viene richiesto loro di impostare una password. Il messaggio potrebbe impiegare alcuni minuti per arrivare nella casella di posta dell'utente SSO.

Fornisci all'utente SSO le applicazioni che hai creato in precedenza.

Quindi, chiedi all'utente SSO di eseguire le seguenti operazioni:

1. Se la cartella che contiene il AWS config file condiviso non esiste, creala. Se la cartella esiste e contiene una sottocartella chiamata `.sso`, elimina quella sottocartella.

La posizione di questa cartella è `%USERPROFILE%\ .aws` in genere in Windows, Linux e macOS.
`~/ .aws`

2. Crea un AWS config file condiviso in quella cartella, se necessario, e aggiungi un profilo come segue:

```
[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. Esegui l'applicazione Amazon S3. Viene visualizzata un'eccezione di runtime.
4. Eseguire il comando AWS CLI seguente:

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

5. Nella pagina di accesso Web risultante, accedi. Utilizza il nome utente contenuto nel messaggio di invito e la password creata in risposta al messaggio.
6. Esegui nuovamente l'applicazione Amazon S3. L'applicazione ora visualizza l'elenco dei bucket S3.
7. Esegui l'applicazione IAM. L'applicazione visualizza l'elenco degli utenti IAM. Questo è vero anche se non è stato eseguito un secondo accesso. L'applicazione IAM utilizza il token temporaneo creato in precedenza.

Pulizia

Se non vuoi conservare le risorse che hai creato durante questo tutorial, puliscile. Queste potrebbero essere AWS risorse o risorse presenti nel tuo ambiente di sviluppo, ad esempio file e cartelle.

Distribuisci le applicazioni su AWS

Dopo aver sviluppato l'applicazione o il servizio .NET Core nativi per il cloud su una macchina di sviluppo, ti consigliamo di distribuirla su AWS. Puoi farlo utilizzando AWS Management Console o determinati servizi come AWS CloudFormation o AWS Cloud Development Kit (AWS CDK). Puoi anche utilizzare AWS strumenti che sono stati creati allo scopo di distribuzione. Utilizzando questi strumenti, è possibile effettuare le seguenti operazioni.

Esegui la distribuzione da .NET CLI

È possibile utilizzare i seguenti AWS strumenti per .NET CLI per distribuire le applicazioni su AWS:

- [AWS Strumento di distribuzione per .NET CLI](#): supporta le distribuzioni [AWS App Runners](#) su Amazon Elastic Container Service (Amazon ECS) e [AWS Elastic Beanstalk](#)
- [AWS Lambda Strumenti per l'interfaccia della riga di comando .NET: supporta](#) le distribuzioni di progetti AWS Lambda

Esegui la distribuzione dai toolkit IDE

Puoi utilizzare i AWS toolkit per distribuire le tue applicazioni direttamente dall'IDE di tua scelta:

- [AWS Toolkit for Visual Studio](#)

Note

La funzionalità «Pubblica su AWS» del toolkit espone le stesse funzionalità del AWS Deploy Tool for .NET CLI. Per ulteriori informazioni, consulta la sezione [Pubblica su AWS nella Guida per l'utente](#). AWS Toolkit for Visual Studio

- [AWS Toolkit for JetBrains](#)

Vedi [Lavora con applicazioni AWS serverless](#) e [Lavora con AWS App Runner](#).

- [AWS Toolkit per VS Code](#)

[Vedi Utilizzo e utilizzo delle applicazioni serverless. AWS App Runner](#)

- [AWS Toolkit for Azure DevOps](#)

Casi d'uso

Le sezioni seguenti contengono scenari di casi d'uso per determinati tipi di applicazioni, incluse informazioni su come utilizzare l'interfaccia della riga di comando di .NET per distribuire tali applicazioni.

- [App ASP.NET Core](#)
- [App.NET Console](#)
- [App Blazor WebAssembly](#)
- [Progetti AWS Lambda](#)

App ASP.NET Core

[Lo strumento di AWS distribuzione](#) per l'interfaccia della riga di comando .NET consente di distribuire le applicazioni ASP.NET e guida l'utente attraverso un processo di distribuzione. È uno strumento interattivo per l'interfaccia della riga di comando .NET che consente di distribuire applicazioni.NET con conoscenze minime. AWS

Il Deploy Tool ha le seguenti funzionalità:

- Consigli di calcolo per la tua applicazione: ottieni i consigli di calcolo e scopri qual è l'AWSelaborazione più adatta alla tua applicazione.
- Generazione di file Docker: lo strumento genera un Dockerfile se necessario o utilizza un Dockerfile esistente.
- Imballaggio e distribuzione automatici: lo strumento crea gli elementi di distribuzione, effettua il provisioning dell'infrastruttura utilizzando un progetto di AWS CDK distribuzione generato e distribuisce l'applicazione sull'elaborazione prescelta. AWS
- Distribuzioni ripetibili e condivisibili: puoi generare e modificare AWS CDK progetti di distribuzione per adattarli al tuo caso d'uso specifico. Puoi anche controllare le versioni dei tuoi progetti e condividerli con il tuo team per distribuzioni ripetibili.
- Guida all'apprendimento AWS CDK di in.NET: lo strumento consente di apprendere gradualmente AWS gli strumenti di base su cui è basato, come ad esempio. AWS CDK

[Lo strumento AWS Deploy](#) supporta la distribuzione di applicazioni ASP.NET Core nei seguenti servizi: AWS

- Utilizzo del [servizio Amazon ECS AWS Fargate](#): supporta le distribuzioni di applicazioni Web su Amazon Elastic Container Service (Amazon ECS) con potenza di calcolo gestita da un motore di elaborazione serverless. AWS Fargate
- [AWS App Runner](#)- Supporta le implementazioni su un servizio completamente gestito che semplifica per gli sviluppatori la distribuzione di applicazioni Web e API containerizzate su larga scala. Non è richiesta alcuna esperienza precedente in materia di infrastruttura.
- [AWS Elastic Beanstalk](#)- Supporta le implementazioni di un servizio che semplifica per gli sviluppatori la distribuzione di applicazioni Web e API in un ambiente completamente gestito su larga scala. Non è richiesta alcuna esperienza precedente in materia di infrastruttura.

Per ulteriori informazioni, consulta la [panoramica degli strumenti](#). Per iniziare da lì, accedi a Documentazione, Guida introduttiva e scegli [Come installare](#) per le istruzioni di installazione.

App.NET Console

[Lo strumento di AWS distribuzione](#) per l'interfaccia della riga di comando .NET consente di distribuire le applicazioni.NET Console come servizio o attività pianificata come immagine contenitore su Linux e guida l'utente attraverso un processo di distribuzione. Se l'applicazione non dispone di un Dockerfile, lo strumento lo genera automaticamente. Altrimenti, viene utilizzato un Dockerfile esistente.

Il Deploy Tool ha le seguenti funzionalità:

- Consigli di calcolo per la tua applicazione: ottieni i consigli di calcolo e scopri qual è l'AWSelaborazione più adatta alla tua applicazione.
- Generazione di file Docker: lo strumento genera un Dockerfile se necessario o utilizza un Dockerfile esistente.
- Imballaggio e distribuzione automatici: lo strumento crea gli elementi di distribuzione, effettua il provisioning dell'infrastruttura utilizzando un progetto di AWS CDK distribuzione generato e distribuisce l'applicazione sull'elaborazione prescelta. AWS
- Distribuzioni ripetibili e condivisibili: puoi generare e modificare AWS CDK progetti di distribuzione per adattarli al tuo caso d'uso specifico. Puoi anche controllare le versioni dei tuoi progetti e condividerli con il tuo team per distribuzioni ripetibili.
- Guida all'apprendimento AWS CDK di us.NET: lo strumento consente di apprendere gradualmente AWS gli strumenti di base su cui è basato, come ad esempio. AWS CDK

Il [AWSDeploy Tool](#) supporta la distribuzione di applicazioni.NET Console ai seguenti servizi: AWS

- Utilizzo di [Amazon ECS Service AWS Fargate](#): supporta la distribuzione di applicazioni.NET come servizio (ad esempio, un processore in background) su Amazon Elastic Container Service (Amazon ECS) con potenza di elaborazione gestita da un motore di elaborazione serverless. AWS Fargate
- Utilizzo di [Amazon ECS Scheduled Task AWS Fargate](#): supporta le distribuzioni di applicazioni.NET come attività pianificata (ad esempio, end-of-day processo) su Amazon ECS con potenza di calcolo gestita da un motore di elaborazione serverless. AWS Fargate

[Per ulteriori informazioni, consulta la panoramica dello strumento.](#) Per iniziare da lì, accedi a Documentazione, Guida introduttiva e scegli [Come installare](#) per le istruzioni di installazione.

App Blazor WebAssembly

[Lo strumento di AWS distribuzione](#) per la CLI.NET ti aiuta a ospitare la tua applicazione WebAssembly Blazor in Amazon S3, utilizzando Amazon per la distribuzione in rete di contenuti. CloudFront La tua app viene distribuita in un bucket S3 per l'hosting web. Lo strumento crea e configura un bucket S3, quindi carica l'applicazione Blazor nel bucket.

Il Deploy Tool ha le seguenti funzionalità:

- Imballaggio e distribuzione automatici: lo strumento crea gli elementi di implementazione, effettua il provisioning dell'infrastruttura utilizzando un progetto di AWS CDK distribuzione generato e distribuisce l'applicazione sull'elaborazione prescelta. AWS
- Distribuzioni ripetibili e condivisibili: puoi generare e modificare AWS CDK progetti di distribuzione per adattarli al tuo caso d'uso specifico. Puoi anche controllare le versioni dei tuoi progetti e condividerli con il tuo team per distribuzioni ripetibili.
- Guida all'apprendimento AWS CDK di us.NET: lo strumento consente di apprendere gradualmente AWS gli strumenti di base su cui è basato, come ad esempio. AWS CDK

Per ulteriori informazioni, consulta la [panoramica degli strumenti](#). Per iniziare da lì, accedi a Documentazione, Guida introduttiva e scegli [Come installare](#) per le istruzioni di installazione.

Progetti AWS Lambda

AWS Lambda è un servizio di elaborazione che consente di eseguire del codice senza la necessità di effettuare il provisioning o la gestione dei server. Esegue il codice in un'infrastruttura di calcolo ad

alta disponibilità ed esegue tutte le operazioni di amministrazione delle risorse di calcolo. Per ulteriori informazioni su Lambda, consulta [Che cos'è AWS Lambda?](#) nella Guida per gli sviluppatori di AWS Lambda.

È possibile implementare funzioni Lambda tramite l'interfaccia a riga di comando (CLI) .NET.

Argomenti

- [Prerequisiti](#)
- [Comandi Lambda disponibili](#)
- [Passaggi per implementare](#)

Prerequisiti

Prima di iniziare a utilizzare l'interfaccia a riga di comando .NET per distribuire funzioni Lambda, è necessario soddisfare i seguenti prerequisiti:

- Verificare di avere installato l'interfaccia CLI .NET. Ad esempio: `dotnet --version`. Se necessario, vai a <https://dotnet.microsoft.com/download> per installarlo.
- Configurare l'interfaccia CLI .NET per l'utilizzo con Lambda. Per una descrizione su come effettuare questa operazione, consultare [Interfaccia a riga di comando di .NET Core](#) nella AWS Lambda Guida per gli sviluppatori. In tale procedura, il seguente è il comando di distribuzione:

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

In caso di dubbi su come utilizzare un ruolo IAM per questo esercizio, non includere la `--function-role role` parte. Lo strumento ti aiuterà a creare un nuovo ruolo.

Comandi Lambda disponibili

Per visualizzare un elenco dei comandi Lambda disponibili tramite l'interfaccia CLI .NET, aprire un prompt dei comandi o un terminale e digitare `dotnet lambda --help`. L'output del comando sarà simile al seguente:

```
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>
```

L'output elenca tutti i comandi attualmente disponibili.

Passaggi per implementare

Le seguenti istruzioni presuppongono che la procedura in sia stata creata `AWS Lambda.NET`. Ai fini di questa procedura, il progetto è denominato `DotNetCoreLambdaTest`.

1. Apri un prompt dei comandi o un terminale e accedi alla cartella contenente il file di progetto `.NET Lambda`.
2. Specificare (sì `dotnet lambda deploy-function`).
3. Se richiesto, immettere la `AWS Regione` (la regione in cui verrà implementata la tua funzione `Lambda`).
4. Quando richiesto, digitare il nome della funzione da distribuire, ad esempio `DotNetCoreLambdaTest`. Può essere il nome di una funzione già esistente nel tuo `Account AWS` o uno che non è ancora stato schierato lì.
5. Quando richiesto, selezionare o creare il ruolo `IAM` che `Lambda` assumerà durante l'esecuzione della funzione.

Al completamento, il `Creazione di una nuova funzione Lambda` viene visualizzato.

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

Se si distribuisce una funzione già esistente nel proprio account, la funzione di distribuzione richiede solo la `AWS Regione` (se necessario). In questo caso, l'output del comando termina con `Updating code for existing function`.

Dopo che la funzione Lambda è stata implementata, è pronta per l'uso. Per ulteriori informazioni, consulta [Esempi su come utilizzare AWS Lambda](#).

Lambda monitora automaticamente le funzioni Lambda per te e segnala i parametri tramite Amazon CloudWatch. Per monitorare e risolvere problemi relativi alla funzione Lambda, consultare [Monitoraggio e risoluzione dei problemi delle applicazioni Lambda](#).

Esegui la migrazione del tuo progetto per AWS SDK for .NET

Questa sezione fornisce informazioni sulle attività di migrazione che potrebbero riguardare l'utente e istruzioni su come eseguirle.

Argomenti

- [Cosa c'è di nuovo nel AWS SDK for .NET](#)
- [Piattaforme supportate da AWS SDK for .NET](#)
- [Migrazione alla versione 3 di AWS SDK for .NET](#)
- [Migrazione alla versione 3.5 di AWS SDK for .NET](#)
- [Migrazione alla versione 3.7 di AWS SDK for .NET](#)
- [Migrazione da .NET Standard 1.3](#)

Cosa c'è di nuovo nel AWS SDK for .NET

Consulta la pagina del prodotto all'[indirizzo https://aws.amazon.com/sdk-for-net/](https://aws.amazon.com/sdk-for-net/) per informazioni di alto livello sui nuovi sviluppi relativi a AWS SDK for .NET

Di seguito sono riportate le novità di AWS SDK for .NET.

28 marzo 2024: versione preliminare del AWS Message Processing Framework per .NET

Questa è la documentazione non definitiva per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il [AWS Message Processing Framework for .NET](#) è un framework AWS nativo che semplifica lo sviluppo di applicazioni di elaborazione di messaggi.NET che utilizzano AWS servizi come Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon EventBridge

23 febbraio 2024: è stato aggiunto il supporto per .NET 8

Il supporto per .NET 8 è stato aggiunto a AWS SDK for .NET. Utilizza i [NuGet pacchetti](#) o gli [assembly più recenti che supportano .NET 8 e versioni successive](#). Puoi trovare ulteriori informazioni su questo supporto, incluso il [supporto per Lambda](#), nel post del blog [.NET 8 Support on](#). AWS

18 febbraio 2024: imminenti modifiche al supporto di .NET Framework

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e la versione minima .NET Framework passerà alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

17 luglio 2023: Il framework AWS Lambda Annotations è stato rilasciato per la disponibilità generale

Il [framework AWS Lambda Annotations](#) rende l'esperienza di scrittura di funzioni Lambda in C# più naturale per gli sviluppatori .NET utilizzando la tecnologia di generazione di sorgenti C#. Ora è disponibile a livello generale.

15/07/23: Il Distributed Cache Provider per DynamoDB è stato rilasciato in anteprima

Questa è la documentazione preliminare per una funzionalità presente nella versione di anteprima ed è soggetta a modifiche.

La libreria Distributed Cache Provider consente di utilizzare Amazon DynamoDB come storage per il framework di cache distribuita di ASP.NET Core. [Per ulteriori informazioni, consulta il post sul blog Introduzione a AWS .NET Distributed Cache Provider for DynamoDB \(Preview\) e al repository GitHub](#)

13/07/2022: Il Deploy Tool è stato rilasciato AWS

Il AWS Deploy Tool è stato rilasciato. Questo strumento è uno strumento interattivo per l'interfaccia della riga di comando .NET e AWS Toolkit for Visual Studio consente di distribuire applicazioni .NET con conoscenze AWS minime e con il minor numero di clic o comandi. Per ulteriori informazioni, consulta [Distribuisci le applicazioni su AWS](#).

24 agosto 2020: è stata rilasciata la versione 3.5 dell'SDK

- L'esperienza .NET è stata standardizzata trasferendo il supporto per tutte le varianti non Framework dell'SDK a .NET Standard 2.0. Per ulteriori informazioni, consulta [Migalla alla versione 3.5](#).
- Sono stati aggiunti degli impaginatori a molti client di servizio, che semplificano l'impaginazione dei risultati delle API. Per ulteriori informazioni, consulta [Impaginatori](#).

Piattaforme supportate da AWS SDK for .NET

AWS SDK for .NET fornisce gruppi di assembly separati per gli sviluppatori destinati alle diverse piattaforme. Tuttavia, non tutte le funzionalità SDK sono presenti in tutte le singole piattaforme. Questo argomento descrive le differenze nel supporto di ciascuna piattaforma.

.NET Core

AWS SDK for .NET Supporta applicazioni scritte per .NET Core (.NET Core 3.1, .NET 5, .NET 6 e così via). AWSi client di servizio supportano solo modelli di chiamata asincroni in .NET core. Ciò influisce anche su molte delle astrazioni di alto livello create su client di servizio, come Amazon TransferUtility S3, che supporterà solo chiamate asincrone nell'ambiente .NET Core.

.NET Standard 2.0

Le varianti non Framework AWS SDK for .NET sono conformi [a .NET Standard 2.0](#). AWS SDK for .NET Fornisce solo metodi asincroni per applicazioni scritte su .NET Standard.

.NET Framework 4.5

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e la versione minima di .NET Framework passerà alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

Questa versione di AWS SDK for .NET è compilata su .NET Framework 4.5 e viene eseguita nel runtime di .NET 4.0. AWSi [client di servizio supportano modelli di chiamata sincroni e asincroni e utilizzano le parole chiave async e await introdotte in C# 5.0](#).

.NET Framework 3.5

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e cambierà la versione minima di .NET Framework alla 4.6.2. Per ulteriori informazioni,

consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

Questa versione di AWS SDK for .NET è compilata su .NET Framework 3.5 e viene eseguita nel runtime .NET 2.0 o .NET 4.0. AWSi client di servizio supportano modelli di chiamata sincroni e asincroni e utilizzano i precedenti pattern Begin e End.

Note

AWS SDK for .NET non è conforme allo standard FIPS (Federal Information Processing Standard) quando viene utilizzato da applicazioni create con la versione 2.0 di CLR. Per informazioni dettagliate su come sostituire un'implementazione conforme a FIPS in tale ambiente, consultate il blog di Microsoft e la classe HMACSHA256 (HMACSHA256cng) del team di [sicurezza CLR](#) in Security.Cryptography.dll. [CryptoConfig](#)

Portable Class Library e Xamarin

AWS SDK for .NET contiene anche un'implementazione della libreria di classi portabile. L'implementazione della Portable Class Library può essere indirizzata a più piattaforme, tra cui Universal Windows Platform (UWP) e Xamarin su iOS e Android. Per ulteriori dettagli, consulta [Mobile SDK for .NET e Xamarin](#). AWSi client di servizio supportano solo modelli di chiamata asincroni.

Supporto Unity

Per informazioni sul supporto di Unity, consulta [Considerazioni speciali per il supporto di Unity](#).

Ulteriori informazioni

[Migaglia alla versione 3.5 di AWS SDK for .NET](#)

Migrazione alla versione 3 di AWS SDK for .NET

Questo argomento descrive le modifiche della versione 3 di AWS SDK for .NET e come migrare il codice a questa versione dell'SDK.

Informazioni sulle versioni di AWS SDK for .NET

L'AWS SDK for .NET, originariamente rilasciato nel novembre del 2009, è stato progettato per .NET Framework 2.0. Da allora, .NET è stato migliorato con le versioni .NET Framework 4.0 e .NET Framework 4.5 e ha aggiunto nuove piattaforme di destinazione: WinRT e Windows Phone.

La versione 2 di AWS SDK for .NET è stata aggiornata per sfruttare le nuove caratteristiche della piattaforma .NET e per puntare a WinRT e Windows Phone.

La versione 3 di AWS SDK for .NET è stata aggiornata per rendere gli assembly modulari.

Riprogettazione dell'architettura per l'SDK

L'intera versione 3 di AWS SDK for .NET è stata riprogettata per essere modulare. Ogni servizio è ora implementato nel relativo assembly, anziché in un unico assembly globale. Non dovrai più aggiungere l'intero AWS SDK for .NET alla tua applicazione. È ora possibile aggiungere assembly solo per i servizi utilizzati dalla tua applicazione.

Modifiche importanti

Le seguenti sezioni descrivono le modifiche alla versione 3 di AWS SDK for .NET.

AWSClientFactory rimossa

La classe `Amazon.AWSClientFactory` è stata rimossa. Ora, per creare un client di servizio, utilizza il costruttore del client di servizio. Ad esempio, per creare `AmazonEC2Client`:

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

Amazon.Runtime.AssumeRoleAWSCredentials rimossa

La classe `Amazon.Runtime.AssumeRoleAWSCredentials` è stata rimossa perché si trovava in uno spazio dei nomi principale ma aveva una dipendenza da AWS Security Token Service e perché da qualche tempo era obsoleta nell'SDK. Al suo posto, utilizza la classe `Amazon.SecurityToken.AssumeRoleAWSCredentials`.

Metodo SetACL rimosso da S3Link

La classe `S3Link` fa parte della classe `Amazon.DynamoDBv2` e viene utilizzato per archiviare in Amazon S3 gli oggetti che sono riferimenti in un articolo DynamoDB. Questa è una funzionalità utile, ma

non volevamo creare una dipendenza di compilazione sul pacchetto Amazon.S3 per DynamoDB. Di conseguenza, abbiamo semplificato i metodi esposti Amazon.S3 dalla classe S3Link, sostituendo il metodo SetACL con il metodo MakeS3ObjectPublic. Per un maggiore controllo sulla lista di controllo accessi (ACL) sull'oggetto, utilizza direttamente il pacchetto Amazon.S3.

Rimozione delle classi risultanti obsolete

Per la maggior parte dei servizi di AWS SDK for .NET, le operazioni restituiscono un oggetto di risposta contenente i metadati per l'operazione, ad esempio l'ID della richiesta e un oggetto risultante. Avere una risposta separata e una classe risultante era ridondante e creava un'ulteriore digitazione per gli sviluppatori. Nella versione 2 di AWS SDK for .NET, abbiamo inserito tutte le informazioni della classe risultante nella classe di risposta. Abbiamo inoltre contrassegnato come obsolete le classi risultanti per scoraggiarne l'uso. Nella versione 3 di AWS SDK for .NET, abbiamo rimosso queste classi risultanti obsolete per ridurre le dimensioni dell'SDK.

AWSSModifiche alla sezione di Config

È possibile effettuare la configurazione avanzata di AWS SDK for .NET mediante il file App.config o Web.config. A tale scopo viene utilizzata una sezione di configurazione <aws> come quella riportata di seguito, che fa riferimento al nome dell'assembly SDK.

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

Nella versione 3 di AWS SDK for .NET, l'assembly AWSSDK non esiste più. Abbiamo inserito il codice comune nell'assembly AWSSDK.Core. Di conseguenza, dovrai modificare i riferimenti all'assembly AWSSDK nel file App.config o Web.config all'assembly AWSSDK.Core, come riportato di seguito.

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

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

Puoi anche modificare le impostazioni di configurazione mediante la classe `Amazon.AWSConfigs`. Nella versione 3 dell'AWS SDK for .NET, abbiamo trasferito le impostazioni di configurazione per DynamoDB dalla `Amazon.AWSConfigs` classe alla `Amazon.AWSConfigsDynamoDB` classe.

Migalla alla versione 3.5 di AWS SDK for .NET

La versione 3.5 di AWS SDK for .NET standardizza ulteriormente l'esperienza .NET passando il supporto per tutte le varianti non Framework dell'SDK a [.NET Standard 2.0](#). A seconda dell'ambiente e della base di codice, per sfruttare le funzionalità della versione 3.5, potrebbe essere necessario eseguire determinati lavori di migrazione.

In questo argomento vengono descritte le modifiche apportate alla versione 3.5 e le possibili operazioni da eseguire per la migrazione dell'ambiente o del codice dalla versione 3.

Cosa è cambiato per la versione 3.5

Di seguito viene descritto ciò che è o non è cambiato in AWS SDK for .NET versione 3.5.

.NET Framework e .NET Core

Il supporto per .NET Framework e .NET Core non è cambiato.

Xamarin

I progetti Xamarin (nuovi ed esistenti) devono essere indirizzati a .NET Standard 2.0. Vedere il [supporto .NET Standard 2.0 in Xamarin.Forms](#) e il [supporto di implementazione .NET](#).

Unità

Le app Unity devono essere indirizzate ai profili .NET Standard 2.0 o .NET 4.x utilizzando Unity 2018.1 o versioni successive. Per ulteriori informazioni, vedere il [supporto del profilo .NET](#). Inoltre, se viene utilizzato IL2CPP per compilare, è necessario disabilitare la rimozione del codice aggiungendo `unlink.xml`, come descritto in [Riferimenti a AWS SDK for .NET Standard 2.0 di Unity, Xamarin o UWP](#). Dopo aver eseguito il port del codice in una delle basi di codice consigliate, l'app Unity può accedere a tutti i servizi offerti dall'SDK.

Poiché Unity supporta .NET Standard 2.0, il pacchetto dell'AWS SDK versione 3.5 non dispone più di codice specifico di unità, incluse alcune funzionalità di livello superiore. Per fornire una transizione migliore, tutto il codice Unity è disponibile per riferimento nella [aw/aws-sdk-unity-net](https://github.com/aws/aws-sdk-unity-net) GitHub repository. Se si riscontrano funzionalità mancanti che influiscono sull'utilizzo di AWS con Unity, è possibile presentare una richiesta di funzionalità all'indirizzo <https://github.com/aws/dotnet/issues>.

Consulta anche [Considerazioni speciali per il supporto di Unity](#).

Universal Windows Platform (UWP)

Destina la tua applicazione UWP alla [versione 16299 o successiva](#) (aggiornamento Fall Creators, versione 1709, rilasciata a ottobre 2017).

Windows Phone e Silverlight

La versione 3.5 di AWS SDK for .NET non supporta queste piattaforme perché Microsoft non le sta più sviluppando attivamente. Per ulteriori informazioni, consulta gli argomenti seguenti:

- [Fine del supporto per Windows 10 Mobile](#)
- [Fine del supporto per Silverlight](#)

Librerie di classi portatili legacy (PCL basati su profili)

Considera il retargeting della tua libreria in .NET Standard. Per ulteriori informazioni, vedere [Confronto con le librerie di classi portatili](#) di Microsoft.

Amazon Cognito Sync Manager e Amazon Mobile Analytics Manager

Astrazioni di alto livello che facilitano l'utilizzo di Amazon Cognito Sync e Amazon Mobile Analytics vengono rimosse dalla versione 3.5 di AWS SDK for .NET. AWS AppSync è il sostituto preferito per Amazon Cognito Sync. Amazon Pinpoint è il sostituto preferito di Amazon Mobile Analytics.

Se il tuo codice è influenzato dalla mancanza di codice di libreria di livello superiore per AWS AppSync e Amazon Pinpoint, puoi registrare il tuo interesse in uno o entrambi i seguenti formati GitHub problemi: <https://github.com/aws/dotnet/issues/20> e <https://github.com/aws/dotnet/issues/19>. Puoi anche ottenere le librerie per Amazon Cognito Sync Manager e Amazon Mobile Analytics Manager dai seguenti GitHub repository: [aw/amazon-cognito-sync-manager-net](https://github.com/aws/amazon-cognito-sync-manager-net) e [aw/aws-mobile-analytics-manager-net](https://github.com/aws/mobile-analytics-manager-net).

Migrazione del codice sincrono

Versione 3.5 di AWS SDK for .NET supporta sia .NET Framework che .NET Standard (tramite le versioni .NET Core come .NET core 3.1, .NET 5 e così via). Le varianti dell'SDK conformi a .NET Standard forniscono solo metodi asincroni, pertanto se si desidera sfruttare .NET Standard, è necessario modificare il codice sincrono in modo che venga eseguito in modo asincrono.

I frammenti di codice riportati di seguito mostrano come è possibile modificare il codice sincrono in codice asincrono. Il codice in questi frammenti viene utilizzato per visualizzare il numero di bucket Amazon S3.

Il codice originale chiama [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}");
```

Per utilizzare la versione 3.5 dell'SDK, chiamare [ListBucketsAsync](#) anziché.

```
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}");
```

Migrazione alla versione 3.7 dell'AWS SDK for .NET

alla versione 3.7, l'AWS SDK for .NET non supporta più .NET Standard 1.3.

Per informazioni sulla migrazione da .NET Standard 1.3, vedere [Migrazione da .NET Standard 1.3](#).

Migrazione da .NET Standard 1.3

Il 27 giugno 2019 Microsoft [ha chiuso il supporto](#) per le versioni di NET Core 1.0 e NET Core 1.1. A seguito di questo annuncio, AWS ha terminato il supporto per .NET Standard 1.3 su AWS SDK for .NET il 31 dicembre 2020.

AWS ha continuato a fornire aggiornamenti del servizio e correzioni di sicurezza sull'AWS SDK for .NET indirizzate a NET Standard 1.3 fino al 1° ottobre 2020. Dopo tale data, la destinazione NET Standard 1.3 entrò in modalità di manutenzione, il che significava che non venivano rilasciati nuovi aggiornamenti; AWS ha applicato solo correzioni di bug critici e patch di sicurezza.

Il 31 dicembre 2020, il supporto per .NET Standard 1.3 sull'AWS SDK for .NET arrivò alla fine della vita. Dopo tale data non sono state applicate correzioni di bug o patch di sicurezza. Gli artefatti creati con quel target rimangono disponibili per il download su NuGet.

Cosa occorre sapere

- Se si eseguono applicazioni che utilizzano .NET Framework, non si è interessati.
- Se si eseguono applicazioni che utilizzano .NET Core 2.0 o versioni successive, non si è interessati.
- Se si eseguono applicazioni che utilizzano .NET Core 1.0 o .NET Core 1.1, eseguire la migrazione delle applicazioni a una versione più recente di .NET Core seguendo le [istruzioni di migrazione di Microsoft](#). Si consiglia un minimo di .NET Core 3.1.
- Se si eseguono applicazioni business critical che non possono essere aggiornate in questo momento, è possibile continuare a utilizzare la versione corrente di AWS SDK for .NET.

In caso di domande o dubbi, [contattare il supporto tecnico AWS](#).

Lavora con AWS i servizi in AWS SDK for .NET

Le sezioni seguenti contengono esempi, tutorial, attività e guide che mostrano come utilizzarli per AWS SDK for .NET lavorare con AWS i servizi. Questi esempi e tutorial si basano su un'API che fornisce. AWS SDK for .NET Per vedere quali classi e metodi sono disponibili nell'API, consulta [l'AWS SDK for .NET API Reference](#).

Se non lo conosci AWS SDK for .NET, ti consigliamo di dare prima un'occhiata all'[Fai un breve tour](#) argomento. Fornisce un'introduzione all'SDK.

Puoi trovare altri esempi di codice nel Code [Examples Repository](#) e nel [repository awslabs su AWS GitHub](#)

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

- [Esempi di codice con linee guida per il AWS SDK for .NET](#)
- [Utilizzo AWS Lambda per il servizio di calcolo](#)
- [Librerie e framework di alto livello per AWS SDK for .NET](#)
- [Programmazione AWS OpsWorks per lavorare con stack e applicazioni](#)
- [Support per altri AWS servizi e configurazione](#)

Esempi di codice con linee guida per il AWS SDK for .NET

Le sezioni seguenti contengono esempi di codice e forniscono indicazioni per gli esempi. Possono aiutarti a imparare a utilizzarli AWS SDK for .NET per lavorare con AWS i servizi.

Se non lo conosci AWS SDK for .NET, ti consigliamo di dare prima un'occhiata all'[Fai un breve tour](#) argomento. Fornisce un'introduzione all'SDK.

Prima di iniziare, assicuratevi di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

- [Accesso AWS CloudFormation con AWS SDK for .NET](#)

- [Autenticazione degli utenti con Amazon Cognito](#)
- [Utilizzo di database Amazon DynamoDB NoSQL](#)
- [Utilizzo di Amazon EC2](#)
- [Accesso AWS Identity and Access Management \(IAM\) con AWS SDK for .NET](#)
- [Utilizzo dello storage Internet di Amazon Simple Storage Service](#)
- [Invio di notifiche dal cloud utilizzando Amazon Simple Notification Service](#)
- [Messaggistica tramite Amazon SQS](#)

Accesso AWS CloudFormation con AWS SDK for .NET

I AWS SDK for .NET supporti [AWS CloudFormation](#), che creano e forniscono implementazioni dell'AWS infrastruttura in modo prevedibile e ripetuto.

API

AWS SDK for .NET Fornisce API per i clienti. AWS CloudFormation Le API consentono di lavorare con AWS CloudFormation funzionalità come modelli e stack. Questa sezione contiene un numero limitato di esempi che mostrano i modelli che è possibile seguire quando si lavora con queste API. Per visualizzare il set completo di API, consulta l'[AWS SDK for .NET API Reference](#) (e scorri fino a «Amazon. CloudFormation»).

[Le AWS CloudFormation API sono fornite da. AWSSDK CloudFormation](#)pacchetto.

Prerequisiti

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

Argomenti

- [AWS Elencare le risorse utilizzando AWS CloudFormation](#)

AWS Elencare le risorse utilizzando AWS CloudFormation

Questo esempio mostra come usare per AWS SDK for .NET elencare le risorse in AWS CloudFormation pile. L'esempio utilizza l'API di basso livello. L'applicazione non accetta argomenti,

ma raccoglie semplicemente informazioni per tutti gli stack accessibili alle credenziali dell'utente e quindi visualizza le informazioni su tali stack.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.CloudFormation](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon. CloudFormation](#)

Classe [AmazonCloudFormationClient](#)

- [Spazio dei nomi Amazon. CloudFormation](#).Modello

[Classe I. CloudFormationPaginatorFactory DescribeStacks](#)

Classe [DescribeStackResourcesRequest](#)

Classe [DescribeStackResourcesResponse](#)

[Stack](#) di classi

Classe [StackResource](#)

[Tag](#) di classe

```
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;  
    }  
}
```

Autenticazione degli utenti con Amazon Cognito

Note

Le informazioni contenute in questo argomento si riferiscono specificamente ai progetti basati su .NET Framework e alla AWS SDK for .NET versione 3.3 e precedenti.

Utilizzando Amazon Cognito Identity, puoi creare identità uniche per i tuoi utenti e autenticarli per un accesso sicuro alle tue AWS risorse come Amazon S3 o Amazon DynamoDB. Amazon Cognito Identity supporta provider di identità pubblici come Amazon, Facebook, Twitter/Digits, Google o qualsiasi provider compatibile con OpenID Connect, nonché identità non autenticate. Amazon Cognito supporta anche [identità autenticate dagli sviluppatori](#), che consentono di registrare e autenticare gli utenti utilizzando il proprio processo di autenticazione back-end, continuando a utilizzare Amazon Cognito Sync per sincronizzare i dati degli utenti e accedere alle risorse. AWS

Per ulteriori informazioni su [Amazon Cognito](#), consulta la [Amazon Cognito Developer Guide](#).

I seguenti esempi di codice mostrano come usare facilmente Amazon Cognito Identity. L'[Provider di credenziali](#) esempio mostra come creare e autenticare le identità degli utenti. L'[CognitoAuthentication libreria di estensioni](#) esempio mostra come utilizzare la libreria di CognitoAuthentication estensioni per autenticare i pool di utenti di Amazon Cognito.

Argomenti

- [Provider di credenziali Amazon Cognito](#)
- [Esempi di librerie di CognitoAuthentication estensioni Amazon](#)

Provider di credenziali Amazon Cognito

Note

Le informazioni contenute in questo argomento si riferiscono specificamente ai progetti basati su .NET Framework e alla AWS SDK for .NET versione 3.3 e precedenti.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, disponibile in [AWSSDK.CognitoIdentity](#) NuGetpackage, è un oggetto di credenziali che utilizza Amazon Cognito e AWS Security Token Service the AWS STS() per recuperare le credenziali per effettuare chiamate. AWS

Il primo passo per la configurazione di `CognitoAWSCredentials` è creare un "pool di identità". (Un pool di identità è un archivio di informazioni sull'identità dell'utente specifico per il tuo account. Le informazioni sono recuperabili su più piattaforme client, dispositivi e sistemi operativi, in modo tale che, se un utente inizia a usare l'applicazione su un telefono e successivamente su un tablet, le informazioni sull'applicazione restino a disposizione di quell'utente. Puoi creare un nuovo pool di identità dalla console Amazon Cognito. Se stai utilizzando la console, ti verranno fornite le altre informazioni necessarie:

- Il tuo numero account: un numero di 12 cifre, ad esempio 123456789012, univoco per il tuo account.
- L'ARN del ruolo non autenticato: un ruolo che verrà assunto dagli utenti non autenticati. Ad esempio, questo ruolo è in grado di fornire le autorizzazioni di sola lettura per i dati.
- L'ARN del ruolo autenticato: un ruolo che verrà assunto dagli utenti autenticati. Questo ruolo è in grado di fornire autorizzazioni più estese per i dati.

Configura Cognito AWSCredentials

Il seguente esempio di codice mostra come eseguire la configurazione `CognitoAWSCredentials`, che puoi quindi utilizzare per effettuare una chiamata ad Amazon S3 come utente non autenticato. In questo modo è possibile effettuare chiamate con solo una quantità minima dei dati richiesti per autenticare l'utente. Le autorizzazioni utente sono controllate dal ruolo, perciò è possibile configurare l'accesso necessario.

```
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();
}
```

Utilizza AWS come utente non autenticato

Il seguente esempio di codice mostra come iniziare a utilizzare AWS come utente non autenticato, quindi autenticarsi tramite Facebook e aggiornare le credenziali per utilizzare le credenziali di Facebook. Tramite questo approccio, è possibile concedere diverse funzionalità agli utenti autenticati tramite il ruolo autenticato. Ad esempio, sarebbe possibile avere un'applicazione per telefono che consente agli utenti di visualizzare i contenuti in modo anonimo, ma permette loro di pubblicare se connessi a uno o più provider configurati.

```
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();
}
```

L'oggetto `CognitoAWSCredentials` offre anche più funzionalità se utilizzato con il `AmazonCognitoSyncClient` che fa parte di AWS SDK for .NET. Se si utilizzano entrambi `AmazonCognitoSyncClient` e `CognitoAWSCredentials`, non è necessario specificare

le `IdentityId` proprietà `IdentityPoolId` e quando si effettuano chiamate con `AmazonCognitoSyncClient` Queste proprietà vengono automaticamente riempite da `CognitoAWSCredentials`. Il prossimo codice illustra quanto sopra, nonché un evento che notifica ogni modifica dell'`IdentityId` per `CognitoAWSCredentials`. Il codice `IdentityId` può variare in alcuni casi, ad esempio quando si passa da un utente non autenticato a uno autenticato.

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

Esempi di librerie di CognitoAuthentication estensioni Amazon

Note

Le informazioni in questo argomento sono specifiche per i progetti basati su .NET Framework e la AWS SDK for .NET versione 3.3 e precedenti.

La libreria di CognitoAuthentication estensioni, disponibile in [Amazon.Extensions.CognitoAuthentication](#) NuGet pacchetto, semplifica il processo di autenticazione dei pool di utenti di Amazon Cognito per gli sviluppatori .NET Core e Xamarin. La libreria si basa sull'API del provider di identità Amazon Cognito per creare e inviare chiamate API di autenticazione degli utenti.

Utilizzo della libreria di CognitoAuthentication estensioni

Amazon Cognito dispone di alcuni ChallengeName valori AuthFlow e funzionalità integrati per un flusso di autenticazione standard per convalidare nome utente e password tramite Secure Remote Password (SRP). Per ulteriori informazioni sul flusso di autenticazione, consulta [Flusso di autenticazione del pool di utenti Amazon Cognito](#).

I seguenti esempi richiedono le seguenti istruzioni using:

```
// Required for all examples
using System;
using Amazon;
using Amazon.CognitoIdentity;
using Amazon.CognitoIdentityProvider;
using Amazon.Extensions.CognitoAuthentication;
using Amazon.Runtime;
// Required for the GetS3BucketsAsync example
using Amazon.S3;
using Amazon.S3.Model;
```

Usa l'autenticazione di base

Crea un [AmazonCognitoIdentityProviderClient](#) account [anonimo](#) utilizzando `AWSCredentials`, che non richiede richieste firmate. Non è necessario fornire una regione, il codice sottostante chiamerà `FallbackRegionFactory.GetRegionEndpoint()` se non viene fornita una regione. Creare `CognitoUserPool` e oggetti `CognitoUser`. Chiamare il metodo `StartWithSrpAuthAsync` con un codice `InitiateSrpAuthRequest` che contiene le password utente.

```
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;
}
```

Autentica con sfide

Inoltre, è più semplice continuare il flusso di autenticazione anche in presenza di problematiche, ad esempio con `NewPasswordRequired` Multi-Factor Authentication (MFA). Gli unici requisiti sono `CognitoAuthentication` gli oggetti, la password dell'utente per SRP e le informazioni necessarie per la sfida successiva, che viene acquisita dopo aver richiesto all'utente di inserirla. Il codice seguente mostra un modo per verificare il tipo di sfida e ottenere le risposte appropriate per l'MFA e `NewPasswordRequired` le sfide durante il flusso di autenticazione.

Effettuare una richiesta di autenticazione di base come in precedenza e `await` una `AuthFlowResponse`. Quando la risposta è ricevuta in loop attraverso l'oggetto restituito `AuthenticationResult`. Se il tipo `ChallengeName` è `NEW_PASSWORD_REQUIRED`, chiamare il metodo `RespondToNewPasswordRequiredAsync`.

```
public static async void GetCredsChallengesAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
    Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest(){
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
    user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);

    while (authResponse.AuthenticationResult == null)
    {
        if (authResponse.ChallengeName == ChallengeNameType.NEW_PASSWORD_REQUIRED)
        {
            Console.WriteLine("Enter your desired new password:");
            string newPassword = Console.ReadLine();

            authResponse = await user.RespondToNewPasswordRequiredAsync(new
    RespondToNewPasswordRequiredRequest()
            {
```

```
        SessionID = authResponse.SessionID,
        NewPassword = newPassword
    });
    accessToken = authResponse.AuthenticationResult.AccessToken;
}
else if (authResponse.ChallengeName == ChallengeNameType.SMS_MFA)
{
    Console.WriteLine("Enter the MFA Code sent to your device:");
    string mfaCode = Console.ReadLine();

    AuthFlowResponse mfaResponse = await user.RespondToSmsMfaAuthAsync(new
RespondToSmsMfaRequest()
    {
        SessionID = authResponse.SessionID,
        MfaCode = mfaCode

    }).ConfigureAwait(false);
    accessToken = authResponse.AuthenticationResult.AccessToken;
}
else
{
    Console.WriteLine("Unrecognized authentication challenge.");
    accessToken = "";
    break;
}
}

if (authResponse.AuthenticationResult != null)
{
    Console.WriteLine("User successfully authenticated.");
}
else
{
    Console.WriteLine("Error in authentication process.");
}
}
```

Utilizza AWS le risorse dopo l'autenticazione

Una volta che un utente è autenticato utilizzando la `CognitoAuthentication` libreria, il passaggio successivo consiste nel consentire all'utente di accedere alle AWS risorse appropriate. A tale scopo è necessario creare un pool di identità tramite la console Amazon Cognito Federated Identities.

Specificando il pool di utenti di Amazon Cognito che hai creato come provider, utilizzando i relativi PoolID e ClientID, puoi consentire agli utenti del pool di utenti di Amazon Cognito di accedere alle risorse collegate al tuo account. AWS È anche possibile specificare diversi ruoli per consentire a entrambi gli utenti autenticati e non autenticati di accedere alle diverse risorse. È possibile modificare queste regole nella console IAM, dove è possibile aggiungere o rimuovere le autorizzazioni nel campoAction (Operazione) della policy collegata del ruolo. Quindi, utilizzando il pool di identità, il pool di utenti e le informazioni utente di Amazon Cognito appropriati, puoi effettuare chiamate a risorse diverse AWS . L'esempio seguente mostra un utente autenticato con SRP che accede ai diversi bucket Amazon S3 consentiti dal ruolo del pool di identità associato.

```
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);
        }
    }
}
```

Altre opzioni di autenticazione

Oltre a SRP e MFA NewPasswordRequired, CognitoAuthentication la libreria di estensioni offre un flusso di autenticazione più semplice per:

- Custom - Iniziare con una chiamata a `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Inizia con una chiamata a `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Inizia con una chiamata a `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Inizia con una chiamata a `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Chiamare il metodo appropriato in funzione del flusso desiderato. Quindi continuare a richiedere all'utente le sfide presentate negli oggetti `AuthFlowResponse` di ciascuna chiamata al metodo. Chiamare il metodo di risposta appropriata, ad esempio `RespondToSmsMfaAuthAsync` per le sfide MFA e `RespondToCustomAuthAsync` per le sfide personalizzate.

Utilizzo di database Amazon DynamoDB NoSQL

Note

I modelli di programmazione descritti in questi argomenti sono presenti sia in .NET Framework che in .NET (Core), ma le convenzioni di chiamata sono diverse, siano esse sincrone o asincrone.

AWS SDK for .NET Supporta Amazon DynamoDB, un servizio di database NoSQL veloce offerto da AWS. L'SDK fornisce tre modelli di programmazione per comunicare con DynamoDB: il modello di basso livello, il modello di documento e il modello di persistenza degli oggetti.

Le seguenti informazioni introducono questi modelli e le relative API, forniscono esempi su come e quando utilizzarli e forniscono collegamenti a risorse di programmazione DynamoDB aggiuntive disponibili in. AWS SDK for .NET

Argomenti

- [Modello di basso livello](#)
- [Modello di documento](#)
- [Modello di persistenza degli oggetti](#)
- [Ulteriori informazioni](#)
- [Utilizzo di espressioni con Amazon DynamoDB e AWS SDK for .NET](#)
- [Supporto JSON in Amazon DynamoDB](#)

Modello di basso livello

Il modello di programmazione di basso livello include le chiamate dirette al servizio DynamoDB. Puoi accedere a questo modello attraverso lo spazio dei nomi [Amazon.DynamoDBv2](#).

Dei tre modelli, il modello di basso livello richiede di scrivere la maggior parte del codice. Ad esempio, è necessario convertire i tipi di dati.NET nei loro equivalenti in DynamoDB. Tuttavia, questo modello consente di accedere alla maggior parte delle caratteristiche.

Gli esempi seguenti mostrano come utilizzare il modello di basso livello per creare una tabella, modificare una tabella e inserire elementi in una tabella in DynamoDB.

Creazione di una tabella

Nell'esempio seguente, è necessario creare una tabella utilizzando il metodo `CreateTable` della classe `AmazonDynamoDBClient`. Il metodo `CreateTable` utilizza un'istanza della classe `CreateTableRequest` che contiene caratteristiche quali i nomi degli attributi degli item richiesti, la definizione della chiave primaria e la capacità di throughput. Il metodo `CreateTable` restituisce un'istanza della classe `CreateTableResponse`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

Console.WriteLine("Getting list of tables");
List<string> currentTables = client.ListTables().TableNames;
```

```
Console.WriteLine("Number of tables: " + currentTables.Count);
if (!currentTables.Contains("AnimalsInventory"))
{
    var request = new CreateTableRequest
    {
        TableName = "AnimalsInventory",
        AttributeDefinitions = new List<AttributeDefinition>
        {
            new AttributeDefinition
            {
                AttributeName = "Id",
                // "S" = string, "N" = number, and so on.
                AttributeType = "N"
            },
            new AttributeDefinition
            {
                AttributeName = "Type",
                AttributeType = "S"
            }
        },
        KeySchema = new List<KeySchemaElement>
        {
            new KeySchemaElement
            {
                AttributeName = "Id",
                // "HASH" = hash key, "RANGE" = range key.
                KeyType = "HASH"
            },
            new KeySchemaElement
            {
                AttributeName = "Type",
                KeyType = "RANGE"
            }
        },
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = 10,
            WriteCapacityUnits = 5
        },
    };

    var response = client.CreateTable(request);

    Console.WriteLine("Table created with request ID: " +
```

```
    response.ResponseMetadata.RequestId);  
}
```

Verificare che una tabella sia pronta per la modifica

Prima di poterla modificare, una tabella deve essere pronta per la modifica. L'esempio seguente mostra come utilizzare il modello di basso livello per verificare che una tabella in DynamoDB sia pronta. In questo esempio, la tabella di destinazione da verificare viene individuata tramite il metodo `DescribeTable` della classe `AmazonDynamoDBClient`. Ogni cinque secondi, il codice verifica il valore della proprietà `TableStatus` della tabella. Quando lo stato è impostato su `ACTIVE`, la tabella è pronta per essere modificata.

```
// 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);
```

Inserimento di un item in una tabella

Nell'esempio seguente, si utilizza il modello di basso livello per inserire due elementi in una tabella in DynamoDB. Ciascun item è inserito tramite il metodo `PutItem` della classe `AmazonDynamoDBClient`, utilizzando un'istanza della classe `PutItemRequest`. Ognuna delle due istanze della classe `PutItemRequest` richiede il nome della tabella in cui gli item verranno inseriti, con una serie di valori degli attributi degli item.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
    TableName = "AnimalsInventory",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "1" } },
        { "Type", new AttributeValue { S = "Dog" } },
        { "Name", new AttributeValue { S = "Fido" } }
    }
};

var request2 = new PutItemRequest
{
    TableName = "AnimalsInventory",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "2" } },
        { "Type", new AttributeValue { S = "Cat" } },
        { "Name", new AttributeValue { S = "Patches" } }
    }
};

client.PutItem(request1);
client.PutItem(request2);
```

Modello di documento

Il modello di programmazione dei documenti offre un modo più semplice per lavorare con i dati in DynamoDB. Questo modello è progettato in modo specifico per l'accesso alle tabelle e agli item nelle tabelle. [È possibile accedere a questo modello tramite `amazon.DynamoDBv2.DocumentModel` namespace.](#)

Rispetto al modello di programmazione di basso livello, il modello documentale è più facile da codificare con i dati DynamoDB. Ad esempio, non è necessario convertire tanti tipi di dati .NET nei rispettivi equivalenti in DynamoDB. Tuttavia, questo modello non consente l'accesso a tutte le caratteristiche del modello di programmazione di basso livello. Ad esempio, è possibile usare questo modello per creare, recuperare, aggiornare ed eliminare gli item nelle tabelle. Tuttavia, per creare le tabelle, è necessario utilizzare il modello di basso livello. Rispetto al modello di persistenza dell'oggetto, questo modello richiede la scrittura di codice supplementare per memorizzare, caricare ed eseguire la query degli oggetti .NET.

Per ulteriori informazioni sul modello di programmazione documentale di DynamoDB, [consulta.NET: Document](#) model nella Amazon [DynamoDB](#) Developer Guide.

Le sezioni seguenti forniscono informazioni su come creare una rappresentazione della tabella DynamoDB desiderata ed esempi su come utilizzare il modello di documento per inserire elementi nelle tabelle e ottenere elementi dalle tabelle.

Creare una rappresentazione della tabella

Per eseguire operazioni sui dati utilizzando il modello di documento, dovete prima creare un'istanza della `Table` classe che rappresenti una tabella specifica. Esistono due modi principali per eseguire questa operazione.

LoadTable metodo

Il primo meccanismo consiste nell'utilizzare uno dei `LoadTable` metodi statici della `Table` classe, simile all'esempio seguente:

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

Note

Sebbene questo meccanismo funzioni, in determinate condizioni, a volte può portare a latenze o deadlock aggiuntivi dovuti a comportamenti di avvio a freddo e pool di thread. Per

ulteriori informazioni su questi comportamenti, consulta il post del blog [Improved DynamoDB Initialization Patterns](#) for the. AWS SDK for .NET

TableBuilder

Un meccanismo alternativo, la [TableBuilder](#) classe, è stato introdotto nella [versione 3.7.203](#) del pacchetto. `DynamoDBv2`. `AWSSDK NuGet` Questo meccanismo può risolvere i comportamenti sopra menzionati rimuovendo alcune chiamate implicite ai metodi, in particolare il metodo.

`DescribeTable` Questo meccanismo viene utilizzato in modo simile all'esempio seguente:

```
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();
```

Per ulteriori informazioni su questo meccanismo alternativo, consulta nuovamente il post sul blog [Improved DynamoDB Initialization Patterns](#) for the. AWS SDK for .NET

Inserimento di un elemento in una tabella

Nell'esempio seguente, una risposta viene inserita nella tabella `Reply` tramite il `PutItemAsync` metodo della `Table` classe. Il metodo `PutItemAsync` richiede un'istanza della classe `Document`; la classe `Document` è semplicemente una raccolta di attributi inizializzati.

```
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);
```

Ottenere un elemento da una tabella

Nell'esempio seguente, una risposta viene recuperata tramite il `GetItemAsync` metodo della `Table` classe. Per determinare la risposta da ottenere, il `GetItemAsync` metodo utilizza la chiave hash-and-range primaria della risposta di destinazione.

```
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"]);
```

L'esempio precedente converte implicitamente i valori della tabella in stringhe per il metodo.

`WriteLine` È possibile eseguire conversioni esplicite utilizzando i vari metodi «As [type]» della classe. `DynamoDBEntry` Ad esempio, puoi convertire in modo esplicito il valore di `Id` da un tipo di `Primitive` dati a un GUID tramite il metodo: `AsGuid()`

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

Modello di persistenza degli oggetti

Il modello di programmazione della persistenza degli oggetti è progettato specificamente per l'archiviazione, il caricamento e l'interrogazione di oggetti.NET in DynamoDB. [È possibile accedere a questo modello tramite Amazon.DynamoDBV2. DataModel namespace.](#)

Dei tre modelli, il modello di persistenza degli oggetti è il più semplice da utilizzare come codice ogni volta che si archiviano, caricano o interrogano dati DynamoDB. Ad esempio, lavori direttamente con i tipi di dati DynamoDB. Tuttavia, questo modello fornisce l'accesso solo alle operazioni che archiviano, caricano e interrogano oggetti.NET in DynamoDB. Ad esempio, è possibile usare questo modello per creare, recuperare, aggiornare ed eliminare gli item nelle tabelle. Tuttavia, è necessario

prima creare le tabelle utilizzando il modello di basso livello e quindi utilizzare questo modello per mappare le classi .NET alle tabelle.

[Per ulteriori informazioni sul modello di programmazione della persistenza degli oggetti DynamoDB, consulta.NET: Object persistence model nella Amazon DynamoDB Developer Guide.](#)

Gli esempi seguenti mostrano come definire una classe.NET che rappresenta un elemento di DynamoDB, utilizzare un'istanza della classe.NET per inserire un elemento in una tabella DynamoDB e utilizzare un'istanza della classe.NET per ottenere un elemento dalla tabella.

Definizione di una classe.NET che rappresenta un elemento in una tabella

Nel seguente esempio di definizione di classe, l'`DynamoDBTable` attributo specifica il nome della tabella, mentre gli `DynamoDBRangeKey` attributi `DynamoDBHashKey` e modellano la chiave hash-and-range primaria della tabella. L'`DynamoDBGlobalSecondaryIndexHashKey` attributo è definito in modo da poter creare una query per le risposte di un autore specifico.

```
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; }
}
```

Creazione di un contesto per il modello di persistenza degli oggetti

Per utilizzare il modello di programmazione della persistenza degli oggetti per DynamoDB, è necessario creare un contesto che fornisca una connessione a DynamoDB e consenta di accedere alle tabelle, eseguire varie operazioni ed eseguire query.

Contesto di base

L'esempio seguente mostra come creare il contesto più semplice.

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

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

Contesto con DisableFetchingTableMetadata proprietà

L'esempio seguente mostra come è possibile impostare ulteriormente la `DisableFetchingTableMetadata` proprietà della `DynamoDBContextConfig` classe per impedire chiamate implicite al `DescribeTable` metodo.

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

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Se la `DisableFetchingTableMetadata` proprietà è impostata su `false` (impostazione predefinita), come mostrato nel primo esempio, potete omettere dalla classe gli attributi che descrivono la struttura delle chiavi e degli indici degli elementi della tabella. Questi attributi verranno invece dedotti tramite una chiamata implicita al metodo `DescribeTable`. Se `DisableFetchingTableMetadata` è impostato su `true`, come mostrato nel secondo esempio, sui metodi del modello di persistenza degli oggetti come `SaveAsync` e `QueryAsync` si basa interamente sugli attributi definiti nella classe. In questo caso, non viene effettuata una chiamata al `DescribeTable` metodo.

Note

In determinate condizioni, le chiamate al `DescribeTable` metodo possono talvolta portare a latenze o deadlock aggiuntivi dovuti a comportamenti di avvio a freddo e pool di thread. Per questo motivo, a volte è vantaggioso evitare le chiamate a quel metodo.

Per ulteriori informazioni su questi comportamenti, consulta il post del blog [Improved DynamoDB Initialization Patterns](#) for the AWS SDK for .NET

Utilizzo di un'istanza della classe.NET per inserire un elemento in una tabella

In questo esempio, un elemento viene inserito tramite il `SaveAsync` metodo della `DynamoDBContext` classe, che utilizza un'istanza inizializzata della classe.NET che rappresenta l'elemento.

```
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"
});
```

Utilizzo di un'istanza di una classe.NET per ottenere elementi da una tabella

In questo esempio, viene creata una query per trovare tutti i record di «Author1» utilizzando il `QueryAsync` metodo della `DynamoDBContext` classe. Quindi, gli elementi vengono recuperati tramite il metodo della query. `GetNextSetAsync`

```
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);
}
```

Informazioni aggiuntive sul modello di persistenza degli oggetti

Gli esempi e le spiegazioni mostrati sopra a volte includono una proprietà della `DynamoDBContext` classe chiamata `DisableFetchingTableMetadata`. Questa proprietà, introdotta nella [versione 3.7.203 del NuGet pacchetto AWSSDK .DynamoDBv2](#), consente di evitare determinate condizioni che potrebbero causare latenza o deadlock aggiuntivi dovuti a comportamenti di avvio a freddo e pool di thread. Per ulteriori informazioni, consulta il post sul blog [Improved DynamoDB Initialization Patterns](#) for the AWS SDK for .NET

Di seguito sono riportate alcune informazioni aggiuntive su questa proprietà.

- Questa proprietà può essere impostata globalmente nel tuo `web.config` file `app.config` o se utilizzi .NET Framework.
- Questa proprietà può essere impostata globalmente utilizzando la [AWSConfigsDynamoDB](#) classe, come illustrato nell'esempio seguente.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

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

- In alcuni casi, non è possibile aggiungere attributi DynamoDB a una classe .NET, ad esempio se la classe è definita in una dipendenza. In questi casi, è comunque possibile sfruttare la proprietà.

`DisableFetchingTableMetadata` A tale scopo, utilizzate la [TableBuilder](#) classe in aggiunta alla `DisableFetchingTableMetadata` proprietà. La `TableBuilder` classe è stata introdotta anche nella [versione 3.7.203 del AWSSDK NuGet pacchetto.DynamoDBv2](#).

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

Ulteriori informazioni

Utilizzo di informazioni ed AWS SDK for .NET esempi per programmare DynamoDB**

- [API DynamoDB](#)
- [DynamoDB Series Kickoff](#)
- [DynamoDB Series - Modello di documento](#)
- [DynamoDB Series - Schemi di conversione](#)
- [DynamoDB Series - Modello di persistenza degli oggetti](#)
- [DynamoDB Series - Expressions](#)

- [Utilizzo di espressioni con Amazon DynamoDB e AWS SDK for .NET](#)
- [Supporto JSON in Amazon DynamoDB](#)

Informazioni ed esempi sul modello di basso livello

- [Lavorare con le tabelle utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Lavorare con gli elementi utilizzando l'API di AWS SDK for .NET basso livello](#)
- [Interrogazione di tabelle utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Scansione di tabelle utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Utilizzo degli indici secondari locali utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Utilizzo degli indici secondari globali utilizzando l'API di basso livello AWS SDK for .NET](#)

Informazioni ed esempi sul modello di documento

- [Tipi di dati DynamoDB](#)
- [DynamoDBEntry](#)
- [.NET: modello di documento](#)

Informazioni ed esempi sul modello di persistenza degli oggetti

- [.NET: modello di persistenza degli oggetti](#)

Utilizzo di espressioni con Amazon DynamoDB e AWS SDK for .NET

Note

Le informazioni contenute in questo argomento si riferiscono specificamente ai progetti basati su .NET Framework e alla AWS SDK for .NET versione 3.3 e precedenti.

I seguenti esempi di codice mostrano come utilizzare DynamoDB AWS SDK for .NET per programmare DynamoDB con espressioni. Le espressioni indicano gli attributi che si desidera leggere da un elemento in una tabella DynamoDB. Inoltre, puoi usare le espressioni durante la scrittura di un item per indicare qualsiasi condizione che deve essere soddisfatta (operazione detta anche aggiornamento condizionale) e per indicare le modalità di aggiornamento degli attributi. Alcuni

esempi di aggiornamento sostituiscono l'attributo con un nuovo valore o aggiungono nuovi dati a una lista o una mappa. Per ulteriori informazioni, consulta [Lettura e scrittura di item utilizzando le espressioni](#).

Argomenti

- [Dati di esempio.](#)
- [Ottenere un singolo item utilizzando le espressioni e la chiave primaria dell'item](#)
- [Ottenere più item utilizzando le espressioni e la chiave primaria della tabella](#)
- [Ottenere più item utilizzando le espressioni e altri attributi degli item](#)
- [Stampare un item](#)
- [Creare o sostituire un item utilizzando le espressioni](#)
- [Aggiornare un item utilizzando le espressioni](#)
- [Eliminare un item utilizzando le espressioni](#)
- [Ulteriori informazioni](#)

Dati di esempio.

Gli esempi di codice in questo argomento si basano sui seguenti due elementi di esempio in una tabella DynamoDB denominata ProductCatalog. Questi elementi descrivono le informazioni sulle voci relative ai prodotti in un catalogo di biciclette fittizio. Questi elementi si basano sull'esempio fornito in [Case Study: A ProductCatalog](#) Item. I descrittori del tipo di dati come B00L, L, M, N, NS, S e SS corrispondono a quelli nel [Formato di dati JSON](#).

```
{
  "Id": {
    "N": "205"
  },
  "Title": {
    "S": "20-Bicycle 205"
  },
  "Description": {
    "S": "205 description"
  },
  "BicycleType": {
    "S": "Hybrid"
  },
  "Brand": {
    "S": "Brand-Company C"
  }
}
```

```
},
  "Price": {
    "N": "500"
  },
  "Gender": {
    "S": "B"
  },
  "Color": {
    "SS": [
      "Red",
      "Black"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
            "S": "http://example/products/205_rear.jpg"
          }
        }
      }
    ]
  }
}
```

```
    },
    {
      "M": {
        "SideView": {
          "S": "http://example/products/205_left_side.jpg"
        }
      }
    }
  ]
},
"ProductReviews": {
  "M": {
    "FiveStar": {
      "SS": [
        "Excellent! Can't recommend it highly enough! Buy it!",
        "Do yourself a favor and buy this."
      ]
    },
    "OneStar": {
      "SS": [
        "Terrible product! Do not buy this."
      ]
    }
  }
}
},
{
  "Id": {
    "N": "301"
  },
  "Title": {
    "S": "18-Bicycle 301"
  },
  "Description": {
    "S": "301 description"
  },
  "BicycleType": {
    "S": "Road"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "185"
  }
}
```



```
},
"Gender": {
  "S": "F"
},
"Color": {
  "SS": [
    "Blue",
    "Silver"
  ]
},
"ProductCategory": {
  "S": "Bike"
},
"InStock": {
  "BOOL": true
},
"QuantityOnHand": {
  "N": "3"
},
"RelatedItems": {
  "NS": [
    "801",
    "822",
    "979"
  ]
},
"Pictures": {
  "L": [
    {
      "M": {
        "FrontView": {
          "S": "http://example/products/301_front.jpg"
        }
      }
    },
    {
      "M": {
        "RearView": {
          "S": "http://example/products/301_rear.jpg"
        }
      }
    }
  ],
  {
    "M": {
```

```

        "SideView": {
            "S": "http://example/products/301_left_side.jpg"
        }
    }
}
],
},
"ProductReviews": {
    "M": {
        "FiveStar": {
            "SS": [
                "My daughter really enjoyed this bike!"
            ]
        },
        "ThreeStar": {
            "SS": [
                "This bike was okay, but I would have preferred it in my color.",
                "Fun to ride."
            ]
        }
    }
}
}
}
}

```

Ottenere un singolo item utilizzando le espressioni e la chiave primaria dell'item

L'esempio seguente illustra il metodo `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` e un set di espressioni per ottenere e quindi stampare l'item con un codice Id di 205. Solo i seguenti attributi dell'item vengono restituiti: Id, Title, Description, Color, RelatedItems, Pictures e ProductReviews.

```

// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
    TableName = "ProductCatalog",
    ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#pr", "ProductReviews" },
        { "#ri", "RelatedItems" }
    }
}

```

```
    },
    Key = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "205" } }
    },
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);
```

Nell'esempio precedente, la proprietà `ProjectionExpression` specifica gli attributi da restituire. La proprietà `ExpressionAttributeNames` specifica il segnaposto `#pr` per rappresentare l'attributo `ProductReviews` e il segnaposto `#ri` per rappresentare l'attributo `RelatedItems`. La chiamata a `PrintItem` si riferisce a una funzione personalizzata come descritto in [Stampare un item](#).

Ottenere più item utilizzando le espressioni e la chiave primaria della tabella

L'esempio seguente include il metodo `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` e un set di espressioni per ottenere e quindi stampare l'item con un codice `Id` di `301`, ma solo se il valore del codice `Price` è maggiore di `150`. Solo i seguenti attributi dell'item vengono restituiti: `Id`, `Title` e tutti gli attributi di `ThreeStar` 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("====");
}
```

Nell'esempio precedente, la proprietà `ProjectionExpression` specifica gli attributi da restituire. La proprietà `ExpressionAttributeNames` specifica il segnaposto `#pr` per rappresentare l'attributo `ProductReviews` e il segnaposto `#p` per rappresentare l'attributo `Price`. `#pr.ThreeStar` specifica di restituire solo l'attributo `ThreeStar`. La proprietà `ExpressionAttributeValues` specifica il segnaposto `:val` per rappresentare il valore `150`. La proprietà `FilterExpression` specifica che `#p` (`Price`) deve essere maggiore di `:val` (`150`). La chiamata a `PrintItem` si riferisce a una funzione personalizzata come descritto in [Stampare un item](#).

Ottenere più item utilizzando le espressioni e altri attributi degli item

L'esempio seguente include il metodo `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` e un set di espressioni per ottenere e quindi stampare tutti gli item con un codice `ProductCategory` di `Bike`. Solo i seguenti attributi dell'item vengono restituiti: `Id`, `Title` e tutti gli attributi 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("====");
}
```

Nell'esempio precedente, la proprietà `ProjectionExpression` specifica gli attributi da restituire. La proprietà `ExpressionAttributeNames` specifica il segnaposto `#pr` per rappresentare l'attributo `ProductReviews` e il segnaposto `#pc` per rappresentare l'attributo `ProductCategory`. La proprietà `ExpressionAttributeValues` specifica il segnaposto `:catg` per rappresentare il valore `Bike`. La proprietà `FilterExpression` specifica che `#pc` (`ProductCategory`) deve essere uguale a `:catg` (`Bike`). La chiamata a `PrintItem` si riferisce a una funzione personalizzata come descritto in [Stampare un item](#).

Stampare un item

L'esempio seguente mostra come stampare gli attributi e i valori di un item. Questo esempio viene utilizzato negli esempi precedenti che mostrano come [ottenere un singolo item utilizzando le espressioni e la chiave primaria dell'item](#), [ottenere più item utilizzando le espressioni e la chiave primaria della tabella](#) e [ottenere più item utilizzando le espressioni e altri attributi degli item](#).

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
```

```
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
    foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
    {
        Console.Write(kvp.Key + " = ");
        PrintValue(kvp.Value);
    }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
    // Binary attribute value.
    if (value.B != null)
    {
        Console.Write("Binary data");
    }
    // Binary set attribute value.
    else if (value.BS.Count > 0)
    {
        foreach (var bValue in value.BS)
        {
            Console.Write("\n Binary data");
        }
    }
    // List attribute value.
    else if (value.L.Count > 0)
    {
        foreach (AttributeValue attr in value.L)
        {
            PrintValue(attr);
        }
    }
    // Map attribute value.
    else if (value.M.Count > 0)
    {
        Console.Write("\n");
        PrintItem(value.M);
    }
    // Number attribute value.
    else if (value.N != null)
    {
        Console.Write(value.N);
    }
}
```

```
// Number set attribute value.
else if (value.NS.Count > 0)
{
    Console.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.WriteLine(value.S);
}
// String set attribute value.
else if (value.SS.Count > 0)
{
    Console.WriteLine("{0}", string.Join("\n", value.SS.ToArray()));
}
// Otherwise, boolean value.
else
{
    Console.WriteLine(value.BOOL);
}

Console.WriteLine("\n");
}
```

Nell'esempio precedente, ogni valore di attributo ha diverse data-type-specific proprietà che possono essere valutate per determinare il formato corretto per stampare l'attributo. Queste proprietà includono B, BOOL, BS, L, M, N, NS, NULL, S e SS, che corrispondono a quelle nel [Formato di dati JSON](#). Per proprietà come B, N, NULL e S, se la proprietà corrispondente non è null, l'attributo è del corrispondente tipo di dati non-null. Per proprietà quali BS, L, e M NSSS, se Count è maggiore di zero, l'attributo è del tipo di non-zero-value dati corrispondente. Se tutte le data-type-specific proprietà dell'attributo sono null o sono Count uguali a zero, l'attributo corrisponde al tipo di BOOL dati.

Creare o sostituire un item utilizzando le espressioni

L'esempio seguente include il metodo `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` e un set di espressioni per aggiornare l'item con un codice `Title` di `18-Bicycle 301`. Se l'item non esiste già, ne viene aggiunto uno nuovo.

```
// 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);
```

Nell'esempio precedente, la proprietà `ExpressionAttributeNames` specifica il segnaposto `#title` per rappresentare l'attributo `Title`. La proprietà `ExpressionAttributeValues` specifica il segnaposto `:product` per rappresentare il valore `18-Bicycle 301`. La proprietà `ConditionExpression` specifica che `#title` (`Title`) deve essere uguale a `:product` (`18-Bicycle 301`). La chiamata al codice `CreateItemData` si riferisce alla seguente funzione personalizzata:

```
// 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;
}

```

Nell'esempio precedente, viene restituito all'intermediario un item di esempio con dati campione. Una serie di attributi e valori corrispondenti vengono costruiti utilizzando tipi di dati come BOOL, L, M, N, NS, S e SS, che corrispondono a quelli nel [Formato di dati JSON](#).

Aggiornare un item utilizzando le espressioni

L'esempio seguente include il metodo

`Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` e un set di espressioni per modificare il codice `Title` in `18" Girl's Bike` per l'item con un codice `Id` di `301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
    TableName = "ProductCatalog",
    Key = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "301" } }
    },
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#title", "Title" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
    },
    UpdateExpression = "SET #title = :newproduct"
};
client.UpdateItem(request);
```

Nell'esempio precedente, la proprietà `ExpressionAttributeNames` specifica il segnaposto `#title` per rappresentare l'attributo `Title`. La proprietà `ExpressionAttributeValues` specifica il segnaposto `:newproduct` per rappresentare il valore `18" Girl's Bike`. La proprietà `UpdateExpression` specifica di modificare il codice `#title` (`Title`) in `:newproduct` (`18" Girl's Bike`).

Eliminare un item utilizzando le espressioni

L'esempio seguente include il metodo

`Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` e un set di espressioni per eliminare il codice `Id` di `301` per l'item con un codice `Title` di `18-Bicycle 301`.

```
// using Amazon.DynamoDBv2;
```

```
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
    TableName = "ProductCatalog",
    Key = new Dictionary<string,AttributeValue>
    {
        { "Id", new AttributeValue { N = "301" } }
    },
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#title", "Title" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":product", new AttributeValue { S = "18-Bicycle 301" } }
    },
    ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

Nell'esempio precedente, la proprietà `ExpressionAttributeNames` specifica il segnaposto `#title` per rappresentare l'attributo `Title`. La proprietà `ExpressionAttributeValues` specifica il segnaposto `:product` per rappresentare il valore `18-Bicycle 301`. La proprietà `ConditionExpression` specifica che `#title` (`Title`) deve essere uguale a `:product` (`18-Bicycle 301`).

Ulteriori informazioni

Per maggiori informazioni ed esempi di codice, consulta:

- [DynamoDB Series - Expressions](#)
- [Accesso agli attributi degli item con le espressioni di proiezione](#)
- [Utilizzo dei segnaposti per i nomi e i valori degli attributi](#)
- [Specifiche delle condizioni con le espressioni di condizione](#)
- [Modifica di item e attributi con le espressioni di aggiornamento](#)
- [Lavorare con gli elementi utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Interrogazione di tabelle utilizzando l'API di basso livello AWS SDK for .NET](#)

- [Scansione di tabelle utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Utilizzo degli indici secondari locali utilizzando l'API di basso livello AWS SDK for .NET](#)
- [Utilizzo degli indici secondari globali utilizzando l'API di basso livello AWS SDK for .NET](#)

Supporto JSON in Amazon DynamoDB

Note

Le informazioni contenute in questo argomento si riferiscono specificamente ai progetti basati su .NET Framework e alla AWS SDK for .NET versione 3.3 e precedenti.

AWS SDK for .NET Supporta i dati JSON quando si lavora con Amazon DynamoDB. Ciò consente di ottenere più facilmente dati in formato JSON e di inserire documenti JSON nelle tabelle DynamoDB.

Argomenti

- [Ottieni dati da una tabella DynamoDB in formato JSON](#)
- [Inserimento di dati in formato JSON in una tabella DynamoDB](#)
- [Conversioni dei tipi di dati DynamoDB in JSON](#)
- [Ulteriori informazioni](#)

Ottieni dati da una tabella DynamoDB in formato JSON

L'esempio seguente mostra come ottenere dati da una tabella DynamoDB in formato JSON:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.WriteLine(jsonText);

// Output:
// {"Name":"Shadow","Type":"Horse","Id":3}
```

```
var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
// {
//   "Name" : "Shadow",
//   "Type" : "Horse",
//   "Id"   : 3
// }
```

Nell'esempio precedente, il metodo `ToJson` della classe `Document` converte un elemento della tabella in una stringa in formato JSON. L'elemento viene recuperato con il metodo `GetItem` della classe `Table`. Per determinare l'elemento da ottenere, in questo esempio, il `GetItem` metodo utilizza la chiave hash-and-range primaria dell'elemento di destinazione. Per determinare la tabella da cui ottenere l'elemento, il `LoadTable` metodo della `Table` classe utilizza un'istanza della `AmazonDynamoDBClient` classe e il nome della tabella di destinazione in `DynamoDB`.

Inserimento di dati in formato JSON in una tabella `DynamoDB`

L'esempio seguente mostra come utilizzare il formato JSON per inserire un elemento in una tabella `DynamoDB`:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var jsonText = "{\"Id\":6,\"Type\":\"Bird\",\"Name\":\"Tweety\"}";
var item = Document.FromJson(jsonText);

table.PutItem(item);
```

Nell'esempio precedente, il metodo `FromJson` della classe `Document` converte una stringa in formato JSON in un elemento. L'elemento viene inserito nella tabella con il metodo `PutItem` della classe `Table`, che utilizza l'istanza della classe `Document` che contiene l'elemento. Per determinare la tabella in cui inserire l'elemento, viene chiamato il `LoadTable` metodo della `Table` classe, specificando un'istanza della `AmazonDynamoDBClient` classe e il nome della tabella di destinazione in `DynamoDB`.

Conversioni dei tipi di dati DynamoDB in JSON

Ogni volta che si chiama il `ToJson` metodo della `Document` classe e quindi sui dati JSON risultanti si chiama il `FromJson` metodo per riconvertire i dati JSON in un'istanza di una `Document` classe, alcuni tipi di dati DynamoDB non verranno convertiti come previsto. Nello specifico:

- I set DynamoDB (S) i tipi `NS`, `BS` e) verranno convertiti in array JSON.
- Gli scalari e i set binari di DynamoDB (B) i tipi `BS` and) verranno convertiti in stringhe o elenchi di stringhe JSON con codifica Base64.

In questo scenario, devi chiamare il metodo `DecodeBase64Attributes` della classe `Document` per sostituire i dati JSON con codifica base64 con la corretta rappresentazione binaria. L'esempio seguente sostituisce un attributo dell'elemento scalare binario con codifica base64 in un'istanza di una classe `Document`, denominato `Picture`, con la corretta rappresentazione binaria. In questo esempio viene eseguita la stessa operazione per un attributo dell'elemento di un set binario con codifica base64 nella stessa istanza della classe `Document`, denominato `RelatedPictures`:

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Ulteriori informazioni

Per ulteriori informazioni ed esempi di programmazione JSON con DynamoDB con, vedi: [AWS SDK for .NET](#)

- [Supporto JSON in DynamoDB](#)
- [Aggiornamento di Amazon DynamoDB - JSON, piano gratuito ampliato, dimensionamento flessibile, elementi più grandi](#)

Utilizzo di Amazon EC2

AWS SDK for .NET Supporta [Amazon EC2](#), un servizio Web che fornisce capacità di calcolo ridimensionabile. Utilizzi questa capacità di calcolo per creare e ospitare i tuoi sistemi software.

API

AWS SDK for .NET Fornisce API per i client Amazon EC2. Le API consentono di lavorare con funzionalità EC2 come gruppi di sicurezza e coppie di chiavi. Le API consentono inoltre di controllare le istanze Amazon EC2. Questa sezione contiene un numero limitato di esempi che mostrano i

modelli che puoi seguire quando lavori con queste API. Per visualizzare il set completo di API, consulta l'[AWS SDK for .NET API Reference](#) (e scorri fino a «Amazon.ec2”).

[Le API di Amazon EC2 sono fornite dal pacchetto.EC2. AWSSDK NuGet](#)

Prerequisiti

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Informazioni sugli esempi

Gli esempi in questa sezione mostrano come lavorare con i client Amazon EC2 e gestire le istanze Amazon EC2.

Il [tutorial sulle istanze Spot di EC2](#) mostra come richiedere istanze Spot di Amazon EC2. Le istanze Spot ti consentono di accedere alla capacità EC2 inutilizzata a un prezzo inferiore a quello offerto su richiesta.

Argomenti

- [Lavorare con i gruppi di sicurezza in Amazon EC2](#)
- [Utilizzo delle coppie di chiavi Amazon EC2](#)
- [Visualizzazione delle regioni e delle zone di disponibilità di Amazon EC2](#)
- [Utilizzo delle istanze Amazon EC2](#)
- [Tutorial sull'istanza Spot di Amazon EC2](#)

Lavorare con i gruppi di sicurezza in Amazon EC2

In Amazon EC2, un gruppo di sicurezza funge da firewall virtuale che controlla il traffico di rete per una o più istanze EC2. Per impostazione predefinita, EC2 associa le istanze a un gruppo di sicurezza che non consente il traffico in entrata. È possibile creare un gruppo di sicurezza che consente alle istanze EC2 di accettare un determinato traffico. Ad esempio, se hai bisogno di connetterti a un'istanza Windows EC2, devi configurare il gruppo di sicurezza per consentire il traffico RDP.

Scopri di più sui gruppi di sicurezza nella [Amazon EC2 User Guide](#) e nella [Amazon EC2 User Guide](#).

Quando si utilizza AWS SDK for .NET, è possibile creare un gruppo di sicurezza da utilizzare in EC2 in un VPC o EC2-Classical. [Per ulteriori informazioni su EC2 in un VPC rispetto a EC2-Classical, consulta la Amazon EC2 User Guide o la Amazon EC2 User Guide.](#)

⚠ Warning

Ritireremo EC2-Classic il 15 agosto 2022. Sugeriamo di effettuare la migrazione da EC2-Classic a un VPC. [Per ulteriori informazioni, consulta *Migrare da EC2-Classic a un VPC nella Amazon EC2 User Guide* o nella *Amazon EC2 User Guide*](#). Per ulteriori informazioni, consulta il post di blog [Il networking EC2-Classic va in pensione: ecco come prepararsi](#).

Per informazioni sulle API e sui prerequisiti, consulta la sezione principale (). [Utilizzo di Amazon EC2](#)

Argomenti

- [Enumerazione dei gruppi di sicurezza](#)
- [Creazione dei gruppi di sicurezza](#)
- [Aggiornamento dei gruppi di sicurezza](#)

Enumerazione dei gruppi di sicurezza

Questo esempio mostra come utilizzare per AWS SDK for .NET enumerare i gruppi di sicurezza. Se fornisci un [Amazon Virtual Private Cloud](#) ID, l'applicazione enumera i gruppi di sicurezza per quel particolare VPC. Altrimenti, l'applicazione visualizza semplicemente un elenco di tutti i gruppi di sicurezza disponibili.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Enumera i gruppi di sicurezza](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Enumera i gruppi di sicurezza

Il seguente frammento enumera i tuoi gruppi di sicurezza. Enumera tutti i gruppi o i gruppi per un particolare VPC, se ne viene fornito uno.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.


```
//
// Method to enumerate the security groups
private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
{
    // A request object, in case we need it.
    var request = new DescribeSecurityGroupsRequest();

    // Put together the properties, if needed
    if(!string.IsNullOrEmpty(vpcID))
    {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"\\nGetting security groups for VPC {vpcID}...\\n");
        request.Filters.Add(new Filter
        {
            Name = "vpc-id",
            Values = new List<string>() { vpcID }
        });
    }

    // Get the list of security groups
    DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

    // Display the list of security groups.
    foreach (SecurityGroup item in response.SecurityGroups)
    {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\\tGroupId: " + item.GroupId);
        Console.WriteLine("\\tGroupName: " + item.GroupName);
        Console.WriteLine("\\tVpcId: " + item.VpcId);
        Console.WriteLine();
    }
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

[Filtro](#) di classe

Classe [SecurityGroup](#)

Il codice

```
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();
}
}
```

Ulteriori considerazioni

- Nota per il caso VPC che il filtro è costruito con la Name parte della coppia nome-valore impostata su «vpc-id». Questo nome deriva dalla descrizione della proprietà della classe. [Filters DescribeSecurityGroupsRequest](#)
- Per ottenere l'elenco completo dei tuoi gruppi di sicurezza, puoi anche utilizzarlo [DescribeSecurityGroupsAsync senza parametri](#).
- Puoi verificare i risultati controllando l'elenco dei gruppi di sicurezza nella console [Amazon EC2](#).

Creazione dei gruppi di sicurezza

Questo esempio mostra come utilizzare il per AWS SDK for .NET creare un gruppo di sicurezza. Puoi fornire l'ID di un VPC esistente per creare un gruppo di sicurezza per EC2 in un VPC. Se non fornisci tale ID, il nuovo gruppo di sicurezza sarà destinato a EC2-Classical, se il tuo account lo supporta. AWS

Se non fornisci un ID VPC e il tuo AWS account non supporta EC2-Classical, il nuovo gruppo di sicurezza apparterrà al VPC predefinito del tuo account. Per ulteriori informazioni, consulta i riferimenti per EC2 in un VPC rispetto a EC2-Classical nella sezione principale (). [Lavorare con i gruppi di sicurezza in Amazon EC2](#)

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Trova i gruppi di sicurezza esistenti](#)
- [Creazione di un gruppo di sicurezza](#)

- [Codice completo](#)

Trova i gruppi di sicurezza esistenti

Il seguente frammento cerca i gruppi di sicurezza esistenti con il nome specificato nel VPC specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//  
// Method to determine if a security group with the specified name  
// already exists in the VPC  
private static async Task<List<SecurityGroup>> FindSecurityGroups(  
    IAmazonEC2 ec2Client, string groupName, string vpcID)  
{  
    var request = new DescribeSecurityGroupsRequest();  
    request.Filters.Add(new Filter{  
        Name = "group-name",  
        Values = new List<string>() { groupName }  
    });  
    if(!string.IsNullOrEmpty(vpcID))  
        request.Filters.Add(new Filter{  
            Name = "vpc-id",  
            Values = new List<string>() { vpcID }  
        });  
  
    var response = await ec2Client.DescribeSecurityGroupsAsync(request);  
    return response.SecurityGroups;  
}
```

Creazione di un gruppo di sicurezza

Il seguente frammento di codice crea un nuovo gruppo di sicurezza se un gruppo con quel nome non esiste nel VPC specificato. Se non viene fornito alcun VPC ed esistono uno o più gruppi con quel nome, lo snippet restituisce semplicemente l'elenco dei gruppi.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to create a new security group (either EC2-Classic or EC2-VPC)  
// If vpcID is empty, the security group will be for EC2-Classic
```

```
private static async Task<List<SecurityGroup>> CreateSecurityGroup(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    // See if one or more security groups with that name
    // already exist in the given VPC. If so, return the list of them.
    var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
    if (securityGroups.Count > 0)
    {
        Console.WriteLine(
            $"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 = "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;
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [CreateSecurityGroupRequest](#)

Classe [CreateSecurityGroupResponse](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

[Filtro](#) di classe

Classe [SecurityGroup](#)

Il codice

```
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(
            $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
    }

    // If the security group doesn't already exists, create it.
    var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
    };
    if(string.IsNullOrEmpty(vpcID))
    {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
    }
    else
    {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
    }
    CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

    // Return the new security group
    DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
            GroupIds = new List<string>() { createResponse.GroupId }
        });
    return describeResponse.SecurityGroups;
}

//
// Method to determine if a security group with the specified name
// already exists in the VPC
private static async Task<List<SecurityGroup>> FindSecurityGroups(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    var request = new DescribeSecurityGroupsRequest();
    request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
    });
    if(!string.IsNullOrEmpty(vpcID))
```

```

        request.Filters.Add(new Filter{
            Name = "vpc-id",
            Values = new List<string>() { vpcID }
        });

        var response = await ec2Client.DescribeSecurityGroupsAsync(request);
        return response.SecurityGroups;
    }

    //
    // Command-line help
    private static void PrintHelp()
    {
        Console.WriteLine(
            "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
            "\n  -g, --group-name: The name you would like the new security group to have."
+
            "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
            "\n      If vpc-id isn't present, the security group will be" +
            "\n      for EC2-Classic (if your AWS account supports this)" +
            "\n      or will use the default VCP for EC2-VPC.");
    }
}

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

Aggiornamento dei gruppi di sicurezza

Questo esempio mostra come utilizzare per aggiungere una regola a un gruppo di sicurezza. AWS SDK for .NET In particolare, l'esempio aggiunge una regola per consentire il traffico in entrata su una determinata porta TCP, che può essere utilizzata, ad esempio, per le connessioni remote a un'istanza EC2. L'applicazione richiede l'ID di un gruppo di sicurezza esistente, un indirizzo IP (o intervallo di indirizzi) in formato CIDR e, facoltativamente, un numero di porta TCP. Quindi aggiunge una regola in entrata al gruppo di sicurezza specificato.

Note

Per utilizzare questo esempio, è necessario un indirizzo IP (o intervallo di indirizzi) in formato CIDR. Per informazioni sui metodi per ottenere l'indirizzo IP del computer locale, vedere [Considerazioni aggiuntive a questa fine di questo argomento](#).

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Aggiungi una regola in entrata](#)
- [Codice completo](#)

- [Ulteriori considerazioni](#)

Aggiungi una regola in entrata

Il seguente frammento aggiunge una regola in entrata a un gruppo di sicurezza per un particolare indirizzo IP (o intervallo) e una porta TCP.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
    IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
    // Create an object to hold the request information for the rule.
    // It uses an IpPermission object to hold the IP information for the rule.
    var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
    ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
    });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [AuthorizeSecurityGroupIngressRequest](#)

Classe [AuthorizeSecurityGroupIngressResponse](#)

Classe [IpPermission](#)

Classe [IpRange](#)

Il codice

```
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 e2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"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);
    }
}
}
```

Ulteriori considerazioni

- Se non si fornisce un numero di porta, per impostazione predefinita l'applicazione utilizza la porta 3389. Questa è la porta per Windows RDP, che consente di connettersi a un'istanza EC2 che esegue Windows. Se stai avviando un'istanza EC2 che esegue Linux, puoi invece utilizzare la porta TCP 22 (SSH).

- Nota che l'esempio è impostato su «tcp»IpProtocol. I valori di IpProtocol possono essere trovati nella descrizione della IpProtocol proprietà della [IpPermission](#) classe.

- Quando usi questo esempio, potresti volere l'indirizzo IP del tuo computer locale. Di seguito sono riportati alcuni dei modi in cui è possibile ottenere l'indirizzo.

- Se il tuo computer locale (da cui ti conatterai alla tua istanza EC2) ha un indirizzo IP pubblico statico, puoi utilizzare un servizio per ottenere quell'indirizzo. Uno di questi servizi è <http://checkip.amazonaws.com/>. Per ulteriori informazioni sull'autorizzazione del traffico in entrata, consulta la Amazon [EC2 User Guide](#) o la [Amazon EC2 User Guide](#).

- Un altro modo per ottenere l'indirizzo IP del computer locale consiste nell'utilizzare la console [Amazon EC2](#).

Seleziona uno dei tuoi gruppi di sicurezza, seleziona la scheda Regole in entrata e scegli Modifica regole in entrata. In una regola in entrata, apri il menu a discesa nella colonna Origine e scegli Il mio IP per visualizzare l'indirizzo IP del tuo computer locale in formato CIDR. Assicurati di annullare l'operazione.

- Puoi verificare i risultati di questo esempio esaminando l'elenco dei gruppi di sicurezza nella console [Amazon EC2](#).

Utilizzo delle coppie di chiavi Amazon EC2

Amazon EC2 utilizza la crittografia a chiave pubblica per crittografare e decrittografare le informazioni di accesso. La crittografia a chiave pubblica utilizza una chiave pubblica per crittografare i dati, quindi

il destinatario utilizza la chiave privata per decrittografare i dati. La chiave pubblica e quella privata sono note come coppia di chiavi. Se desideri accedere a un'istanza EC2, devi specificare una coppia di chiavi all'avvio e quindi fornire la chiave privata della coppia quando ti connetti ad essa.

Quando avvii un'istanza EC2, puoi creare una key pair per essa o usarne una che hai già usato per lanciare altre istanze. Per saperne di più sulle coppie di chiavi di Amazon EC2, consulta la [Amazon EC2 User Guide](#) o la [Amazon EC2 User Guide](#).

Per informazioni sulle API e sui prerequisiti, consulta la sezione principale (). [Utilizzo di Amazon EC2](#)

Argomenti

- [Creazione e visualizzazione di coppie di chiavi](#)
- [Eliminazione di coppie di chiavi](#)

Creazione e visualizzazione di coppie di chiavi

Questo esempio mostra come utilizzare AWS SDK for .NET per creare una key pair. L'applicazione prende il nome per la nuova coppia di chiavi e il nome di un file PEM (con estensione «.pem»). Crea la coppia di chiavi, scrive la chiave privata nel file PEM e quindi visualizza tutte le coppie di chiavi disponibili. Se non si forniscono argomenti della riga di comando, l'applicazione visualizza semplicemente tutte le coppie di chiavi disponibili.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Creazione della coppia di chiavi](#)
- [Visualizza le coppie di chiavi disponibili](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Creazione della coppia di chiavi

Il seguente frammento crea una coppia di chiavi e quindi memorizza la chiave privata nel file PEM specificato.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to create a key pair and save the key material in a PEM file  
private static async Task CreateKeyPair(  
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)  
{  
    // Create the key pair  
    CreateKeyPairResponse response =  
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{  
            KeyName = keyPairName  
        });  
    Console.WriteLine($"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);  
    }  
}
```

Visualizza le coppie di chiavi disponibili

Il seguente frammento mostra un elenco delle coppie di chiavi disponibili.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [CreateKeyPairRequest](#)

Classe [CreateKeyPairResponse](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyPairInfo](#)

Il codice

```
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);
}
}
}
```

Ulteriori considerazioni

- Dopo aver eseguito l'esempio, puoi vedere la nuova coppia di key nella console [Amazon EC2](#).
- Quando si crea una coppia di chiavi, è necessario salvare la chiave privata restituita perché non è possibile recuperarla in un secondo momento.

Eliminazione di coppie di chiavi

Questo esempio mostra come utilizzare AWS SDK for .NET per eliminare una key pair.

L'applicazione prende il nome di una key pair. Elimina la coppia di chiavi e quindi visualizza tutte le coppie di chiavi disponibili. Se non si forniscono argomenti nella riga di comando, l'applicazione visualizza semplicemente tutte le coppie di chiavi disponibili.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Eliminare la key pair](#)

- [Visualizza le coppie di chiavi disponibili](#)
- [Codice completo](#)

Eliminare la key pair

Il seguente frammento elimina una key pair.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

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

Visualizza le coppie di chiavi disponibili

Il seguente frammento mostra un elenco delle coppie di chiavi disponibili.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [DeleteKeyPairRequest](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyPairInfo](#)

Il codice

```
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($"{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}");
}
}
```

Visualizzazione delle regioni e delle zone di disponibilità di Amazon EC2

Amazon EC2 è ospitato in diverse località in tutto il mondo. Tali località sono composte da regioni e zone di disponibilità. Ogni regione è un'area geografica separata con più località isolate note come zone di disponibilità.

Scopri di più su regioni e zone di disponibilità nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#).

Questo esempio mostra come utilizzare il per AWS SDK for .NET ottenere dettagli sulle regioni e le zone di disponibilità relative a un client EC2. L'applicazione visualizza gli elenchi delle regioni e delle zone di disponibilità disponibili per un client EC2.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [DescribeAvailabilityZonesResponse](#)

Classe [DescribeRegionsResponse](#)

Classe [AvailabilityZone](#)

[Regione](#) della classe

```
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);
}
}
}
```

Utilizzo delle istanze Amazon EC2

Puoi utilizzarlo AWS SDK for .NET per controllare le istanze Amazon EC2 con operazioni come creazione, avvio e terminazione. Gli argomenti di questa sezione forniscono alcuni esempi di come eseguire questa operazione. [Scopri di più sulle istanze EC2 nella Amazon EC2 User Guide o nella Amazon EC2 User Guide](#)

Per informazioni sulle API e sui prerequisiti, consulta la sezione principale (). [Utilizzo di Amazon EC2](#)

Argomenti

- [Avvio di un'istanza Amazon EC2](#)

- [Terminazione di un'istanza Amazon EC2](#)

Avvio di un'istanza Amazon EC2

Questo esempio mostra come utilizzare AWS SDK for .NET per avviare una o più istanze Amazon EC2 configurate in modo identico dalla stessa Amazon Machine Image (AMI). Utilizzando [diversi input forniti](#), l'applicazione avvia un'istanza EC2 e quindi monitora l'istanza fino a quando non esce dallo stato «In sospeso».

Quando l'istanza EC2 è in esecuzione, puoi connetterti ad essa in remoto, come descritto in. [\(opzionale\) Connect all'istanza](#)

Puoi avviare un'istanza EC2 in un VPC o in EC2-Classic (se AWS il tuo account lo supporta). [Per ulteriori informazioni su EC2 in un VPC rispetto a EC2-Classic, consulta la Amazon EC2 User Guide o la Amazon EC2 User Guide.](#)

Warning

Ritiremo EC2-Classic il 15 agosto 2022. Sugeriamo di effettuare la migrazione da EC2-Classic a un VPC. [Per ulteriori informazioni, consulta Migrare da EC2-Classic a un VPC nella Amazon EC2 User Guide o nella Amazon EC2 User Guide.](#) Per ulteriori informazioni, consulta il post di blog [Il networking EC2-Classic va in pensione: ecco come prepararsi.](#)

Le sezioni seguenti forniscono frammenti e altre informazioni per questo esempio. Il [codice completo dell'esempio](#) viene mostrato dopo gli snippet e può essere creato ed eseguito così com'è.

Argomenti

- [Raccogli ciò di cui hai bisogno](#)
- [Avvio di un'istanza](#)
- [Monitora l'istanza](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)
- [\(opzionale\) Connect all'istanza](#)
- [Eliminazione](#)

Raccogli ciò di cui hai bisogno

Per avviare un'istanza EC2, avrai bisogno di diversi elementi.

- Un [VPC](#) in cui verrà avviata l'istanza. Se si tratta di un'istanza Windows e ti conetterai ad essa tramite RDP, molto probabilmente il VPC dovrà avere un gateway Internet collegato, oltre a una voce per il gateway Internet nella tabella delle rotte. Per ulteriori informazioni, consulta la sezione [Gateway Internet](#) nella Guida per l'utente di Amazon VPC.
- L'ID di una sottorete esistente nel VPC in cui verrà avviata l'istanza. Un modo semplice per trovarlo o crearlo è accedere alla [console Amazon VPC](#), ma puoi anche ottenerlo a livello di codice utilizzando i metodi and. [CreateSubnetAsyncDescribeSubnetsAsync](#)

Note

Se il tuo AWS account supporta EC2-Classical e questo è il tipo di istanza che desideri avviare, questo parametro non è richiesto. Tuttavia, se il tuo account non supporta EC2-Classical e non fornisci questo parametro, la nuova istanza viene avviata nel VPC predefinito del tuo account.

- L'ID di un gruppo di sicurezza esistente che appartiene al VPC in cui verrà avviata l'istanza. Per ulteriori informazioni, consulta [Lavorare con i gruppi di sicurezza in Amazon EC2](#).
- Se desideri connetterti alla nuova istanza, il gruppo di sicurezza menzionato in precedenza deve disporre di una regola in entrata appropriata che consenta il traffico SSH sulla porta 22 (istanza Linux) o il traffico RDP sulla porta 3389 (istanza Windows). Per informazioni su come eseguire questa operazione [Aggiornamento dei gruppi di sicurezza](#), consulta anche la parte finale [Ulteriori considerazioni](#) dell'argomento.
- L'Amazon Machine Image (AMI) che verrà utilizzata per creare l'istanza. Consulta le informazioni sulle AMI nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#). [Ad esempio, leggi le AMI condivise nella Amazon EC2 User Guide o nella Amazon EC2 User Guide.](#)
- Il nome di una coppia di key pair EC2 esistente, utilizzata per connettersi alla nuova istanza. Per ulteriori informazioni, consulta [Utilizzo delle coppie di chiavi Amazon EC2](#).

- Il nome del file PEM che contiene la chiave privata della coppia di chiavi EC2 menzionata in precedenza. Il file PEM viene utilizzato quando ci si [connette in remoto](#) all'istanza.

Avvio di un'istanza

Il seguente frammento avvia un'istanza EC2.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

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

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

    return instanceIds;
}
```

Monitora l'istanza

Il seguente frammento monitora l'istanza fino a quando non esce dallo stato «In sospeso».

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

Consultate la [InstanceState](#) classe per i valori validi della Instance.State.Code proprietà.

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

```
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

Classe [InstanceType](#)

- [Spazio dei nomi Amazon.ec2.model](#)

Classe [DescribeInstancesRequest](#)

Classe [DescribeInstancesResponse](#)

[Istanza](#) di classe

Classe [InstanceNetworkInterfaceSpecification](#)

Classe [RunInstancesRequest](#)

Classe [RunInstancesResponse](#)

Il codice

```
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.Write(".");
    }
}
```

```
numberRunning = 0;
responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
foreach(Instance i in responseDescribe.Reservations[0].Instances)
{
    // Check the lower byte of State.Code property
    // Code == 0 is the pending state
    if((i.State.Code & 255) > 0) numberRunning++;
}
if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
    break;

// Wait a bit and try again (unless the user wants to stop waiting)
Thread.Sleep(wait);
if(Console.KeyAvailable)
    break;
}

Console.WriteLine("\nNo more instances are pending.");
foreach(Instance i in responseDescribe.Reservations[0].Instances)
{
    Console.WriteLine($"For {i.InstanceId}:");
    Console.WriteLine($"  VPC ID: {i.VpcId}");
    Console.WriteLine($"  Instance state: {i.State.Name}");
    Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
    Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
    Console.WriteLine($"  Key pair name: {i.KeyName}");
}
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
}
}
```

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

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

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



```
    }
  }

  return parsedArgs;
}

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

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

Ulteriori considerazioni

- Quando controlli lo stato di un'istanza EC2, puoi aggiungere un filtro alla `Filter` proprietà dell'[DescribeInstancesRequest](#) oggetto. Utilizzando questa tecnica, puoi limitare la richiesta a determinate istanze; ad esempio, istanze con un particolare tag specificato dall'utente.

- Per brevità, ad alcune proprietà sono stati assegnati valori tipici. Alcune o tutte queste proprietà possono invece essere determinate a livello di codice o tramite l'input dell'utente.
- I valori che è possibile utilizzare per l'[RunInstancesRequest](#) oggetto MinCount e MaxCount le proprietà sono determinati dalla zona di disponibilità di destinazione e dal numero massimo di istanze consentite per il tipo di istanza. Per ulteriori informazioni, consulta [Quante istanze posso eseguire in Amazon EC2 nelle Domande frequenti generali di Amazon EC2](#).
- [Se desideri utilizzare un tipo di istanza diverso da questo esempio, puoi scegliere tra diversi tipi di istanza, consultabili nella Amazon EC2 User Guide o nella Amazon EC2 User Guide.](#)
- Puoi anche associare un [ruolo IAM](#) a un'istanza al momento del lancio. A tale scopo, crea un [IamInstanceProfileSpecification](#) oggetto la cui Name proprietà è impostata sul nome di un ruolo IAM. Quindi aggiungi quell'oggetto alla IamInstanceProfile proprietà dell'[RunInstancesRequest](#) oggetto.

Note

Per avviare un'istanza EC2 a cui è associato un ruolo IAM, la configurazione di un utente IAM deve includere determinate autorizzazioni. [Per ulteriori informazioni sulle autorizzazioni richieste, consulta la Amazon EC2 User Guide o la Amazon EC2 User Guide.](#)

(opzionale) Connect all'istanza

Dopo l'esecuzione di un'istanza, puoi connetterti ad essa in remoto utilizzando il client remoto appropriato. Sia per le istanze Linux che per quelle Windows, è necessario l'indirizzo IP pubblico o il nome DNS pubblico dell'istanza. È inoltre necessario quanto segue.

Per istanze Linux

Puoi usare un client SSH per connetterti alla tua istanza Linux. Assicurati che il gruppo di sicurezza utilizzato all'avvio dell'istanza consenta il traffico SSH sulla porta 22, come descritto in [Aggiornamento dei gruppi di sicurezza](#)

È inoltre necessaria la parte privata della coppia di chiavi utilizzata per avviare l'istanza, ovvero il file PEM.

Per ulteriori informazioni, consulta [Connect to your Linux instance](#) nella Amazon EC2 User Guide.


Per le istanze Windows

Puoi utilizzare un client RDP per connetterti alla tua istanza. Assicurati che il gruppo di sicurezza utilizzato all'avvio dell'istanza consenta il traffico RDP sulla porta 3389, come descritto in.

[Aggiornamento dei gruppi di sicurezza](#)

È inoltre necessaria la password dell'amministratore. È possibile ottenere ciò utilizzando il seguente codice di esempio, che richiede l'ID dell'istanza e la parte privata della coppia di chiavi utilizzata per avviare l'istanza, ovvero il file PEM.

Per ulteriori informazioni, consulta [Connessione all'istanza Windows](#) nella Guida per l'utente di Amazon EC2.

 Warning

Questo codice di esempio restituisce la password di amministratore in testo semplice per l'istanza.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [GetPasswordDataRequest](#)

Classe [GetPasswordDataResponse](#)

Il codice

```
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($"
The password is not available for instance
{instanceID}.");
        Console.WriteLine($"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);
}
}
}
```

Eliminazione

Quando non hai più bisogno della tua istanza EC2, assicurati di terminarla, come descritto in.

[Terminazione di un'istanza Amazon EC2](#)

Terminazione di un'istanza Amazon EC2

Quando non hai più bisogno di una o più istanze Amazon EC2, puoi terminarle.

Questo esempio mostra come utilizzare per AWS SDK for .NET terminare le istanze EC2. Richiede un ID di istanza come input.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

- Spazio dei [nomi Amazon.ec2.model](#)

Classe [TerminateInstancesRequest](#)

Classe [TerminateInstancesResponse](#)

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

namespace EC2TerminateInstance
{
    class Program
    {
        static async Task Main(string[] args)
        {
            if((args.Length == 1) && (args[0].StartsWith("i-")))
            {
                // Terminate the instance
                var ec2Client = new AmazonEC2Client();
                await TerminateInstance(ec2Client, args[0]);
            }
            else
            {
                Console.WriteLine("\nCommand-line argument missing or incorrect.");
                Console.WriteLine("\nUsage: EC2TerminateInstance instance-ID");
                Console.WriteLine(" instance-ID - The EC2 instance you want to terminate.");
                return;
            }
        }
    }

    //
```



```
// Method to terminate an EC2 instance
private static async Task TerminateInstance(IAmazonEC2 ec2Client, string
instanceID)
{
    var request = new TerminateInstancesRequest{
        InstanceIds = new List<string>() { instanceID }};
    TerminateInstancesResponse response =
        await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>() { instanceID }
        });
    foreach (InstanceStateChange item in response.TerminatingInstances)
    {
        Console.WriteLine("Terminated instance: " + item.InstanceId);
        Console.WriteLine("Instance state: " + item.CurrentState.Name);
    }
}
}
```

Dopo aver eseguito l'esempio, è consigliabile accedere alla [console Amazon EC2](#) per verificare che l'[istanza EC2](#) sia stata terminata.

Tutorial sull'istanza Spot di Amazon EC2

Questo tutorial mostra come utilizzarlo per AWS SDK for .NET gestire le istanze Spot di Amazon EC2.

Panoramica

Le istanze Spot consentono di richiedere capacità Amazon EC2 inutilizzata a un prezzo inferiore al prezzo On-Demand. Ciò può ridurre in modo significativo i costi EC2 per le applicazioni che possono essere interrotte.

Di seguito è riportato un riepilogo di alto livello di come le istanze Spot vengono richieste e utilizzate.

1. Crea una richiesta di istanza Spot, specificando il prezzo massimo che sei disposto a pagare.
2. Una volta soddisfatta la richiesta, esegui l'istanza come faresti con qualsiasi altra istanza Amazon EC2.
3. Esegui l'istanza per tutto il tempo che desideri e poi chiudila, a meno che il prezzo Spot non cambi in modo tale che l'istanza venga interrotta per te.

4. Pulisci la richiesta dell'istanza Spot quando non è più necessaria, in modo che le istanze Spot non vengano più create.

Questa è stata una panoramica di altissimo livello sulle istanze Spot. Per comprendere meglio le istanze Spot, consulta la Guida per l'utente di [Amazon EC2 o la Guida per l'utente di Amazon EC2](#).

Informazioni sul tutorial

Mentre segui questo tutorial, usi il AWS SDK for .NET per fare quanto segue:

- Creare una richiesta di istanza spot
- Determina quando la richiesta di istanza Spot è stata soddisfatta
- Annulla la richiesta di istanza Spot
- Terminare le istanze associate

Le sezioni seguenti forniscono frammenti e altre informazioni per questo esempio. Il [codice completo dell'esempio](#) viene mostrato dopo gli snippet e può essere creato ed eseguito così com'è.

Argomenti

- [Prerequisiti](#)
- [Raccogli ciò di cui hai bisogno](#)
- [Creazione di una richiesta di istanza Spot](#)
- [Determina lo stato della tua richiesta di istanza Spot](#)
- [Pulisci le tue richieste di istanze Spot](#)
- [Pulisci le tue istanze Spot](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Prerequisiti

Per informazioni sulle API e sui prerequisiti, consultate la sezione principale (). [Utilizzo di Amazon EC2](#)

Raccogli ciò di cui hai bisogno

Per creare una richiesta di istanza Spot, avrai bisogno di diversi elementi.

- Il numero di istanze e il relativo tipo di istanza. Esistono diversi tipi di istanze tra cui scegliere, consultabili nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#). Il numero predefinito per questo tutorial è 1.
- L'Amazon Machine Image (AMI) che verrà utilizzata per creare l'istanza. Consulta le informazioni sulle AMI nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#). [Ad esempio, leggi le AMI condivise nella Amazon EC2 User Guide o nella Amazon EC2 User Guide.](#)
- Il prezzo massimo che sei disposto a pagare per ora di istanza. Puoi vedere i prezzi per tutti i tipi di istanze (sia per le istanze On-Demand che per le istanze Spot) nella pagina dei prezzi di [Amazon EC2](#). Il prezzo predefinito per questo tutorial viene spiegato più avanti.
- Se desideri connetterti in remoto a un'istanza, un gruppo di sicurezza con la configurazione e le risorse appropriate. Questo è descritto in [Lavorare con i gruppi di sicurezza in Amazon EC2](#) e le informazioni su come [raccogliere ciò di cui hai bisogno](#) e [connetterti a un'istanza in Avvio di un'istanza Amazon EC2](#). Per semplicità, questo tutorial utilizza il gruppo di sicurezza denominato default di cui dispongono tutti gli AWS account più recenti.

Esistono molti approcci per proporre delle richieste per le istanze Spot. Le seguenti sono strategie comuni:

- Effettua richieste che sicuramente costeranno meno dei prezzi su richiesta.
- Effettua richieste in base al valore del calcolo risultante.
- Effettua richieste in modo da acquisire capacità di calcolo il più rapidamente possibile.

Le seguenti spiegazioni si riferiscono alla cronologia dei prezzi Spot nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#).

Riduci i costi al di sotto di quelli disponibili

Hai un processo di elaborazione in batch la cui esecuzione richiede diverse ore o giorni. Tuttavia, hai una certa flessibilità sui tempi di inizio e di fine. Vuoi vedere se riesci a completarlo a un costo inferiore a quello delle istanze on demand.

Puoi esaminare la cronologia dei prezzi Spot per i tipi di istanza utilizzando la console Amazon EC2 o l'API Amazon EC2. Dopo aver analizzato la cronologia dei prezzi per il tipo di istanza desiderato in una determinata zona di disponibilità, hai due alternative per la tua richiesta:

- Specificate una richiesta all'estremità superiore dell'intervallo di prezzi Spot, che sono ancora inferiori al prezzo on demand, prevedendo che la richiesta effettuata una sola volta sull'istanza Spot venga probabilmente soddisfatta ed eseguita per un tempo di calcolo consecutivo sufficiente a completare il lavoro.
- Specifica una richiesta nella fascia bassa della gamma di prezzi, pianificando di combinare più istanze avviate in momenti diversi in un'unica richiesta persistente. Le istanze, in forma aggregata, verrebbero eseguite abbastanza a lungo da completare il processo a un costo totale persino inferiore.

Non paghi più del valore del risultato

Hai un processo di elaborazione dei dati da completare. Conosci abbastanza bene il valore dei risultati del lavoro da sapere quanto valgono in termini di costi di elaborazione.

Dopo aver analizzato la cronologia dei prezzi Spot per il tipo di istanza, scegli un prezzo al quale il costo del tempo di elaborazione non sia superiore al valore dei risultati del lavoro. Crea una richiesta persistente e impostane l'esecuzione intermittente quando il prezzo Spot raggiunge o scende sotto la tua richiesta.

Acquisisci rapidamente capacità di elaborazione

Hai un bisogno imprevisto, a breve termine, di capacità aggiuntiva che non è disponibile tramite le istanze on demand. Dopo aver analizzato la cronologia dei prezzi Spot per il tipo di istanza, scegli un prezzo superiore al prezzo storico più alto per aumentare notevolmente la probabilità che la tua richiesta venga soddisfatta rapidamente e continui a calcolare fino al completamento.

Dopo aver raccolto ciò di cui hai bisogno e scelto una strategia, sei pronto per richiedere un'istanza Spot. Per questa esercitazione il prezzo massimo di spot-istanza predefinito è impostato per essere lo stesso del prezzo su richiesta (che è \$0,003 per questo tutorial). Impostare il prezzo in questo modo massimizza le possibilità che la richiesta venga soddisfatta.

Creazione di una richiesta di istanza Spot

Il seguente frammento mostra come creare una richiesta di istanza Spot con gli elementi raccolti in precedenza.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to create a Spot Instance request  
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(  
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,  
    InstanceType instanceType, string spotPrice, int instanceCount)  
{  
    var launchSpecification = new LaunchSpecification{  
        ImageId = amiId,  
        InstanceType = instanceType  
    };  
    launchSpecification.SecurityGroups.Add(securityGroupName);  
    var request = new RequestSpotInstancesRequest{  
        SpotPrice = spotPrice,  
        InstanceCount = instanceCount,  
        LaunchSpecification = launchSpecification  
    };  
  
    RequestSpotInstancesResponse result =  
        await ec2Client.RequestSpotInstancesAsync(request);  
    return result.SpotInstanceRequests[0];  
}
```

Il valore importante restituito da questo metodo è l'ID della richiesta dell'istanza Spot, contenuto nel `SpotInstanceRequestId` membro dell'oggetto restituito [SpotInstanceRequest](#).

Note

Ti verranno addebitati i costi per ogni istanza Spot lanciata. Per evitare costi inutili, assicurati di [annullare qualsiasi richiesta](#) e [terminare qualsiasi](#) istanza.

Determina lo stato della tua richiesta di istanza Spot

Il seguente frammento mostra come ottenere informazioni sulla richiesta di istanza Spot. Puoi utilizzare queste informazioni per prendere determinate decisioni nel codice, ad esempio se continuare ad aspettare che venga soddisfatta una richiesta di istanza Spot.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//
```

```
// Method to get information about a Spot Instance request, including the status,  
// instance ID, etc.  
// It gets the information for a specific request (as opposed to all requests).  
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(  
    IAmazonEC2 ec2Client, string requestId)  
{  
    var describeRequest = new DescribeSpotInstanceRequestsRequest();  
    describeRequest.SpotInstanceRequestIds.Add(requestId);  
  
    DescribeSpotInstanceRequestsResponse describeResponse =  
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);  
    return describeResponse.SpotInstanceRequests[0];  
}
```

Il metodo restituisce informazioni sulla richiesta dell'istanza Spot, come l'ID dell'istanza, lo stato e il codice di stato. Puoi visualizzare i codici di stato per le richieste di istanze Spot nella Guida per l'utente di [Amazon EC2](#) o nella Guida per l'utente di [Amazon EC2](#).

Pulisci le tue richieste di istanze Spot

Quando non è più necessario richiedere istanze Spot, è importante annullare le richieste in sospeso per evitare che vengano soddisfatte nuovamente. Il seguente frammento mostra come annullare una richiesta di istanza Spot.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to cancel a Spot Instance request  
private static async Task CancelSpotInstanceRequest(  
    IAmazonEC2 ec2Client, string requestId)  
{  
    var cancelRequest = new CancelSpotInstanceRequestsRequest();  
    cancelRequest.SpotInstanceRequestIds.Add(requestId);  
  
    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);  
}
```

Pulisci le tue istanze Spot

Per evitare costi inutili, è importante chiudere tutte le istanze avviate dalle richieste di istanze Spot; la semplice cancellazione delle richieste di istanze Spot non comporterà la chiusura delle istanze, il che

significa che continueranno a ricevere i relativi costi. Il seguente frammento mostra come terminare un'istanza dopo aver ottenuto l'identificatore di istanza per un'istanza Spot attiva.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//
// Method to terminate a Spot Instance
private static async Task TerminateSpotInstance(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    // Retrieve the Spot Instance request to check for running instances.
    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

    // If there are any running instances, terminate them
    if( (describeResponse.SpotInstanceRequests[0].Status.Code
        == "request-canceled-and-instance-running")
        || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
    {
        TerminateInstancesResponse response =
            await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
                InstanceIds = new List<string>(){
                    describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
            Console.WriteLine($"\\n Terminated instance: {item.InstanceId}");
            Console.WriteLine($" Instance state: {item.CurrentState.Name}\\n");
        }
    }
}
```

Codice completo

Il seguente esempio di codice richiama i metodi descritti in precedenza per creare e annullare una richiesta di istanza Spot e terminare un'istanza Spot.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.EC2](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.ec2](#)

Classe [AmazonEC2Client](#)

Classe [InstanceType](#)

- [Spazio dei nomi Amazon.ec2.model](#)

Classe [CancelSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsResponse](#)

Classe [InstanceStateChange](#)

Classe [LaunchSpecification](#)

Classe [RequestSpotInstancesRequest](#)

Classe [RequestSpotInstancesResponse](#)

Classe [SpotInstanceRequest](#)

Classe [TerminateInstancesRequest](#)

Classe [TerminateInstancesResponse](#)

Il codice

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



```
namespace EC2SpotInstanceRequests
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Some default values.
            // These could be made into command-line arguments instead.
            var instanceType = InstanceType.T1Micro;
            string securityGroupName = "default";
            string spotPrice = "0.003";
            int instanceCount = 1;

            // Parse the command line arguments
            if((args.Length != 1) || (!args[0].StartsWith("ami-")))
            {
                Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
                Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
Instances.");
                return;
            }

            // Create the Amazon EC2 client.
            var ec2Client = new AmazonEC2Client();

            // Create the Spot Instance request and record its ID
            Console.WriteLine("\nCreating spot instance request...");
            var req = await CreateSpotInstanceRequest(
                ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
            string requestId = req.SpotInstanceRequestId;

            // Wait for an EC2 Spot Instance to become active
            Console.WriteLine(
                $"Waiting for Spot Instance request with ID {requestId} to become active...");
            int wait = 1;
            var start = DateTime.Now;
            while(true)
            {
                Console.Write(".");

                // Get and check the status to see if the request has been fulfilled.
                var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
                if(requestInfo.Status.Code == "fulfilled")
                {
```

```
        Console.WriteLine($"\\nSpot Instance request {requestId} " +
            $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
        break;
    }

    // Wait a bit and try again, longer each time (1, 2, 4, ...)
    Thread.Sleep(wait);
    wait = wait * 2;
}

// Show the user how long it took to fulfill the Spot Instance request.
TimeSpan span = DateTime.Now.Subtract(start);
Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

// Perform actions here as needed.
// For this example, simply wait for the user to hit a key.
// That gives them a chance to look at the EC2 console to see
// the running instance if they want to.
Console.WriteLine("Press any key to start the cleanup...");
Console.ReadKey(true);

// Cancel the request.
// Do this first to make sure that the request can't be re-fulfilled
// once the Spot Instance has been terminated.
Console.WriteLine("Canceling Spot Instance request...");
await CancelSpotInstanceRequest(ec2Client, requestId);

// Terminate the Spot Instance that's running.
Console.WriteLine("Terminating the running Spot Instance...");
await TerminateSpotInstance(ec2Client, requestId);

Console.WriteLine("Done. Press any key to exit...");
Console.ReadKey(true);
}

//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,
    InstanceType instanceType, string spotPrice, int instanceCount)
{
    var launchSpecification = new LaunchSpecification{
        ImageId = amiId,
```

```
        InstanceType = instanceType
    };
    launchSpecification.SecurityGroups.Add(securityGroupName);
    var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
    };

    RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
    return result.SpotInstanceRequests[0];
}

//
// Method to get information about a Spot Instance request, including the status,
// instance ID, etc.
// It gets the information for a specific request (as opposed to all requests).
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
    return describeResponse.SpotInstanceRequests[0];
}

//
// Method to cancel a Spot Instance request
private static async Task CancelSpotInstanceRequest(
    IAmazonEC2 ec2Client, string requestId)
{
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(requestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}

//
```

```
// Method to terminate a Spot Instance
private static async Task TerminateSpotInstance(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    // Retrieve the Spot Instance request to check for running instances.
    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

    // If there are any running instances, terminate them
    if( (describeResponse.SpotInstanceRequests[0].Status.Code
        == "request-canceled-and-instance-running")
        || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
    {
        TerminateInstancesResponse response =
            await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
                InstanceIds = new List<string>(){
                    describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
            Console.WriteLine($" \n Terminated instance: {item.InstanceId}");
            Console.WriteLine($" Instance state: {item.CurrentState.Name}\n");
        }
    }
}
}
```

Ulteriori considerazioni

- Dopo aver eseguito il tutorial, è consigliabile accedere alla [console Amazon EC2](#) per verificare che la [richiesta dell'istanza Spot sia stata annullata e che l'istanza Spot](#) sia stata terminata.

Accesso AWS Identity and Access Management (IAM) con AWS SDK for .NET

I AWS SDK for .NET supporti [AWS Identity and Access Management](#), che sono un servizio web che consente AWS ai clienti di gestire gli utenti e le autorizzazioni degli utenti in AWS.

Un utente AWS Identity and Access Management (IAM) è un'entità in AWS cui crei. L'entità rappresenta una persona o un'applicazione con AWS cui interagisce. Per ulteriori informazioni sugli utenti IAM, consulta [IAM Users](#) e [IAM and STS Limits](#) nella IAM User Guide.

Concedi le autorizzazioni a un utente creando una policy IAM. La policy contiene un documento di policy che elenca le azioni che un utente può eseguire e le risorse su cui tali azioni possono influire. Per ulteriori informazioni sulle policy IAM, consulta [Policies and Permissions](#) nella IAM User Guide.

Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

API

AWS SDK for .NET Fornisce API per i clienti IAM. Le API consentono di lavorare con funzionalità IAM come utenti, ruoli e chiavi di accesso.

Questa sezione contiene un piccolo numero di esempi che mostrano i modelli che puoi seguire quando lavori con queste API. Per visualizzare il set completo di API, consulta l'[AWS SDK for .NET API Reference](#) (e scorri fino a «Amazon. IdentityManagement»).

Questa sezione contiene anche [un esempio](#) che mostra come collegare un ruolo IAM alle istanze Amazon EC2 per semplificare la gestione delle credenziali.

[Le API IAM sono fornite da. AWSSDK IdentityManagement](#) NuGetpacchetto.

Prerequisiti

Prima di iniziare, assicuratevi di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

Argomenti

- [Creazione di policy gestite da IAM da JSON](#)
- [Visualizza il documento relativo alla policy di una policy gestita da IAM](#)
- [Concessione dell'accesso utilizzando un ruolo IAM](#)

Creazione di policy gestite da IAM da JSON

Questo esempio mostra come utilizzare per AWS SDK for .NET creare una [policy gestita da IAM](#) da un determinato documento di policy in JSON. L'applicazione crea un oggetto client IAM, legge il documento di policy da un file e quindi crea la policy.

Note

Per un documento di policy di esempio in JSON, consulta le [considerazioni aggiuntive](#) alla fine di questo argomento.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Creare la policy](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Creare la policy

Il seguente frammento crea una policy gestita da IAM con il nome e il documento di policy specificati.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//
// Method to create an IAM policy from a JSON file
private static async Task<CreatePolicyResponse> CreateManagedPolicy(
    IAmazonIdentityManagementService iamClient, string policyName, string
    jsonFilename)
{
    return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.IdentityManagement](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon. IdentityManagement](#)

Classe [AmazonIdentityManagementServiceClient](#)

- [Spazio dei nomi Amazon. IdentityManagement.Modello](#)

Classe [CreatePolicyRequest](#)

Classe [CreatePolicyResponse](#)

Il codice

```
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);
}
}
}
```

Ulteriori considerazioni

- Di seguito è riportato un esempio di documento di policy che è possibile copiare in un file JSON e utilizzare come input per questa applicazione:

```
{
  "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": "*"
    }
  ]
}
```

- Puoi verificare che la policy sia stata creata cercando nella [console IAM](#). Nell'elenco a discesa Filtra policy, seleziona Customer managed. Elimina la politica quando non è più necessaria.
- Per ulteriori informazioni sulla creazione delle policy, consulta [Creating IAM Policy](#) e [IAM JSON Policy di riferimento](#) nella [IAM User Guide](#)

Visualizza il documento relativo alla policy di una policy gestita da IAM

Questo esempio mostra come utilizzare il AWS SDK for .NET per visualizzare un documento di policy. L'applicazione crea un oggetto client IAM, trova la versione predefinita della policy gestita IAM specificata e quindi visualizza il documento relativo alla policy in JSON.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Trova la versione predefinita](#)
- [Visualizza il documento relativo alla policy](#)
- [Codice completo](#)

Trova la versione predefinita

Il seguente frammento trova la versione predefinita della policy IAM specificata.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
    IAmazonIdentityManagementService iamClient, string policyArn)
{
    // Retrieve all the versions of this policy
    string defaultVersion = string.Empty;
    ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
            PolicyArn = policyArn});

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

```
    return defaultVersion;
}
```

Visualizza il documento relativo alla policy

Il seguente frammento mostra il documento di policy in JSON della policy IAM specificata.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//
// Method to retrieve and display the policy document of an IAM policy
private static async Task ShowPolicyDocument(
    IAmazonIdentityManagementService iamClient, string policyArn, string
defaultVersion)
{
    // Retrieve the policy document of the default version
    GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
            PolicyArn = policyArn,
            VersionId = defaultVersion});

    // Display the policy document (in JSON)
    Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format:");
    Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.IdentityManagement](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon. IdentityManagement](#)
Classe [AmazonIdentityManagementServiceClient](#)

- [Spazio dei nomi Amazon. IdentityManagement.Modello](#)

Classe [GetPolicyVersionRequest](#)

Classe [GetPolicyVersionResponse](#)

Classe [ListPolicyVersionsRequest](#)

Classe [ListPolicyVersionsResponse](#)

Classe [PolicyVersion](#)

Il codice

```
using System;
using System.Web;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamDisplayPolicyJson
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            if(args.Length != 1)
            {
                Console.WriteLine("\nUsage: IamDisplayPolicyJson policy-arn");
                Console.WriteLine("  policy-arn: The ARN of the policy to retrieve.");
                return;
            }
            if(!args[0].StartsWith("arn:"))
            {
                Console.WriteLine("\nCould not find policy ARN in the command-line arguments:");
                Console.WriteLine($"{args[0]}");
                return;
            }

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

```
// Retrieve and display the policy document of the given policy
string defaultVersion = await GetDefaultVersion(iamClient, args[0]);
if(string.IsNullOrEmpty(defaultVersion))
    Console.WriteLine($"Could not find the default version for policy {args[0]}.");
else
    await ShowPolicyDocument(iamClient, args[0], defaultVersion);
}

//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
    IAmazonIdentityManagementService iamClient, string policyArn)
{
    // Retrieve all the versions of this policy
    string defaultVersion = string.Empty;
    ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
            PolicyArn = policyArn});

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

    return defaultVersion;
}

//
// Method to retrieve and display the policy document of an IAM policy
private static async Task ShowPolicyDocument(
    IAmazonIdentityManagementService iamClient, string policyArn, string
defaultVersion)
{
    // Retrieve the policy document of the default version
    GetPolicyVersionResponse responsePolicy =
```

```
await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
    PolicyArn = policyArn,
    VersionId = defaultVersion});

// Display the policy document (in JSON)
Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
Console.WriteLine(
    $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
}
}
}
```

Concessione dell'accesso utilizzando un ruolo IAM

Questo tutorial mostra come utilizzare per AWS SDK for .NET abilitare i ruoli IAM sulle istanze Amazon EC2.

Panoramica

Tutte le richieste AWS devono essere firmate crittograficamente utilizzando credenziali emesse da AWS. Pertanto, è necessaria una strategia per gestire le credenziali per le applicazioni eseguite su istanze Amazon EC2. È necessario distribuire, archiviare e ruotare queste credenziali in modo sicuro, ma anche mantenerle accessibili alle applicazioni.

Con i ruoli IAM, puoi gestire efficacemente queste credenziali. Crei un ruolo IAM e lo configuri con le autorizzazioni richieste da un'applicazione, quindi colleghi quel ruolo a un'istanza EC2. Scopri di più sui vantaggi dell'utilizzo dei ruoli IAM nella [Amazon EC2 User Guide](#) o nella [Amazon EC2 User Guide](#). Consulta anche le informazioni sui [ruoli IAM](#) nella IAM User Guide.

Per un'applicazione creata utilizzando AWS SDK for .NET, quando l'applicazione costruisce un oggetto client per un AWS servizio, l'oggetto cerca le credenziali da diverse fonti potenziali. L'ordine in cui esegue la ricerca è mostrato in [Risoluzione di credenziali e profili](#)

Se l'oggetto client non trova credenziali da nessun'altra fonte, recupera credenziali temporanee con le stesse autorizzazioni di quelle che sono state configurate nel ruolo IAM e che si trovano nei metadati dell'istanza EC2. Queste credenziali vengono utilizzate per effettuare chiamate dall'oggetto client.

AWS

Informazioni sul tutorial

Seguendo questo tutorial, usi AWS SDK for .NET (e altri strumenti) per avviare un'istanza Amazon EC2 con un ruolo IAM associato, quindi vedi un'applicazione sull'istanza che utilizza le autorizzazioni del ruolo IAM.

Argomenti

- [Crea un'applicazione Amazon S3 di esempio](#)
- [Creazione di un ruolo IAM](#)
- [Avvia un'istanza EC2 e associa il ruolo IAM](#)
- [Connect all'istanza EC2](#)
- [Esegui l'applicazione di esempio sull'istanza EC2](#)
- [Eliminazione](#)

Crea un'applicazione Amazon S3 di esempio

Questa applicazione di esempio recupera un oggetto da Amazon S3. Per eseguire l'applicazione, è necessario quanto segue:

- Un bucket Amazon S3 che contiene un file di testo.
- AWS credenziali sulla tua macchina di sviluppo che ti consentono di accedere al bucket.

Per informazioni sulla creazione di un bucket Amazon S3 e sul caricamento di un oggetto, consulta la [Amazon Simple Storage Service User Guide](#). Per informazioni sulle AWS credenziali, consulta [Configura l'autenticazione SDK con AWS](#)

Creare un progetto .NET Core con il codice seguente. Quindi prova l'applicazione sulla tua macchina di sviluppo.

Note

Sul computer di sviluppo è installato il .NET Core Runtime, che consente di eseguire l'applicazione senza pubblicarla. Quando crei un'istanza EC2 più avanti in questo tutorial, puoi scegliere di installare .NET Core Runtime sull'istanza. Questo ti offre un'esperienza simile e un trasferimento di file più piccolo.

Tuttavia, puoi anche scegliere di non installare il .NET Core Runtime sull'istanza. Se scegli questa linea d'azione, devi pubblicare l'applicazione in modo che tutte le dipendenze siano incluse quando la trasferisci sull'istanza.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDKS.3](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.S3](#)

Classe [AmazonS3Client](#)

- Spazio dei nomi [Amazon.S3.Model](#)

Classe [GetObjectResponse](#)

Il codice

```
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);
}
}
```

```
}
```

Se lo desideri, puoi rimuovere temporaneamente le credenziali che usi sulla tua macchina di sviluppo per vedere come risponde l'applicazione. (Ma assicuratevi di ripristinare le credenziali quando avete finito.)

Creazione di un ruolo IAM

Crea un ruolo IAM con le autorizzazioni appropriate per accedere ad Amazon S3.

1. Apri la [console IAM](#).
2. Nel riquadro di navigazione, scegli Ruoli, quindi scegli Crea ruolo.
3. Seleziona AWS servizio, trova e scegli EC2, quindi scegli Avanti: Autorizzazioni.
4. In Allega politiche di autorizzazione, trova e seleziona AmazonS3. ReadOnlyAccess Se lo desideri, esamina la politica, quindi scegli Avanti: tag.
5. Aggiungi i tag se lo desideri, quindi scegli Avanti: revisione.
6. Digita un nome e una descrizione per il ruolo e quindi scegli Create role (Crea ruolo). Ricorda questo nome perché sarà necessario all'avvio dell'istanza EC2.

Avvia un'istanza EC2 e associa il ruolo IAM

Avvia un'istanza EC2 con il ruolo IAM creato in precedenza. Puoi farlo nei seguenti modi.

- Utilizzo della console EC2

Segui le istruzioni per avviare un'istanza nella [Amazon EC2 User Guide o nella Amazon EC2 User Guide](#).

Durante la procedura guidata, dovresti almeno visitare la pagina Configure Instance Details in modo da poter selezionare il ruolo IAM che hai creato in precedenza.

- Utilizzando il AWS SDK for .NET

Per informazioni su questo argomento [Avvio di un'istanza Amazon EC2](#), vedere, [Ulteriori considerazioni](#) inclusa la fine dell'argomento.

Per avviare un'istanza EC2 a cui è associato un ruolo IAM, la configurazione di un utente IAM deve includere determinate autorizzazioni. [Per ulteriori informazioni sulle autorizzazioni richieste, consulta la Amazon EC2 User Guide o la Amazon EC2 User Guide.](#)

Connect all'istanza EC2

Connect all'istanza EC2 in modo da poter trasferire l'applicazione di esempio su di essa e quindi eseguire l'applicazione. Avrai bisogno del file che contiene la parte privata della key pair che hai usato per avviare l'istanza, ovvero il file PEM.

Puoi farlo seguendo la procedura di connessione nella Guida per l'utente di [Amazon EC2 o nella Guida per l'utente di Amazon EC2](#). Quando ti connetti, fallo in modo da poter trasferire i file dalla macchina di sviluppo all'istanza.

Se usi Visual Studio su Windows, puoi anche connetterti all'istanza utilizzando Toolkit for Visual Studio. Per ulteriori informazioni, consulta [Connessione a un'istanza Amazon EC2 nella Guida](#) per l'AWS Toolkit for Visual Studio utente.

Esegui l'applicazione di esempio sull'istanza EC2

1. Copia i file dell'applicazione dall'unità locale all'istanza.

I file da trasferire dipendono da come è stata creata l'applicazione e dal fatto che nell'istanza sia installato o meno il.NET Core Runtime. Per informazioni su come trasferire file sulla tua istanza, consulta la [Amazon EC2 User Guide](#) o la [Amazon EC2 User Guide](#).

2. Avvia l'applicazione e verifica che funzioni con gli stessi risultati della macchina di sviluppo.
3. Verifica che l'applicazione utilizzi le credenziali fornite dal ruolo IAM.
 - a. Aprire la [console di Amazon EC2](#).
 - b. Seleziona l'istanza e scollega il ruolo IAM tramite Actions, Instance Settings, Attach/Replace IAM Role.
 - c. Esegui nuovamente l'applicazione e verifica che restituisca un errore di autorizzazione.

Eliminazione

Quando hai finito con questo tutorial e se non desideri più l'istanza EC2 che hai creato, assicurati di terminare l'istanza per evitare costi indesiderati. Puoi farlo nella [console Amazon EC2](#) o a livello di codice, come descritto in [Terminazione di un'istanza Amazon EC2](#). Se lo desideri, puoi anche

eliminare altre risorse che hai creato per questo tutorial. Queste potrebbero includere un ruolo IAM, una coppia di chiavi EC2 e un file PEM, un gruppo di sicurezza, ecc.

Utilizzo dello storage Internet di Amazon Simple Storage Service

AWS SDK for .NET Supporta [Amazon S3](#), che è lo storage per Internet. È concepito per rendere più accessibili agli sviluppatori risorse informatiche su grande scala per il Web.

API

AWS SDK for .NET Fornisce API per i client Amazon S3. Le API consentono di lavorare con risorse Amazon S3 come bucket e item. Per visualizzare il set completo di API per Amazon S3, consulta quanto segue:

- [AWS SDK for .NET Riferimento all'API](#) (e scorri fino a «Amazon.S3").
- [Documentazione](#) sulla crittografia Amazon.Extensions.S3

Le API di Amazon S3 sono fornite dai seguenti pacchetti: NuGet

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

Prerequisiti

[Prima di iniziare, assicurati di aver configurato l'ambiente e il progetto.](#) Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Esempi in questo documento

I seguenti argomenti di questo documento mostrano come utilizzarlo AWS SDK for .NET per lavorare con Amazon S3.

- [Utilizzo delle chiavi KMS per la crittografia S3](#)

Esempi in altri documenti

I seguenti collegamenti alla [Amazon S3 Developer Guide](#) forniscono ulteriori esempi su come utilizzarla per AWS SDK for .NET lavorare con Amazon S3.

Note

Sebbene questi esempi e considerazioni aggiuntive sulla programmazione siano stati creati per la versione 3 di Using .NET Framework, sono validi anche per le versioni successive di che AWS SDK for .NET utilizzano .NET Core. AWS SDK for .NET Talvolta sono necessarie piccole modifiche al codice.

Esempi di programmazione in Amazon S3

- [Gestione delle ACL](#)
- [Creazione di un bucket](#)
- [Caricamento di un oggetto](#)
- [Caricamento multiparte con l'API di alto livello \(Amazon.S3.Transfer\). TransferUtility\)](#)
- [Caricamento in più parti con l'API di basso livello](#)
- [Elenco degli oggetti](#)
- [Elenco delle chiavi](#)
- [Recupero di un oggetto](#)
- [Copia di un oggetto](#)
- [Copia di un oggetto con l'API per il caricamento in più parti](#)
- [Eliminazione di un oggetto](#)
- [Eliminazione di più oggetti](#)
- [Ripristino di un oggetto](#)
- [Configurazione di un bucket per le notifiche](#)
- [Gestione del ciclo di vita di un oggetto](#)
- [Generazione di un URL prefirmato per un oggetto](#)
- [Gestione di siti Web](#)
- [Abilitazione della condivisione di risorse tra le origini \(CORS\)](#)

Considerazioni aggiuntive sulla programmazione

- [Utilizzo di AWS SDK for .NET per la programmazione in Amazon S3](#)
- [Esecuzione di richieste mediante le credenziali temporanee per gli utenti IAM](#)

- [Esecuzione di richieste mediante le credenziali temporanee per gli utenti federati](#)
- [Specifica della crittografia lato server](#)
- [Specifica della crittografia lato server con chiavi di crittografia fornite dal cliente](#)

Utilizzo AWS KMS delle chiavi per la crittografia Amazon S3 nel AWS SDK for .NET

Questo esempio mostra come utilizzare AWS Key Management Service le chiavi per crittografare gli oggetti Amazon S3. L'applicazione crea una chiave master del cliente (CMK) e la utilizza per creare un oggetto [AmazonS3 EncryptionClient](#) V2 per la crittografia lato client. L'applicazione utilizza quel client per creare un oggetto crittografato da un determinato file di testo in un bucket Amazon S3 esistente. Quindi decifra l'oggetto e ne visualizza il contenuto.

Warning

Una classe simile chiamata `AmazonS3EncryptionClient` è obsoleta ed è meno sicura della classe. `AmazonS3EncryptionClientV2` Per migrare il codice esistente che utilizza, consulta. `AmazonS3EncryptionClient` [Migrazione del client di crittografia S3](#)

Argomenti

- [Creare materiali di crittografia](#)
- [Crea e crittografa un oggetto Amazon S3](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Creare materiali di crittografia

Il seguente frammento crea un `EncryptionMaterials` oggetto che contiene un ID di chiave KMS.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
// Create a customer master key (CMK) and store the result
CreateKeyResponse createKeyResponse =
    await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
var kmsEncryptionContext = new Dictionary<string, string>();
```

```
var kmsEncryptionMaterials = new EncryptionMaterialsV2(  
    createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);
```

Crea e crittografa un oggetto Amazon S3

Il seguente frammento crea un `AmazonS3EncryptionClientV2` oggetto che utilizza i materiali di crittografia creati in precedenza. Utilizza quindi il client per creare e crittografare un nuovo oggetto Amazon S3.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to create and encrypt an object in an S3 bucket  
static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(  
    EncryptionMaterialsV2 materials, string bucketName,  
    string fileName, string itemName)  
{  
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials  
    var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)  
    {  
        StorageMode = CryptoStorageMode.ObjectMetadata  
    };  
    var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);  
  
    // Create, encrypt, and put the object  
    await s3EncClient.PutObjectAsync(new PutObjectRequest  
    {  
        BucketName = bucketName,  
        Key = itemName,  
        ContentBody = File.ReadAllText(fileName)  
    });  
  
    // Get, decrypt, and return the object  
    return await s3EncClient.GetObjectAsync(new GetObjectRequest  
    {  
        BucketName = bucketName,  
        Key = itemName  
    });  
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

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

Elementi di programmazione:

- [Spazio dei nomi Amazon.Extensions.S3.Encryption](#)

Classe [AmazonS3 V2 EncryptionClient](#)

Classe [AmazonS3 CryptoConfiguration V2](#)

Classe [CryptoStorageMode](#)

Classe [EncryptionMaterialsV2](#)

- [Spazio dei nomi Amazon.Extensions.S3.Encryption.Primitives](#)

Classe [KmsType](#)

- [Namespace Amazon.S3.Model](#)

Classe [GetObjectRequest](#)

Classe [GetObjectResponse](#)

Classe [PutObjectRequest](#)

- [Spazio dei nomi Amazon. KeyManagementService](#)

Classe [AmazonKeyManagementServiceClient](#)

- [Spazio dei nomi Amazon. KeyManagementService.Modello](#)

Classe [CreateKeyRequest](#)

Classe [CreateKeyResponse](#)

Il codice

```
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);  
    }  
}
```

Ulteriori considerazioni

- Puoi controllare i risultati di questo esempio. Per farlo, vai alla [console Amazon S3](#) e apri il bucket che hai fornito all'applicazione. Quindi trova il nuovo oggetto, scaricalo e aprilo in un editor di testo.
- La classe [AmazonS3 EncryptionClient V2](#) implementa la stessa interfaccia della classe standard. `AmazonS3Client` Ciò semplifica il trasferimento del codice alla `AmazonS3EncryptionClientV2` classe in modo che la crittografia e la decrittografia avvengano automaticamente e in modo trasparente nel client.
- Uno dei vantaggi dell'utilizzo di una AWS KMS chiave come chiave principale è che non è necessario archiviare e gestire le proprie chiavi master; questa operazione viene eseguita da. AWS Un secondo vantaggio è che la `AmazonS3EncryptionClientV2` classe di AWS SDK for .NET è interoperabile con la `AmazonS3EncryptionClientV2` classe di. AWS SDK for Java Ciò significa che è possibile crittografare con AWS SDK for Java e decrittografare con, e viceversa. AWS SDK for .NET

Note

La `AmazonS3EncryptionClientV2` classe di AWS SDK for .NET supporta le chiavi master KMS solo se eseguita in modalità metadati. La modalità del file di istruzioni della `AmazonS3EncryptionClientV2` classe di AWS SDK for .NET è incompatibile con la `AmazonS3EncryptionClientV2` classe di. AWS SDK for Java

- Per ulteriori informazioni sulla crittografia lato client con la `AmazonS3EncryptionClientV2` classe e su come funziona la crittografia a busta, consulta [Client Side Data Encryption with e Amazon AWS SDK for .NET S3](#).

Invio di notifiche dal cloud utilizzando Amazon Simple Notification Service

Note

Le informazioni in questo argomento sono specifiche per i progetti basati su .NET Framework e la AWS SDK for .NET versione 3.3 e precedenti.

AWS SDK for .NET Supporta Amazon Simple Notification Service (Amazon SNS), un servizio Web che consente ad applicazioni, utenti finali e dispositivi di inviare istantaneamente notifiche dal cloud. Per ulteriori dettagli, consulta la pagina [Amazon SNS](#).

Elencare gli argomenti relativi ad Amazon SNS

L'esempio seguente mostra come elencare gli argomenti di Amazon SNS, le sottoscrizioni a ciascun argomento e gli attributi per ogni argomento. Questo esempio utilizza l'impostazione predefinita.

[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));
```

Invio di un messaggio a un argomento Amazon SNS

L'esempio seguente mostra come inviare un messaggio a un argomento di Amazon SNS. L'esempio utilizza un argomento, l'ARN dell'argomento Amazon SNS.

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

using Amazon;
using Amazon.SimpleNotificationService;
```

```
using Amazon.SimpleNotificationService.Model;

namespace SnsSendMessage
{
    class Program
    {
        static void Main(string[] args)
        {
            /* Topic ARNs must be in the correct format:
             *   arn:aws:sns:REGION:ACCOUNT_ID:NAME
             *
             * where:
             *   REGION      is the region in which the topic is created, such as us-
west-2
             *   ACCOUNT_ID is your (typically) 12-character account ID
             *   NAME        is the name of the topic
             */
            string topicArn = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
Amazon.RegionEndpoint.USWest2);

            var request = new PublishRequest
            {
                Message = message,
                TopicArn = topicArn
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to topic:");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

```
}
```

Guarda l'[esempio completo](#), che include informazioni su come creare ed eseguire l'esempio dalla riga di comando, su. GitHub

Invio di un messaggio SMS a un numero di telefono

L'esempio seguente mostra come inviare un messaggio SMS a un numero di telefono. L'esempio prende un argomento, il numero di telefono, che deve essere in uno dei due formati descritti nei commenti.

```
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);
    }
}
}
```

Guarda l'[esempio completo](#), che include informazioni su come creare ed eseguire l'esempio dalla riga di comando, su GitHub.

Messaggistica tramite Amazon SQS

AWS SDK for .NET Supporta [Amazon Simple Queue Service \(Amazon SQS\)](#), un servizio di accodamento dei messaggi che gestisce i messaggi o i flussi di lavoro tra i componenti di un sistema.

Le code Amazon SQS forniscono un meccanismo che consente di inviare, archiviare e ricevere messaggi tra componenti software come microservizi, sistemi distribuiti e applicazioni serverless. Ciò consente di disaccoppiare tali componenti e ti libera dalla necessità di progettare e gestire il tuo sistema di messaggistica. Per informazioni sul funzionamento delle code e dei messaggi in Amazon SQS, consulta i tutorial di Amazon SQS [e l'architettura Amazon SQS Basic nella Amazon Simple Queue Service Developer Guide](#).

Important

A causa della natura distribuita delle code, Amazon SQS non può garantire che riceverai i messaggi nell'ordine preciso in cui vengono inviati. Se devi mantenere l'ordine dei messaggi, usa una coda [FIFO di Amazon SQS](#).

API

AWS SDK for .NET Fornisce API per i client Amazon SQS. Le API ti consentono di lavorare con funzionalità di Amazon SQS come code e messaggi. Questa sezione contiene un piccolo numero di esempi che mostrano gli schemi che puoi seguire quando lavori con queste API. Per visualizzare il set completo di API, consulta l'[AWS SDK for .NET API Reference](#) (e scorri fino a «Amazon.sqs»).

[Le API di Amazon SQS sono fornite dal pacchetto.SQS. AWSSDK](#) NuGet

Prerequisiti

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

Argomenti

- [Creazione di code Amazon SQS](#)
- [Aggiornamento delle code Amazon SQS](#)
- [Eliminazione delle code Amazon SQS](#)
- [Invio di messaggi Amazon SQS](#)
- [Ricezione di messaggi Amazon SQS](#)

Creazione di code Amazon SQS

Questo esempio mostra come utilizzare per AWS SDK for .NET creare una coda Amazon SQS. L'applicazione crea una [coda di lettere](#) non scritte se non viene fornito l'ARN corrispondente. Quindi crea una coda di messaggi standard, che include una coda di lettere non scritte (quella che hai fornito o quella che è stata creata).

Se non fornite alcun argomento della riga di comando, l'applicazione mostra semplicemente le informazioni su tutte le code esistenti.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Mostra le code esistenti](#)
- [Crea la coda](#)
- [Ottieni l'ARN di una coda](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Mostra le code esistenti

Il seguente frammento mostra un elenco delle code esistenti nella regione del client SQS e gli attributi di ciascuna coda.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

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

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

Crea la coda

Il seguente frammento crea una coda. Lo snippet include l'uso di una coda di lettere morte, ma una coda di lettere morte non è necessariamente necessaria per le code.

L'esempio alla [fine di questo argomento mostra questo frammento in](#) uso.

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



```

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

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

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

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

```

Ottieni l'ARN di una coda

Il seguente frammento ottiene l'ARN della coda identificata dall'URL della coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```

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

```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.SQS](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.sqs](#)

Classe [AmazonSQSClient](#)

Classe [QueueAttributeName](#)

- [Spazio dei nomi Amazon.SQS.Model](#)

Classe [CreateQueueRequest](#)

Classe [CreateQueueResponse](#)

Classe [GetQueueAttributesResponse](#)

Classe [ListQueuesResponse](#)

Il codice

```
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($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"{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);
    }
}
}
```

Ulteriori considerazioni

- Il nome della coda deve essere composto da caratteri alfanumerici, trattini e caratteri di sottolineatura.
 - I nomi e gli URL delle code fanno distinzione tra maiuscole e minuscole
 - Se hai bisogno dell'URL della coda ma hai solo il nome della coda, usa uno dei metodi. `AmazonSQSClient.GetQueueUrlAsync`
 - Per informazioni sui vari attributi di coda che puoi impostare, consulta [CreateQueueRequest](#) l'[AWS SDK for .NET API Reference](#) o [SetQueueAttributes](#) il [Amazon Simple Queue Service API Reference](#).
 - Questo esempio specifica un polling lungo per tutti i messaggi sulla coda che crei. Questa operazione viene eseguita utilizzando l'attributo. `ReceiveMessageWaitTimeSeconds`

Puoi anche specificare un polling lungo durante una chiamata ai `ReceiveMessageAsync` metodi della classe [AmazonSQSClient](#). Per ulteriori informazioni, consulta [Ricezione di messaggi Amazon SQS](#).
- Per informazioni sul polling breve rispetto al polling lungo, [consulta Short and long polling nella Amazon Simple Queue Service Developer Guide](#).

- Una coda di lettere morte è quella che altre code (di origine) possono indirizzare per i messaggi che non vengono elaborati correttamente. Per ulteriori informazioni, consulta le [code di lettera morta di Amazon SQS nella Guida](#) per gli sviluppatori di Amazon Simple Queue Service.
- Puoi anche visualizzare l'elenco delle code e i risultati di questo esempio nella console [Amazon SQS](#).

Aggiornamento delle code Amazon SQS

Questo esempio mostra come utilizzare per AWS SDK for .NET aggiornare una coda Amazon SQS. Dopo alcuni controlli, l'applicazione aggiorna l'attributo specificato con il valore dato, quindi mostra tutti gli attributi per la coda.

Se negli argomenti della riga di comando è incluso solo l'URL della coda, l'applicazione mostra semplicemente tutti gli attributi della coda.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Mostra gli attributi della coda](#)
- [Convalida il nome dell'attributo](#)
- [Aggiorna l'attributo della coda](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Mostra gli attributi della coda

Il seguente frammento mostra gli attributi della coda identificata dall'URL della coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

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

```

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

```

Convalida il nome dell'attributo

Il seguente frammento di codice convalida il nome dell'attributo da aggiornare.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```

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

```

Aggiorna l'attributo della coda

Il seguente frammento aggiorna un attributo della coda identificato dall'URL della coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```

//
// Method to update a queue attribute
private static async Task UpdateAttribute(
    IAmazonSQS sqsClient, string qUrl, string attribute, string value)
{
    await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
}

```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.SQS](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.sqs](#)

Classe [AmazonSQSClient](#)

Classe [QueueAttributeName](#)

- [Spazio dei nomi Amazon.SQS.Model](#)

Classe [GetQueueAttributesResponse](#)

Il codice

```
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($"The 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($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"{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);
}
}
}
```

Ulteriori considerazioni

- Per aggiornare l'`RedrivePolicy` attributo, è necessario citare l'intero valore ed evitare le virgolette per le coppie chiave/valore, a seconda del sistema operativo in uso.

In Windows, ad esempio, il valore è costruito in modo simile al seguente:

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

Eliminazione delle code Amazon SQS

Questo esempio mostra come utilizzare per AWS SDK for .NET eliminare una coda Amazon SQS. L'applicazione elimina la coda, attende fino a un determinato periodo di tempo che la coda scompaia, quindi mostra un elenco delle code rimanenti.

Se non si fornisce alcun argomento della riga di comando, l'applicazione mostra semplicemente un elenco delle code esistenti.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Eliminare la coda](#)
- [Attendi che la coda finisca](#)
- [Mostra un elenco di code esistenti](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Eliminare la coda

Il seguente frammento elimina la coda identificata dall'URL di coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//  
// Method to delete an SQS queue  
private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)  
{  
    Console.WriteLine($"Deleting queue {qUrl}...");
```



```
await sqsClient.DeleteQueueAsync(qUrl);
Console.WriteLine($"Queue {qUrl} has been deleted.");
}
```

Attendi che la coda finisca

Il seguente frammento attende il completamento del processo di eliminazione, che potrebbe richiedere 60 secondi.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//
// Method to wait up to a given number of seconds
private static async Task Wait(
    IAmazonSQS sqsClient, int numSeconds, string qUrl)
{
    Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
    Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
    for(int i=0; i<numSeconds; i++)
    {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
            if(url == qUrl)
            {
                found = true;
                break;
            }
        }
        if(!found) break;
    }
}
```

Mostra un elenco di code esistenti

Il seguente frammento mostra un elenco delle code esistenti nella regione del client SQS.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//  
// Method to show a list of the existing queues  
private static async Task ListQueues(IAmazonSQS sqsClient)  
{  
    ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");  
    Console.WriteLine("\nList of queues:");  
    foreach(var qUrl in responseList.QueueUrls)  
        Console.WriteLine($"- {qUrl}");  
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.SQS](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.sqs](#)
Classe [AmazonSQSClient](#)
- Spazio dei nomi [Amazon.SQS.Model](#)
Classe [ListQueuesResponse](#)

Il codice

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

Ulteriori considerazioni

- La chiamata `DeleteQueueAsync` API non verifica se la coda che stai eliminando viene utilizzata come coda di lettere non scritte. Una procedura più sofisticata potrebbe verificarlo.
- Puoi anche visualizzare l'elenco delle code e i risultati di questo esempio nella console [Amazon SQS](#).

Invio di messaggi Amazon SQS

[Questo esempio mostra come utilizzare per inviare messaggi AWS SDK for .NET a una coda Amazon SQS, che puoi creare a livello di codice o utilizzando la console Amazon SQS.](#)

L'applicazione invia un singolo messaggio alla coda e poi un batch di messaggi. L'applicazione attende quindi l'input dell'utente, che può essere costituito da messaggi aggiuntivi da inviare alla coda o da una richiesta di uscita dall'applicazione.

Questo esempio e il [prossimo esempio sulla ricezione di messaggi](#) possono essere usati insieme per visualizzare il flusso dei messaggi in Amazon SQS.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Invio di un messaggio](#)
- [Inviare un batch di messaggi](#)
- [Elimina tutti i messaggi dalla coda](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Invio di un messaggio

Il seguente frammento invia un messaggio alla coda identificata dall'URL della coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
{
    SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}
```

Inviare un batch di messaggi

Il seguente frammento invia un batch di messaggi alla coda identificata dall'URL di coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
    Console.WriteLine($" \nSending a batch of messages to queue\n {qUrl}");
    SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
}
```

Elimina tutti i messaggi dalla coda

Il seguente frammento elimina tutti i messaggi dalla coda identificata dall'URL di coda specificato. Questa operazione è nota anche come eliminazione della coda.

L'esempio [alla fine di questo argomento mostra questo](#) frammento in uso.

```
//  
// Method to delete all messages from the queue  
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)  
{  
    Console.WriteLine($"\\nPurging messages from queue\\n {qUrl}...");  
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);  
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");  
}
```

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.SQS](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.sqs](#)
 - Classe [AmazonSQSClient](#)
- Spazio dei nomi [Amazon.SQS.Model](#)
 - Classe [PurgeQueueResponse](#)
 - Classe [SendMessageBatchResponse](#)
 - Classe [SendMessageResponse](#)
 - Classe [SendMessageBatchRequestEntry](#)
 - Classe [SendMessageBatchResultEntry](#)

Il codice

```
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($"Purging messages from queue\n {qUrl}...");  
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);  
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");  
}  
}
```

Ulteriori considerazioni

- Per informazioni sulle varie limitazioni sui messaggi, inclusi i caratteri consentiti, consulta [Quote relative ai messaggi](#) nella [Amazon Simple Queue Service Developer Guide](#).
- I messaggi rimangono in coda finché non vengono eliminati o la coda non viene eliminata. Quando un messaggio viene ricevuto da un'applicazione, non sarà visibile nella coda anche se esiste ancora nella coda. Per ulteriori informazioni sui timeout di visibilità, consulta [Amazon SQS visibility timeout](#).

- Oltre al corpo del messaggio, puoi anche aggiungere attributi ai messaggi. Per ulteriori informazioni, consulta [Metadati dei messaggi](#).

Ricezione di messaggi Amazon SQS

[Questo esempio mostra come utilizzare per AWS SDK for .NET ricevere messaggi da una coda Amazon SQS, che puoi creare a livello di codice o utilizzando la console Amazon SQS.](#)

L'applicazione legge un singolo messaggio dalla coda, elabora il messaggio (in questo caso, visualizza il corpo del messaggio sulla console) e quindi elimina il messaggio dalla coda.

L'applicazione ripete questi passaggi finché l'utente non digita un tasto sulla tastiera.

Questo esempio e l'[esempio precedente sull'invio di messaggi](#) possono essere usati insieme per visualizzare il flusso dei messaggi in Amazon SQS.

Le sezioni seguenti forniscono frammenti di questo esempio. Successivamente viene mostrato [il codice completo dell'esempio](#), che può essere creato ed eseguito così com'è.

Argomenti

- [Ricevi un messaggio](#)
- [Eliminare un messaggio](#)
- [Codice completo](#)
- [Ulteriori considerazioni](#)

Ricevi un messaggio

Il seguente frammento riceve un messaggio dalla coda identificata dall'URL di coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

```
//  
// Method to read a message from the given queue  
// In this example, it gets one message at a time  
private static async Task<ReceiveMessageResponse> GetMessage(  
    IAmazonSQS sqsClient, string qUrl, int waitTime=0)  
{  
    return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{  
        QueueUrl=qUrl,  
        MaxNumberOfMessages=MaxMessages,
```

```
        WaitTimeSeconds=waitTime
        // (Could also request attributes, set visibility timeout, etc.)
    });
}
```

Eliminare un messaggio

Il seguente frammento elimina un messaggio dalla coda identificata dall'URL di coda specificato.

L'esempio [alla fine di questo argomento mostra questo frammento](#) in uso.

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

Codice completo

Questa sezione mostra i riferimenti pertinenti e il codice completo per questo esempio.

Riferimenti SDK

NuGet pacchetti:

- [AWSSDK.SQS](#)

Elementi di programmazione:

- [Spazio dei nomi Amazon.sqs](#)
 - Classe [AmazonSQSClient](#)
- Spazio dei nomi [Amazon.SQS.Model](#)
 - Classe [ReceiveMessageRequest](#)
 - Classe [ReceiveMessageResponse](#)

Il codice

```
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);
}
}
}
```

Ulteriori considerazioni

- Per specificare un polling lungo, in questo esempio viene utilizzata la `WaitTimeSeconds` proprietà per ogni chiamata al `ReceiveMessageAsync` metodo.

È inoltre possibile specificare un polling lungo per tutti i messaggi in una coda utilizzando l'`ReceiveMessageWaitTimeSeconds` attributo durante la [creazione](#) o l'[aggiornamento](#) della coda.

Per informazioni sul polling breve rispetto al polling lungo, [consulta Short and long polling nella Amazon Simple Queue Service Developer Guide](#).

- Durante l'elaborazione dei messaggi, puoi utilizzare la maniglia di ricezione per modificare il timeout di visibilità dei messaggi. Per informazioni su come eseguire questa operazione, consulta i `ChangeMessageVisibilityAsync` metodi della classe [AmazonSQSClient](#).
- La chiamata al `DeleteMessageAsync` metodo rimuove incondizionatamente il messaggio dalla coda, indipendentemente dall'impostazione del timeout di visibilità.

Utilizzo AWS Lambda per il servizio di calcolo

I AWS SDK for .NET supportano AWS Lambda, che consentono di eseguire il codice senza effettuare il provisioning o la gestione di server. Per ulteriori informazioni, consulta la [pagina AWS Lambda del prodotto](#) e la [Guida per gli AWS Lambda sviluppatori](#), in particolare la sezione [Working with C#](#).

API

AWS SDK for .NET fornisce API per AWS Lambda [Le API consentono di utilizzare funzionalità Lambda come funzioni, trigger ed eventi](#). Per visualizzare il set completo di API, consulta [Lambda AWS SDK for .NET nel riferimento API](#).

[Le API Lambda sono fornite tramite pacchetti. NuGet](#)

Prerequisiti

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

Argomenti

- [Usare le annotazioni per scrivereAWS Lambdafunzioni](#)

Usare le annotazioni per scrivereAWS Lambdafunzioni

Quando si scrivono funzioni Lambda, a volte è necessario scrivere una grande quantità di codice del gestore e aggiornarleAWS CloudFormationmodelli, tra le altre attività. Lambda Annotations è un framework che aiuta ad alleggerire questi oneri per le funzioni Lambda .NET 6, rendendo così l'esperienza di scrittura di Lambda più naturale in C#.

Come esempio dei vantaggi derivanti dall'utilizzo del framework Lambda Annotations, considera i seguenti frammenti di codice che aggiungono due numeri.

Senza annotazioni Lambda

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

Con annotazioni Lambda

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

Come mostrato nell'esempio, Lambda Annotations può eliminare la necessità di un determinato codice di targa della caldaia.

Per informazioni dettagliate su come utilizzare il framework e per ulteriori informazioni, consulta le seguenti risorse:

- [LaGitHubLEGGIMI](#) per la documentazione sulle API e gli attributi di Lambda Annotations.
- [La post sul blog](#) per Lambda Annotations.
- [LaAmazon.Lambda.Annotations](#) NuGet pacchetto.
- [LaProgetto Photo Asset Management](#) sul GitHub. Nello specifico, vedi [PamApiAnnotations](#) cartella e riferimenti a Lambda Annotations nel progetto [LEGGIMI](#).

Librerie e framework di alto livello per AWS SDK for .NET

Le sezioni seguenti contengono informazioni su librerie e framework di alto livello che non fanno parte delle funzionalità principali dell'SDK. Queste librerie e framework utilizzano le funzionalità SDK di base per creare funzionalità che semplificano determinate attività.

Se non lo conosci AWS SDK for .NET, ti consigliamo di dare prima un'occhiata all'[Fai un breve tour](#) argomento. Fornisce un'introduzione all'SDK.

Prima di iniziare, assicurati di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Argomenti

- [AWS Framework di elaborazione dei messaggi per.NET](#)

AWS Framework di elaborazione dei messaggi per.NET

Questa è la documentazione non definitiva per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework for .NET è un framework AWS nativo che semplifica lo sviluppo di applicazioni di elaborazione dei messaggi.NET che utilizzano AWS servizi come Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon EventBridge. Il framework riduce la quantità di codice standard che gli sviluppatori devono scrivere, consentendoti di concentrarti sulla logica aziendale durante la pubblicazione e l'utilizzo di messaggi. Per informazioni dettagliate su come il framework può semplificare lo sviluppo, consulta il post di blog [Introducing the AWS Message Processing Framework for .NET \(Preview\)](#). La prima parte, in particolare, fornisce una dimostrazione che mostra la differenza tra l'utilizzo di chiamate API di basso livello e l'utilizzo del framework.

Il Message Processing Framework supporta le seguenti attività e funzionalità:

- Invio di messaggi a SQS e pubblicazione di eventi su SNS e EventBridge
- Ricezione e gestione di messaggi da SQS utilizzando un poller a esecuzione prolungata, generalmente utilizzato nei servizi in background. Ciò include la gestione del timeout di visibilità durante la gestione di un messaggio per impedire ad altri client di elaborarlo.
- Gestione dei messaggi nelle funzioni AWS Lambda .
- Code FIFO (first-in-first-out) SQS e argomenti SNS.
- OpenTelemetry per la registrazione.

Per dettagli su queste attività e funzionalità, consulta la sezione Caratteristiche del [post del blog](#) e gli argomenti elencati di seguito.

Prima di iniziare, assicuratevi di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Altre risorse

- Il [AWS.Messaging](#) pacchetto su [NuGet.org](#).
- Il [riferimento all'API](#).
- Il [README file nel GitHub repository all'indirizzo https://github.com/aws-labs/aws-dotnet-messaging](#)
- [Iniezione di dipendenze.NET](#) da Microsoft.
- [.NET Generic Host](#) di Microsoft.

Argomenti

- [Inizia a usare il AWS Message Processing Framework per .NET](#)
- [Pubblica messaggi con il AWS Message Processing Framework per .NET](#)
- [Consuma messaggi con il AWS Message Processing Framework for .NET](#)
- [Utilizzo di FIFO con il AWS Message Processing Framework per .NET](#)
- [Registrazione e telemetria aperta per il AWS Message Processing Framework per.NET](#)
- [Personalizzazione del framework di elaborazione dei AWS messaggi per.NET](#)
- [Sicurezza per il AWS Message Processing Framework per.NET](#)

Inizia a usare il AWS Message Processing Framework per .NET

Questa è la documentazione preliminare per una funzionalità disponibile nella versione di anteprima. ed è soggetta a modifiche.

Prima di iniziare, assicuratevi di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Questo argomento fornisce informazioni utili per iniziare a utilizzare il Message Processing Framework. Oltre alle informazioni sui prerequisiti e sulla configurazione, viene fornito un tutorial che mostra come implementare uno scenario comune.

Prerequisiti e configurazione

- Le credenziali fornite per l'applicazione devono disporre delle autorizzazioni appropriate per il servizio di messaggistica e le operazioni utilizzate. Per ulteriori informazioni, consulta gli argomenti sulla sicurezza per [SQS](#), [SNS](#) e [EventBridge](#) nelle rispettive guide per sviluppatori.
- Per utilizzare il AWS Message Processing Framework per .NET, è necessario aggiungere il [AWS.Messaging](#) NuGetpacchetto al progetto. Per esempio:

```
dotnet add package AWS.Messaging
```

- Il framework si integra con il contenitore di [servizi di dependency injection \(DI\)](#) di .NET. È possibile configurare il framework durante l'avvio dell'applicazione chiamando `AddAWSMessageBus` per aggiungerlo al contenitore DI.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

Tutorial

Questo tutorial dimostra come utilizzare il AWS Message Processing Framework per .NET. Crea due applicazioni: un'API ASP.NET Core Minimal che invia messaggi a una coda Amazon SQS quando riceve una richiesta su un endpoint API e un'applicazione console di lunga durata che esegue il polling di questi messaggi e li gestisce.

- Le istruzioni di questo tutorial privilegiano l'interfaccia della riga di comando .NET, ma è possibile eseguire questo tutorial utilizzando strumenti multiplatforma, come .NET CLI o Microsoft Visual Studio. Per informazioni sugli strumenti, consulta [Installa e configura la tua toolchain](#)
- Questo tutorial presuppone che tu stia utilizzando il tuo [default] profilo per le credenziali. Si presuppone inoltre che le credenziali a breve termine siano disponibili con le autorizzazioni appropriate per l'invio e la ricezione di messaggi Amazon SQS. [Per ulteriori informazioni, consulta gli argomenti sulla Configura l'autenticazione SDK con AWS sicurezza per SQS.](#)

Note

Eseguendo questo tutorial, potresti incorrere in costi per la messaggistica SQS.

Fasi

- [Crea una coda SQS](#)
- [Crea ed esegui l'applicazione di pubblicazione](#)
- [Crea ed esegui l'applicazione di gestione](#)
- [Rimozione](#)

Crea una coda SQS

Questo tutorial richiede una coda SQS per inviare e ricevere messaggi. È possibile creare una coda utilizzando uno dei seguenti comandi per il o il AWS CLI . AWS Tools for PowerShell Prendi nota dell'URL della coda che viene restituito in modo da poterlo specificare nella configurazione del framework che segue.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

AWS Tools for PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Crea ed esegui l'applicazione di pubblicazione

Utilizzate la procedura seguente per creare ed eseguire l'applicazione di pubblicazione.

1. Aprire un prompt dei comandi o un terminale. Trovare o creare una cartella del sistema operativo in cui è possibile creare un progetto.NET.
2. In tale cartella, eseguire il seguente comando per creare il progetto.NET.

```
dotnet new webapi --name Publisher
```

3. Naviga nella cartella del nuovo progetto. Aggiungi una dipendenza dal AWS Message Processing Framework per .NET.

```
cd Publisher
dotnet add package AWS.Messaging
```

Note

Se lo utilizzi AWS IAM Identity Center per l'autenticazione, assicurati di aggiungere anche `AWSSDK.SSO` e `AWSSDK.SSO0IDC`.

4. Sostituisci il codice `Program.cs` con il codice seguente.

```
using AWS.Messaging;
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
    // The SQS URL should be passed as a command line argument or set in the Debug
    launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
```

```
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
    Publisher.GreetingMessage message) =>
    {
        return await PostGreeting(message, publisher);
    })
    .WithName("SendGreeting")
    .WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}
```

```
    }  
}  
  
namespace Publisher  
{  
    /// <summary>  
    /// This class represents the message contents.  
    /// </summary>  
    public class GreetingMessage  
    {  
        public string? SenderName { get; set; }  
        public string? Greeting { get; set; }  
    }  
}
```

5. Esegui il comando seguente. Questo dovrebbe aprire una finestra del browser con l'interfaccia utente di Swagger, che ti permetterà di esplorare e testare la tua API.

```
dotnet watch run <queue URL created earlier>
```

6. Apri `/greetingendpoint` e scegli Try it out.
7. `senderName` specificate `greeting` i valori per il messaggio e scegliete Esegui. Questo richiama la tua API, che invia il messaggio SQS.

Crea ed esegui l'applicazione di gestione

Utilizzare la procedura seguente per creare ed eseguire l'applicazione di gestione.

1. Aprire un prompt dei comandi o un terminale. Trovare o creare una cartella del sistema operativo in cui è possibile creare un progetto.NET.
2. In tale cartella, eseguire il seguente comando per creare il progetto.NET.

```
dotnet new console --name Handler
```

3. Naviga nella cartella del nuovo progetto. Aggiungi una dipendenza dal AWS Message Processing Framework per .NET. Aggiungi anche il `Microsoft.Extensions.Hosting` pacchetto, che consente di configurare il framework tramite l'[host generico.NET](#).

```
cd Handler  
dotnet add package AWS.Messaging
```



```
dotnet add package Microsoft.Extensions.Hosting
```

Note

Se lo utilizzi AWS IAM Identity Center per l'autenticazione, assicurati di aggiungere anche `AWSSDK.SSO` e `AWSSDK.SSO0IDC`.

4. Sostituisci il codice `Program.cs` con il codice seguente.

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
        Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);

            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
            GreetingMessage>("greetingMessage");

        }
        // You can add additional message handlers here, using different message
        types.
    });
});
```

```
var host = builder.Build();
await host.RunAsync();

namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from
{messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. Esegui il comando seguente. Questo avvia un sondaggio di lunga durata.

```
dotnet run <queue URL created earlier>
```

Poco dopo l'avvio, l'applicazione riceverà il messaggio inviato nella prima parte di questo tutorial e registrerà il seguente messaggio:

```
Received message {greeting} from {senderName}
```

6. Premi **Ctrl+C** per interrompere il sondaggio.

Rimozione

Utilizzate uno dei seguenti comandi per AWS CLI o per AWS Tools for PowerShell eliminare la coda.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

AWS Tools for PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Pubblica messaggi con il AWS Message Processing Framework per .NET

Questa è la documentazione preliminare per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework for .NET supporta la pubblicazione di uno o più tipi di messaggi, l'elaborazione di uno o più tipi di messaggi o l'esecuzione di entrambe le operazioni nella stessa applicazione.

Il codice seguente mostra una configurazione per un'applicazione che pubblica tipi di messaggi diversi su AWS servizi diversi.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
    EventBridge bus
```

```
builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-  
west-2:012345678910:event-bus/default");  
});
```

Dopo aver registrato il framework durante l'avvio, inserisci il generico `IMessagePublisher` nel tuo codice. Chiama il suo `PublishAsync` metodo per pubblicare uno qualsiasi dei tipi di messaggio configurati sopra. L'editore generico determinerà la destinazione verso cui indirizzare il messaggio in base al tipo.

Nell'esempio seguente, un controller ASP.NET MVC riceve sia `ChatMessage` messaggi che `OrderInfo` eventi dagli utenti, quindi li pubblica rispettivamente su Amazon SQS e Amazon SNS. Entrambi i tipi di messaggi possono essere pubblicati utilizzando l'editore generico configurato in precedenza.

```
[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<ActionResult> PublishChatMessage([FromBody] ChatMessage message)  
    {  
        // Perform business and validation logic on the ChatMessage here.  
        if (message == null)  
        {  
            return BadRequest("A chat message was not submitted. Unable to forward to  
the message queue.");  
        }  
        if (string.IsNullOrEmpty(message.MessageDescription))  
        {  
            return BadRequest("The MessageDescription cannot be null or empty.");  
        }  
  
        // Send the ChatMessage to SQS, using the generic publisher.  
        await _messagePublisher.PublishAsync(message);  
    }  
}
```

```
        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

Per indirizzare un messaggio alla logica di gestione appropriata, il framework utilizza dei metadati denominati identificatori del tipo di messaggio. Per impostazione predefinita, si tratta del nome completo del tipo.NET del messaggio, incluso il nome dell'assembly. Se state inviando e gestendo messaggi, questo meccanismo funziona bene se condividete la definizione degli oggetti del messaggio tra progetti. Tuttavia, se i messaggi vengono ridefiniti in diversi namespace o se state scambiando messaggi con altri framework o linguaggi di programmazione, potrebbe essere necessario sostituire l'identificatore del tipo di messaggio.

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
        // Register that you'll publish messages of type GreetingMessage to an existing
        queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

Editori specifici per servizi

L'esempio mostrato sopra utilizza il formato generico `IMessagePublisher`, che può essere pubblicato su qualsiasi AWS servizio supportato in base al tipo di messaggio configurato. Il framework fornisce anche editori specifici per servizi per Amazon SQS, Amazon SNS e Amazon EventBridge. Questi editori specifici espongono opzioni che si applicano solo a quel servizio e che possono essere inserite utilizzando i tipi, e. `ISQSPublisher`, `ISNSPublisher`, `IEventBridgePublisher`.

[Ad esempio, quando si inviano messaggi a una coda FIFO SQS, è necessario impostare l'ID del gruppo di messaggi appropriato.](#) Il codice seguente mostra nuovamente l'`ChatMessage` esempio, ma ora viene utilizzato un `ISQSPublisher` per impostare opzioni specifiche di SQS.

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
        });
    }
}
```

```
        MessageDeduplicationId = <message-deduplication-id>,
        MessageGroupId = <message-group-id>
    });

    return Ok();
}
}
```

Lo stesso può essere fatto per SNS e EventBridge, rispettivamente, utilizzando `ISNSPublisher` e `IEventBridgePublisher`

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

Per impostazione predefinita, i messaggi di un determinato tipo vengono inviati alla destinazione configurata in anticipo. Tuttavia, puoi sovrascrivere la destinazione di un singolo messaggio utilizzando gli editori specifici del messaggio. Puoi anche sostituire il AWS SDK for .NET client sottostante utilizzato per pubblicare il messaggio, il che può essere utile nelle applicazioni multi-tenant in cui è necessario modificare ruoli o credenziali, a seconda della destinazione.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consuma messaggi con il AWS Message Processing Framework for .NET

Questa è la documentazione preliminare per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework for .NET consente di utilizzare i messaggi che sono stati [pubblicati](#) utilizzando il framework o uno dei servizi di messaggistica. I messaggi possono essere utilizzati in diversi modi, alcuni dei quali sono descritti di seguito.

Gestori di messaggi

Per utilizzare i messaggi, implementa un gestore di messaggi utilizzando l'`IMessageHandler` interfaccia per ogni tipo di messaggio che desideri elaborare. La mappatura tra tipi di messaggi e gestori di messaggi viene configurata all'avvio del progetto.

```
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();
```

Il codice seguente mostra un esempio di gestore di messaggi per un messaggio. `ChatMessage`

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

L'esterno `MessageEnvelope` contiene i metadati utilizzati dal framework. `message` La sua proprietà è il tipo di messaggio (in questo caso `ChatMessage`).

Puoi tornare `MessageProcessStatus.Success()` a indicare che il messaggio è stato elaborato correttamente e il framework eliminerà il messaggio dalla coda di Amazon SQS. Al momento della restituzione `MessageProcessStatus.Failed()`, il messaggio rimarrà in coda dove potrà essere nuovamente elaborato o spostato in una coda di [lettere morte, se configurato](#).

Gestione dei messaggi in un processo di lunga durata

Puoi chiamare `AddSQSPoller` con un URL di coda SQS per avviare un processo di lunga durata [BackgroundService](#) che esegua il polling continuo della coda ed elabori i messaggi.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
```

```
{
    // Register an SQS Queue that the framework will poll for messages.
    // NOTE: The URL given below is an example. Use the appropriate URL for
    your SQS Queue.
    builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
    {
        // The maximum number of messages from this queue that the framework
        will process concurrently on this client.
        options.MaxNumberOfConcurrentMessages = 10;

        // The duration each call to SQS will wait for new messages.
        options.WaitTimeSeconds = 20;
    });

    // Register all IMessageHandler implementations with the message type they
    should process.
    builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
});
})
.Build()
.RunAsync();
```

Configurazione di SQS Message Poller

Il poller dei messaggi SQS può essere configurato da when call. `SQSMessagePollerOptions` `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- Il numero massimo di messaggi dalla coda da elaborare contemporaneamente. Il valore predefinito è 10.
- `WaitTimeSeconds`- La durata (in secondi) per cui la chiamata `ReceiveMessage` SQS attende l'arrivo di un messaggio nella coda prima di tornare. Se un messaggio è disponibile, la chiamata ritorna prima del. `WaitTimeSeconds` Il valore predefinito è 20.

Gestione del timeout di visibilità dei messaggi

I messaggi SQS hanno un periodo di [timeout di visibilità](#). Quando un consumatore inizia a gestire un determinato messaggio, questo rimane in coda ma viene nascosto agli altri consumatori per evitare di elaborarlo più di una volta. Se il messaggio non viene gestito ed eliminato prima di diventare nuovamente visibile, un altro consumatore potrebbe tentare di gestire lo stesso messaggio.

Il framework traccerà e tenterà di estendere il timeout di visibilità per i messaggi che sta attualmente gestendo. È possibile configurare questo comportamento su `SQSMessagePollerOptions` when `callAddSQSPoller`.

- `VisibilityTimeout`- La durata in secondi in cui i messaggi ricevuti vengono nascosti alle successive richieste di recupero. Il valore predefinito è 30.
- `VisibilityTimeoutExtensionThreshold`- Quando il timeout di visibilità di un messaggio rientra in questo numero di secondi dalla scadenza, il framework prolungherà il timeout di visibilità (di altri secondi). `VisibilityTimeout` Il valore predefinito è 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Con quale frequenza, in secondi, il framework verificherà la presenza di messaggi che mancano pochi `VisibilityTimeoutExtensionThreshold` secondi alla scadenza e quindi ne estenderà il timeout di visibilità. Il valore predefinito è 1.

Nell'esempio seguente, il framework controllerà ogni secondo i messaggi che sono ancora in fase di gestione. Per quei messaggi entro 5 secondi dalla loro nuova visibilità, il framework prolungherà automaticamente il timeout di visibilità di ogni messaggio di altri 30 secondi.

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

Gestione dei messaggi nelle funzioni AWS Lambda

Puoi utilizzare il AWS Message Processing Framework per .NET con [l'integrazione di SQS con Lambda](#). Questo è fornito dal pacchetto. `AWS.Messaging.Lambda` Consulta il suo [README](#) per iniziare.

Utilizzo di FIFO con il AWS Message Processing Framework per .NET

Questa è la documentazione preliminare per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

[Per i casi d'uso in cui l'ordine e la deduplicazione dei messaggi sono fondamentali, il AWS Message Processing Framework for .NET supporta le code Amazon SQS first-in-first-out \(FIFO\) e gli argomenti di Amazon SNS.](#)

Pubblicazione

Quando si pubblicano messaggi su una coda o un argomento FIFO, è necessario impostare l'ID del gruppo di messaggi, che specifica il gruppo a cui appartiene il messaggio. I messaggi all'interno di un gruppo vengono elaborati in ordine. Puoi impostarlo sugli editor di messaggi specifici per SQL e SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Sottoscrizione in corso

Quando si gestiscono i messaggi da una coda FIFO, il framework gestisce i messaggi all'interno di un determinato gruppo di messaggi nell'ordine in cui sono stati ricevuti per ogni chiamata. `ReceiveMessages` Il framework entra automaticamente in questa modalità operativa quando è configurato con una coda che termina con `.fifo`

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET.
        services.AddAWSMessageBus(builder =>
        {
            // Because this is a FIFO queue, the framework automatically handles these
            messages in order.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MPF.fifo");
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Registrazione e telemetria aperta per il AWS Message Processing Framework per.NET

Questa è la documentazione preliminare per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework per .NET è OpenTelemetry progettato per registrare [le tracce](#) di ogni messaggio pubblicato o gestito dal framework. Questo è fornito dal pacchetto. [AWS.Messaging.Telemetry.OpenTelemetry](#) Consulta il suo [README](#) per iniziare.

Note

Per informazioni di sicurezza relative alla registrazione, vedere. [Sicurezza per il AWS Message Processing Framework per.NET](#)

Personalizzazione del framework di elaborazione dei AWS messaggi per.NET

Questa è la documentazione non definitiva per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework per .NET crea, invia e gestisce i messaggi in tre diversi «livelli»:

1. Al livello più esterno, il framework crea la richiesta o la risposta AWS-native specifica per un servizio. Con Amazon SQS, ad esempio, crea [SendMessage](#) richieste e lavora con gli [Message](#) oggetti definiti dal servizio.
2. [All'interno della richiesta e della risposta SQS, il framework imposta l'MessageBodyelemento \(o Message per Amazon SNS Detail o EventBridge Amazon\) su un formato JSON. CloudEvent](#) Contiene i metadati impostati dal framework accessibile sull'oggetto durante la gestione di un messaggio. MessageEnvelope
3. Al livello più interno, l' data attributo all'interno dell'oggetto CloudEvent JSON contiene una serializzazione JSON dell'oggetto.NET che è stato inviato o ricevuto come messaggio.

```
{  
  "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>"  
}
```

È possibile personalizzare la modalità di configurazione e lettura della busta del messaggio:

- "id" identifica in modo univoco il messaggio. Per impostazione predefinita è impostato su un nuovo GUID, ma questo può essere sovrascritto implementando il proprio `IMessageIdGenerator` e inserendolo nel contenitore DI.
- "type" controlla come il messaggio viene indirizzato ai gestori. Per impostazione predefinita, utilizza il nome completo del tipo .NET che corrisponde al messaggio. È possibile sovrascriverlo tramite il `messageTypeIdentifier` parametro durante la mappatura del tipo di messaggio alla destinazione tramite `AddSQSPublisher`, `AddSNSPublisher`, o `AddEventBridgePublisher`.
- "source" indica quale sistema o server ha inviato il messaggio.
 - Questo sarà il nome della funzione in caso di pubblicazione da AWS Lambda, il nome del cluster e l'ARN dell'attività se su Amazon ECS, l'ID dell'istanza se su Amazon EC2, altrimenti un valore di fallback di `/aws/messaging`.
 - Puoi sovrascriverlo tramite o su `AddMessageSource` `AddMessageSourceSuffix` `MessageBusBuilder`.
- "time" impostato sulla corrente `DateTime` in UTC. Questo può essere sovrascritto implementando il proprio `IDateTimeHandler` e iniettandolo nel contenitore DI.
- "data" contiene una rappresentazione JSON dell'oggetto .NET che è stato inviato o ricevuto come messaggio:
 - `ConfigureSerializationOptions` su `MessageBusBuilder` consente di configurare [System.Text.Json.JsonSerializerOptions](#) ciò che verrà utilizzato durante la serializzazione e la deserializzazione del messaggio.
 - Per inserire attributi aggiuntivi o trasformare la busta del messaggio una volta creata dal framework, puoi implementarla e registrarla tramite `on. ISerializationCallback` `AddSerializationCallback` `MessageBusBuilder`.

Sicurezza per il AWS Message Processing Framework per.NET

Questa è la documentazione preliminare per una funzionalità della versione di anteprima. ed è soggetta a modifiche.

Il AWS Message Processing Framework per .NET si basa su AWS SDK for .NET per comunicare con. AWS Per ulteriori informazioni sulla sicurezza in AWS SDK for .NET, vedere. [Sicurezza per questo AWS prodotto o servizio](#)

Per motivi di sicurezza, il framework non registra i messaggi di dati inviati dall'utente. Se si desidera abilitare questa funzionalità per scopi di debug, è necessario chiamare `EnableDataMessageLogging()` il Message Bus come segue:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Se scoprite un potenziale problema di sicurezza, fate riferimento alla [politica di sicurezza](#) per la segnalazione delle informazioni.

Programmazione AWS OpsWorks per lavorare con stack e applicazioni

Warning

AWS OpsWorks sta raggiungendo la fine del ciclo di vita e non accetta nuovi clienti. I clienti esistenti non saranno interessati fino a marzo o maggio del 2024, a seconda dei servizi che stanno utilizzando, momento in cui il servizio non sarà più disponibile. Per prepararsi a questa transizione, consigliamo ai clienti esistenti di migrare ad altre soluzioni il prima possibile. Per ulteriori informazioni, consulta la [pagina dei dettagli del prodotto OpsWorks](#).

AWS SDK for .NET supporta AWS OpsWorks, un servizio che offre un modo semplice e flessibile per creare e gestire stack e applicazioni. Con AWS OpsWorks, è possibile effettuare il provisioning AWS delle risorse, gestirne la configurazione, distribuire applicazioni su tali risorse e monitorarne

lo stato. Per ulteriori informazioni, consulta la [pagina del OpsWorks prodotto](#) e la [Guida per l'AWS OpsWorksutente](#).

API

AWS SDK for .NET fornisce API per AWS OpsWorks. [Le API consentono di lavorare con AWS OpsWorks funzionalità come gli stack con i relativi livelli, istanze e app](#). Per visualizzare il set completo di API, consulta l'[AWS SDK for .NET API Reference](#) (e scorri fino a «Amazon. OpsWorks»).

[Le AWS OpsWorks API sono fornite da. AWSSDK OpsWorks](#) NuGet pacchetto.

Prerequisiti

Prima di iniziare, assicuratevi di aver [configurato l'ambiente e il progetto](#). Consulta anche le informazioni contenute in [Funzionalità SDK](#).

Support per altri AWS servizi e configurazione

La AWS SDK for .NET supporta AWS oltre a quelli descritti nelle sezioni precedenti. Per ulteriori informazioni sulle API per tutti i servizi supportati, consulta la documentazione [AWS SDK for .NET Documentazione di riferimento API](#).

Oltre agli spazi dei nomi per singoli AWS servizi, AWS SDK for .NET fornisce le seguenti API:

Area	Descrizione	Risorse
Supporto AWS	Accesso programmatico a AWS Custodie di Support e funzionalità Trusted Advisor.	Consulta Amazon.AWSSupport e Amazon.AWSSupport.Model .
Generali	Classi helper ed enumerazioni.	Consulta Amazon e Amazon.Util .

AWS SDK for .NET esempi di codice

Gli esempi di codice riportati in questo argomento mostrano come utilizzare AWS SDK for .NET with AWS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Esempi cross-service: applicazioni di esempio che funzionano su più servizi Servizi AWS.

Esempi

- [Azioni e scenari utilizzando AWS SDK for .NET](#)
- [Esempi trasversali di servizi che utilizzano AWS SDK for .NET](#)

Azioni e scenari utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Servizi AWS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Servizi

- [Esempi di ACM che utilizzano AWS SDK for .NET](#)
- [Esempi di Aurora che utilizzano AWS SDK for .NET](#)
- [Esempi di Auto Scaling utilizzando AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon Bedrock AWS SDK for .NET](#)
- [Esempi di Amazon Bedrock Runtime utilizzando AWS SDK for .NET](#)

- [AWS CloudFormation esempi utilizzando AWS SDK for .NET](#)
- [CloudWatch esempi che utilizzano AWS SDK for .NET](#)
- [CloudWatch Registra esempi utilizzando AWS SDK for .NET](#)
- [Esempi di Amazon Cognito Identity Provider che utilizzano AWS SDK for .NET](#)
- [Esempi di Amazon Comprehend con AWS SDK for .NET](#)
- [Esempi di utilizzo di DynamoDB AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon EC2 AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon ECS AWS SDK for .NET](#)
- [Elastic Load Balancing - Esempi di utilizzo della versione 2 AWS SDK for .NET](#)
- [EventBridge esempi utilizzando AWS SDK for .NET](#)
- [AWS Glue esempi utilizzando AWS SDK for .NET](#)
- [Esempi IAM che utilizzano AWS SDK for .NET](#)
- [Esempi di Amazon Keyspaces utilizzando AWS SDK for .NET](#)
- [Esempi di Kinesis che utilizzano AWS SDK for .NET](#)
- [AWS KMS esempi utilizzando AWS SDK for .NET](#)
- [Esempi di utilizzo di Lambda AWS SDK for .NET](#)
- [MediaConvert esempi utilizzando AWS SDK for .NET](#)
- [Organizations: esempi che utilizzano AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon Pinpoint AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon Polly AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon RDS AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon Rekognition AWS SDK for .NET](#)
- [Esempi di registrazione del dominio Route 53 utilizzando AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon S3 AWS SDK for .NET](#)
- [Esempi di utilizzo di S3 Glacier AWS SDK for .NET](#)
- [SageMaker esempi che utilizzano AWS SDK for .NET](#)
- [Esempi di Secrets Manager che utilizzano AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon SES AWS SDK for .NET](#)
- [Esempi di API Amazon SES v2 con AWS SDK for .NET](#)
- [Esempi di utilizzo di Amazon SNS AWS SDK for .NET](#)

- [Esempi di utilizzo di Amazon SQS AWS SDK for .NET](#)
- [Esempi di Step Functions utilizzando AWS SDK for .NET](#)
- [AWS STS esempi utilizzando AWS SDK for .NET](#)
- [AWS Support esempi che utilizzano AWS SDK for .NET](#)
- [Esempi di Amazon Transcribe utilizzando AWS SDK for .NET](#)
- [Esempi di Amazon Translate con AWS SDK for .NET](#)

Esempi di ACM che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with ACM.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

DescribeCertificate

Il seguente esempio di codice mostra come utilizzare `DescribeCertificate`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;  
}  
}
```

- Per i dettagli sull'API, [DescribeCertificate](#) consulta AWS SDK for .NET API Reference.

ListCertificates

Il seguente esempio di codice mostra come utilizzare `ListCertificates`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
}
}
```

- Per i dettagli sull'API, [ListCertificates](#) consulta AWS SDK for .NET API Reference.

Esempi di Aurora che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Aurora.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Aurora

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Aurora.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DescribeDBClusters](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateDBCluster

Il seguente esempio di codice mostra come usare `CreateDBCluster`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [CreateDBCluster](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

CreateDBClusterParameterGroup

Il seguente esempio di codice mostra come usare `CreateDBClusterParameterGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [CreateDB ClusterParameterGroup](#) in AWS SDK for .NET API Reference.

CreateDBClusterSnapshot

Il seguente esempio di codice mostra come utilizzare `CreateDBClusterSnapshot`

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [CreateDB ClusterSnapshot](#) in AWS SDK for .NET API Reference.

CreateDBInstance

Il seguente esempio di codice mostra come utilizzare `CreateDBInstance`

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per informazioni dettagliate sull'API, consulta [CreateDBInstance](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DeleteDBCluster

Il seguente esempio di codice mostra come usare `DeleteDBCluster`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta la sezione [DeleteDBCluster](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DeleteDBClusterParameterGroup

Il seguente esempio di codice mostra come usare `DeleteDBClusterParameterGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DeleteDB ClusterParameterGroup](#) in AWS SDK for .NET API Reference.

DeleteDBInstance

Il seguente esempio di codice mostra come utilizzare `DeleteDBInstance`

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DeleteDBInstance](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DescribeDBClusterParameterGroups

Il seguente esempio di codice mostra come usare `DescribeDBClusterParameterGroups`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
/// <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();
}
```

- Per i dettagli sull'API, consulta [DescribeDB ClusterParameterGroups](#) in AWS SDK for .NET API Reference.

DescribeDBClusterParameters

Il seguente esempio di codice mostra come utilizzare `DescribeDBClusterParameters`

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeDB ClusterParameters](#) in AWS SDK for .NET API Reference.

DescribeDBClusterSnapshots

Il seguente esempio di codice mostra come utilizzare. DescribeDBClusterSnapshots

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeDB ClusterSnapshots](#) in AWS SDK for .NET API Reference.

DescribeDBClusters

Il seguente esempio di codice mostra come utilizzare `DescribeDBClusters`

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DescribeDBClusters](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DescribeDBEngineVersions

Il seguente esempio di codice mostra come usare `DescribeDBEngineVersions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeDB EngineVersions](#) in AWS SDK for .NET API Reference.

DescribeDBInstances

Il seguente esempio di codice mostra come utilizzare `DescribeDBInstances`

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DescribeDBInstances](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DescribeOrderableDBInstanceOptions

Il seguente esempio di codice mostra come usare `DescribeOrderableDBInstanceOptions`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeOrderableDB InstanceOptions](#) in AWS SDK for .NET API Reference.

ModifyDBClusterParameterGroup

Il seguente esempio di codice mostra come utilizzare `ModifyDBClusterParameterGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [ModifyDB ClusterParameterGroup](#) in AWS SDK for .NET API Reference.

Scenari

Uso dei cluster di database

L'esempio di codice seguente mostra come:

- Crea un gruppo di parametri del cluster di database Aurora personalizzati e imposta i relativi valori.
- Crea un cluster di database che utilizza il gruppo di parametri.
- Crea un'istanza database che contiene un database.
- Acquisisci uno snapshot del cluster di database, quindi elimina le risorse.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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(
                $"{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);

```

```
    Console.WriteLine($"9. Create a new DB cluster with identifier
{clusterIdentifier}.");

    DBCluster newCluster;
    var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
    var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
clusterIdentifier);

    if (isClusterCreated)
    {
        Console.WriteLine("Cluster already created.");
        newCluster = clusters.First(i => i.DBClusterIdentifier ==
clusterIdentifier);
    }
    else
    {
        Console.WriteLine("Enter an admin username:");
        var username = Console.ReadLine();

        Console.WriteLine("Enter an admin password:");
        var password = Console.ReadLine();

        newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
            "ExampleDatabase",
            clusterIdentifier,
            parameterGroup.DBClusterParameterGroupName,
            engineName,
            engineVersion,
            username!,
            password!
        );

        Console.WriteLine("10. Waiting for DB cluster to be ready...");
        while (newCluster.Status != "available")
        {
            Console.Write(".");
            Thread.Sleep(5000);
            clusters = await
auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
            newCluster = clusters.First();
        }
    }

    Console.WriteLine(sepBar);
```

```
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"{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($"\\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.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.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
Console.WriteLine("Parameter group deleted.");
```

```

    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null &&
        ynResponse.Equals("y",
            StringComparison.InvariantCultureIgnoreCase);
    return response;
}

```

Metodi wrapper che vengono richiamati dallo scenario per gestire le operazioni Aurora.

```

using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>

```



```

    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
    string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
CreateCustomClusterParameterGroupAsync(
    string parameterGroupFamily,
    string groupName,
    string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.

```

```

    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

```

```
/// <summary>
/// Modify the specified integer parameters with new values from user input.
/// </summary>
/// <param name="groupName">The group name for the parameters.</param>
/// <param name="parameters">The list of integer parameters to modify.</param>
/// <param name="newValue">Optional int value to set for parameters.</param>
/// <returns>The name of the group that was modified.</returns>
public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
{
    foreach (var p in parameters)
    {
        if (p.IsModifiable && 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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateDBCluster](#)
 - [Creato B ClusterParameterGroup](#)
 - [Creato DB ClusterSnapshot](#)
 - [CreateDBInstance](#)
 - [DeleteDBCluster](#)
 - [Eliminare DB ClusterParameterGroup](#)
 - [DeleteDBInstance](#)
 - [Descritto B ClusterParameterGroups](#)
 - [Descritto B ClusterParameters](#)
 - [Descritto B ClusterSnapshots](#)
 - [DescribeDBClusters](#)
 - [Descritto B EngineVersions](#)

- [DescribeDBInstances](#)
- [DescribeOrderableDB InstanceOptions](#)
- [Modifica DB ClusterParameterGroup](#)

Esempi di Auto Scaling utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Auto Scaling.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Auto Scaling

I seguenti esempi di codice mostrano come iniziare a usare Auto Scaling.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.Availability
    });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
groups.");
        }
    }
}
```

- Per i dettagli sull'API, [DescribeAutoScalingGroups](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

AttachLoadBalancerTargetGroups

Il seguente esempio di codice mostra come utilizzare `AttachLoadBalancerTargetGroups`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
/// <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 }
        });
}
```

- Per i dettagli sull'API, [AttachLoadBalancerTargetGroups](#) consulta AWS SDK for .NET API Reference.

CreateAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `CreateAutoScalingGroup`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateAutoScalingGroup](#) consulta AWS SDK for .NET API Reference.

DeleteAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `DeleteAutoScalingGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Aggiorna la dimensione minima di un gruppo con dimensionamento automatico a zero, termina tutte le istanze del gruppo ed elimina il gruppo.

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

- Per i dettagli sull'API, [DeleteAutoScalingGroup](#) consulta AWS SDK for .NET API Reference.

DescribeAutoScalingGroups

Il seguente esempio di codice mostra come utilizzare `DescribeAutoScalingGroups`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAutoScalingGroups](#) consulta AWS SDK for .NET API Reference.

DescribeAutoScalingInstances

Il seguente esempio di codice mostra come utilizzare `DescribeAutoScalingInstances`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAutoScalingInstances](#) consulta AWS SDK for .NET API Reference.

DescribeScalingActivities

Il seguente esempio di codice mostra come utilizzare `DescribeScalingActivities`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [DescribeScalingActivities](#) consulta AWS SDK for .NET API Reference.

DisableMetricsCollection

Il seguente esempio di codice mostra come utilizzare `DisableMetricsCollection`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DisableMetricsCollection](#) consulta AWS SDK for .NET API Reference.

EnableMetricsCollection

Il seguente esempio di codice mostra come utilizzare `EnableMetricsCollection`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [EnableMetricsCollection](#) consulta AWS SDK for .NET API Reference.

SetDesiredCapacity

Il seguente esempio di codice mostra come utilizzare `SetDesiredCapacity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [SetDesiredCapacity](#) consulta AWS SDK for .NET API Reference.

TerminateInstanceInAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `TerminateInstanceInAutoScalingGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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


- Per i dettagli sull'API, [TerminateInstanceInAutoScalingGroup](#) consulta AWS SDK for .NET API Reference.

UpdateAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `UpdateAutoScalingGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateAutoScalingGroup](#) consulta AWS SDK for .NET API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisce le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui lo scenario interattivo al prompt dei comandi.

```
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($"http://{endPoint}\n");
}
else
{
    Console.WriteLine(
        "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
        + "manually verifying that your VPC and security group are
configured correctly and that\n"
        + "you can successfully make a GET request to the load balancer
endpoint:\n");
    Console.WriteLine($"http://{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($"\\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;
}
```

Crea una classe che racchiuda le operazioni di dimensionamento automatico e Amazon EC2.

```
/// <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="deleteKeyName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyName });
        File.Delete($"{deleteKeyName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyName} 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 (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 }
        });
}
}
}

```

Crea una classe che racchiuda le operazioni di Elastic Load Balancing.

```

/// <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 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);
        }
    }
}
}

```

Crea una classe che utilizzi DynamoDB per simulare un servizio di raccomandazione.

```

/// <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");
    }
}
}

```

Crea una classe che racchiuda le operazioni di Systems Manager.

```

/// <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 });
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)

- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacesIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Gestisci gruppi e istanze

L'esempio di codice seguente mostra come:

- Crea un gruppo Amazon EC2 Auto Scaling con un modello di lancio e zone di disponibilità e ottieni informazioni sulle istanze in esecuzione.
- Abilita la raccolta di CloudWatch metriche Amazon.
- Aggiorna la capacità desiderata del gruppo e attendi l'avvio di un'istanza.
- Termina un'istanza nel gruppo.
- Elenca le attività di scalabilità che si verificano in risposta alle richieste degli utenti e ai cambiamenti di capacità.
- Ottieni statistiche per le CloudWatch metriche, quindi ripulisci le risorse.

AWS SDK for .NET

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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($"Metrics collected for {groupName}:");
    metrics.ForEach(metric =>
    {
        Console.WriteLine($"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.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();
}
}
```

Definisci le funzioni richiamate dallo scenario per gestire i modelli e le metriche di lancio. Queste funzioni comprendono Auto Scaling, Amazon EC2 e azioni. CloudWatch

```
namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
```

```
private readonly IAmazonAutoScaling _amazonAutoScaling;

/// <summary>
/// Constructor for the AutoScalingWrapper class.
/// </summary>
/// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
client.</param>
public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
{
    _amazonAutoScaling = amazonAutoScaling;
}

/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
    {
        availabilityZone,
    };

    var request = new CreateAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        AvailabilityZones = zoneList,
        LaunchTemplate = templateSpecification,
        MaxSize = 6,
        MinSize = 1
    };
};
```

```
        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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAutoScalingInstances](#)
 - [DescribeScalingActivities](#)
 - [DisableMetricsCollection](#)
 - [EnableMetricsCollection](#)
 - [SetDesiredCapacity](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Esempi di utilizzo di Amazon Bedrock AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Bedrock. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Amazon Bedrock

I seguenti esempi di codice mostrano come iniziare a usare Amazon Bedrock.

AWS SDK for .NET

 Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListFoundationModels](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)

Azioni

ListFoundationModels

Il seguente esempio di codice mostra come utilizzare `ListFoundationModels`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca i modelli di fondazione Bedrock disponibili.

```
/// <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);
    }
}
```

- Per i dettagli sulle API, consulta la sezione [ListFoundationModels AWS SDK for .NET API Reference](#).

Esempi di Amazon Bedrock Runtime utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Amazon Bedrock Runtime.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [AI21 Labs Jurassic-2](#)
- [Testo Amazon Titan](#)

- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta Llama](#)
- [IA Mistral](#)
- [Scenari](#)

AI21 Labs Jurassic-2

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a AI21 Labs Jurassic-2, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a AI21 Labs Jurassic-2, utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to AI21 Labs Jurassic-2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";
```



```
// Create a request with the model ID, the user message, and an inference
configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- [Per i dettagli sulle API, consulta Converse in API Reference.AWS SDK for .NET](#)

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo a AI21 Labs Jurassic-2, utilizzando l'API Invoke Model.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to AI21 Labs Jurassic-2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = userMessage,
    maxTokens = 512,
    temperature = 0.5
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["completions"]?[0]?["data"]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

Testo Amazon Titan

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

 Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo ad Amazon Titan Text utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to Amazon Titan Text.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,  
        Temperature = 0.5F,  
        TopP = 0.9F  
    }  
};  
  
try  
{  
    // Send the request to the Bedrock Runtime and wait for the result.  
    var response = await client.ConverseAsync(request);  
  
    // Extract and print the response text.  
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";  
    Console.WriteLine(responseText);  
}  
catch (AmazonBedrockRuntimeException e)  
{  
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");  
    throw;  
}
```

- Per i dettagli sull'API, consulta [Converse](#) in API Reference.AWS SDK for .NET

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

AWS SDK for .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo ad Amazon Titan Text utilizzando l'API Converse di Bedrock ed elabora il flusso di risposta in tempo reale.

```
// Use the Converse API to send a text message to Amazon Titan Text
```

```
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);
```

```
// Extract and print the streamed response text in real-time.
foreach (var chunk in response.Stream.AsEnumerable())
{
    if (chunk is ContentBlockDeltaEvent)
    {
        Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta API [ConverseStream](#) Reference AWS SDK for .NET .

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Amazon Titan Text utilizzando l'API Invoke Model.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Amazon Titan Text.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
```

```
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["results"]?[0]?["outputText"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
```



```
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Amazon Titan Text, utilizzando l'API Invoke Model, e stampare il flusso di risposta.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Amazon Titan Text
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
        var text = chunk["outputText"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModelWithResponseStream AWS SDK for .NET API Reference](#).

Anthropic Claude

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

Note

C'è di più su [GitHub](#). Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Anthropic Claude, utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to Anthropic Claude.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
configuration.
```

```
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sulle API, consulta [Converse](#) in API Reference.AWS SDK for .NET

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ad Anthropic Claude, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

AWS SDK for .NET

 Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Anthropic Claude utilizzando l'API Converse di Bedrock ed elabora il flusso di risposta in tempo reale.

```
// Use the Converse API to send a text message to Anthropic Claude
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
    }
}
```

```
    }
  },
  InferenceConfig = new InferenceConfiguration()
  {
    MaxTokens = 512,
    Temperature = 0.5F,
    TopP = 0.9F
  }
};

try
{
  // Send the request to the Bedrock Runtime and wait for the result.
  var response = await client.ConverseStreamAsync(request);

  // Extract and print the streamed response text in real-time.
  foreach (var chunk in response.Stream.AsEnumerable())
  {
    if (chunk is ContentBlockDeltaEvent)
    {
      Console.WriteLine((chunk as ContentBlockDeltaEvent).Delta.Text);
    }
  }
}
catch (AmazonBedrockRuntimeException e)
{
  Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
  throw;
}
```

- Per i dettagli sull'API, consulta [ConverseStream](#) API Reference. AWS SDK for .NET

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Invoke Model.

AWS SDK for .NET

 Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Anthropic Claude.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
```

```
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Anthropic Claude, utilizzando l'API Invoke Model, e stampare il flusso di risposta.

AWS SDK for .NET

Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Anthropic Claude
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};
```

```
try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
        var text = chunk["delta"]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModelWithResponseStream AWS SDK for .NET](#) API Reference.

Cohere Command

Converse: tutti i modelli

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Cohere Command, utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to Cohere Command.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
```

```
var response = await client.ConverseAsync(request);

// Extract and print the response text.
string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sulle API, consulta [Converse](#) in API Reference.AWS SDK for .NET

ConverseStream: Tutti i modelli

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

AWS SDK for .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Cohere Command, utilizzando l'API Converse di Bedrock ed elabora il flusso di risposta in tempo reale.

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
```

```
}  
catch (AmazonBedrockRuntimeException e)  
{  
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");  
    throw;  
}
```

- Per i dettagli sull'API, consulta API [ConverseStream](#) Reference AWS SDK for .NET .

InvokeModel: Comando R e R+

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command R e R+, utilizzando l'API Invoke Model.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Cohere Command R.  
  
using Amazon;  
using Amazon.BedrockRuntime;  
using Amazon.BedrockRuntime.Model;  
using System;  
using System.IO;  
using System.Text.Json;  
using System.Text.Json.Nodes;  
  
// Create a Bedrock Runtime client in the AWS Region you want to use.  
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);  
  
// Set the model ID, e.g., Command R.  
var modelId = "cohere.command-r-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModel: Comando e luce di comando

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando l'API Invoke Model.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Cohere Command.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command Light.
var modelId = "cohere.command-light-text-v14";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});
```



```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);


    // Extract and print the response text.
    var responseText = modelResponse["generations"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream: Comando R e R+

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando l'API Invoke Model con un flusso di risposta.

AWS SDK for .NET

 Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Cohere Command R
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
```

```
var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

// Extract and print the streamed response text in real-time.
foreach (var item in streamingResponse.Body)
{
    var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
    var text = chunk["text"] ?? "";
    Console.Write(text);
}
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream: Comando e luce di comando

Il seguente esempio di codice mostra come inviare un messaggio di testo a Cohere Command, utilizzando l'API Invoke Model con un flusso di risposta.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Cohere Command
// and print the response stream.

using Amazon;
```

```
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command Light.
var modelId = "cohere.command-light-text-v14";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
        var text = chunk["generations"]?[0]?["text"] ?? "";
    }
}
```

```
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

Meta Llama

Tutti i modelli: Converse API

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Meta Llama utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to Meta Llama.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
```

```
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sulle API, consulta [Converse](#) in API Reference.AWS SDK for .NET

ConverseStream: Tutti i modelli

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Meta Llama utilizzando l'API Converse di Bedrock ed elabora il flusso di risposta in tempo reale.

```
// Use the Converse API to send a text message to Meta Llama
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
```

```
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.WriteLine((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta API [ConverseStream](#) Reference AWS SDK for .NET .

InvokeModel: Llama 2

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 2, utilizzando l'API Invoke Model.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Meta Llama 2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
```

```
        temperature = 0.5
    });

    // Create a request with the model ID and the model's native request payload.
    var request = new InvokeModelRequest()
    {
        ModelId = modelId,
        Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
        ContentType = "application/json"
    };

    try
    {
        // Send the request to the Bedrock Runtime and wait for the response.
        var response = await client.InvokeModelAsync(request);

        // Decode the response body.
        var modelResponse = await JsonNode.ParseAsync(response.Body);

        // Extract and print the response text.
        var responseText = modelResponse["generation"] ?? "";
        Console.WriteLine(responseText);
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
        throw;
    }
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModel: Llama 3

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 3, utilizzando l'API Invoke Model.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Meta Llama 3.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
```

```
        temperature = 0.5
    });

    // Create a request with the model ID and the model's native request payload.
    var request = new InvokeModelRequest()
    {
        ModelId = modelId,
        Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
        ContentType = "application/json"
    };

    try
    {
        // Send the request to the Bedrock Runtime and wait for the response.
        var response = await client.InvokeModelAsync(request);

        // Decode the response body.
        var modelResponse = await JsonNode.ParseAsync(response.Body);


        // Extract and print the response text.
        var responseText = modelResponse["generation"] ?? "";
        Console.WriteLine(responseText);
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
        throw;
    }
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream: Llama 2

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 2, utilizzando l'API Invoke Model, e stampare il flusso di risposta.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Meta Llama 2
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);


    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModelWithResponseStream AWS SDK for .NET](#) API Reference.

InvokeModelWithResponseStream: Llama 3

Il seguente esempio di codice mostra come inviare un messaggio di testo a Meta Llama 3, utilizzando l'API Invoke Model, e stampare il flusso di risposta.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Meta Llama 3
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
```

```
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModelWithResponseStream AWS SDK for .NET API Reference](#).

IA Mistral

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Mistral, utilizzando l'API Converse di Bedrock.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Mistral utilizzando l'API Converse di Bedrock.

```
// Use the Converse API to send a text message to Mistral.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
```

```
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sulle API, consulta [Converse](#) in API Reference.AWS SDK for .NET

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Mistral, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Mistral utilizzando l'API Converse di Bedrock ed elabora il flusso di risposta in tempo reale.

```
// Use the Converse API to send a text message to Mistral
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using Amazon.Runtime;
using System;
using System.Collections.Generic;
using System.Linq;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
```

```
        Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
    }
},
InferenceConfig = new InferenceConfiguration()
{
    MaxTokens = 512,
    Temperature = 0.5F,
    TopP = 0.9F
}
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.WriteLine((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta [ConverseStream](#) API Reference. AWS SDK for .NET

InvokeModel

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Mistral, utilizzando l'API Invoke Model.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Usa l'API Invoke Model per inviare un messaggio di testo.

```
// Use the native inference API to send a text message to Mistral.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
```

```
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["outputs"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModel AWS SDK for .NET API Reference](#).

InvokeModelWithResponseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo ai modelli Mistral AI, utilizzando l'API Invoke Model, e stampare il flusso di risposta.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
// Use the native inference API to send a text message to Mistral
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
```

```
var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

// Extract and print the streamed response text in real-time.
foreach (var item in streamingResponse.Body)
{
    var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
    var text = chunk["outputs"]?[0]?["text"] ?? "";
    Console.Write(text);
}
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Per i dettagli sull'API, consulta la sezione [InvokeModelWithResponseStream AWS SDK for .NET API Reference](#).

Scenari

Crea un'applicazione playground per interagire con i modelli Amazon Bedrock Foundation

Il seguente esempio di codice mostra come creare parchi giochi per interagire con i modelli di base di Amazon Bedrock attraverso diverse modalità.

AWS SDK for .NET

.NET Foundation Model (FM) Playground è un'applicazione di esempio .NET MAUI Blazor che mostra come usare Amazon Bedrock dal codice C#. Questo esempio mostra come gli sviluppatori .NET e C# possono utilizzare Amazon Bedrock per creare applicazioni generative abilitate all'intelligenza artificiale. Puoi testare e interagire con i modelli Amazon Bedrock Foundation utilizzando i seguenti quattro campi da gioco:

- Un parco giochi testuale.
- Un parco giochi per le chat.
- Un parco giochi per le chat vocali.

- Un parco giochi di immagini.

L'esempio elenca e visualizza anche i modelli di base a cui avete accesso e le relative caratteristiche. Per il codice sorgente e le istruzioni di distribuzione, consultate il progetto in [GitHub](#).

Servizi utilizzati in questo esempio

- Runtime di Amazon Bedrock

AWS CloudFormation esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with AWS CloudFormation.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve AWS CloudFormation

Il seguente esempio di codice mostra come iniziare a utilizzare AWS CloudFormation.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [DescribeStackResources](#) consulta AWS SDK for .NET API Reference.

CloudWatch esempi che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with CloudWatch.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve CloudWatch

I seguenti esempi di codice mostrano come iniziare a utilizzare CloudWatch.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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($"\\tMetric: {metric.MetricName}");
            Console.WriteLine($"\\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\\tDimensions: {string.Join(", ",
metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}

```

- Per i dettagli sull'API, [ListMetrics](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

DeleteAlarms

Il seguente esempio di codice mostra come utilizzare `DeleteAlarms`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }
}
```

- Per i dettagli sull'API, [DeleteAlarms](#) consulta AWS SDK for .NET API Reference.

DeleteAnomalyDetector

Il seguente esempio di codice mostra come utilizzare `DeleteAnomalyDetector`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteAnomalyDetector](#) consulta AWS SDK for .NET API Reference.

DeleteDashboards

Il seguente esempio di codice mostra come utilizzare `DeleteDashboards`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteDashboards](#) consulta AWS SDK for .NET API Reference.

DescribeAlarmHistory

Il seguente esempio di codice mostra come utilizzare `DescribeAlarmHistory`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAlarmHistory](#) consulta AWS SDK for .NET API Reference.

DescribeAlarms

Il seguente esempio di codice mostra come utilizzare `DescribeAlarms`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAlarms](#) consulta AWS SDK for .NET API Reference.

DescribeAlarmsForMetric

Il seguente esempio di codice mostra come utilizzare `DescribeAlarmsForMetric`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAlarmsForMetric](#) consulta AWS SDK for .NET API Reference.

DescribeAnomalyDetectors

Il seguente esempio di codice mostra come utilizzare `DescribeAnomalyDetectors`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeAnomalyDetectors](#) consulta AWS SDK for .NET API Reference.

DisableAlarmActions

Il seguente esempio di codice mostra come utilizzare `DisableAlarmActions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DisableAlarmActions](#) consulta AWS SDK for .NET API Reference.

EnableAlarmActions

Il seguente esempio di codice mostra come utilizzare `EnableAlarmActions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [EnableAlarmActions](#) consulta AWS SDK for .NET API Reference.

GetDashboard

Il seguente esempio di codice mostra come utilizzare `GetDashboard`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetDashboard](#) consulta AWS SDK for .NET API Reference.

GetMetricData

Il seguente esempio di codice mostra come utilizzare `GetMetricData`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [GetMetricData](#) consulta AWS SDK for .NET API Reference.

GetMetricStatistics

Il seguente esempio di codice mostra come utilizzare `GetMetricStatistics`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }

```

- Per i dettagli sull'API, [GetMetricStatistics](#) consulta AWS SDK for .NET API Reference.

GetMetricWidgetImage

Il seguente esempio di codice mostra come utilizzare `GetMetricWidgetImage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetMetricWidgetImage](#) consulta AWS SDK for .NET API Reference.

ListDashboards

Il seguente esempio di codice mostra come utilizzare `ListDashboards`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [ListDashboards](#) consulta AWS SDK for .NET API Reference.

ListMetrics

Il seguente esempio di codice mostra come utilizzare `ListMetrics`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [ListMetrics](#) consulta AWS SDK for .NET API Reference.

PutAnomalyDetector

Il seguente esempio di codice mostra come utilizzare `PutAnomalyDetector`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }
```

- Per i dettagli sull'API, [PutAnomalyDetector](#) consulta AWS SDK for .NET API Reference.

PutDashboard

Il seguente esempio di codice mostra come utilizzare `PutDashboard`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [PutDashboard](#) consulta AWS SDK for .NET API Reference.

PutMetricAlarm

Il seguente esempio di codice mostra come utilizzare `PutMetricAlarm`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }

```

- Per i dettagli sull'API, [PutMetricAlarm](#) consulta AWS SDK for .NET API Reference.

PutMetricData

Il seguente esempio di codice mostra come utilizzare `PutMetricData`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, [PutMetricData](#) consulta AWS SDK for .NET API Reference.

Scenari

Inizia con parametri, pannelli di controllo e allarmi

L'esempio di codice seguente mostra come:

- Elenca i CloudWatch namespace e le metriche.
- Ottieni le statistiche per un parametro e per la fatturazione stimata.
- Crea e aggiorna un pannello di controllo.

- Crea e aggiungi i dati a un parametro.
- Crea e attiva un allarme, quindi visualizza la cronologia degli allarmi.
- Aggiungi un rilevatore di anomalie.
- Acquisisci uno schema di parametri, quindi elimina le risorse.

AWS SDK for .NET

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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($"{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 + 1} State: {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($"\\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
    }

    Console.WriteLine(new string('-', 80));
}
```

```
/// <summary>
/// Fetch and open a metrics image for a CloudWatch metric and namespace.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetAndOpenMetricImage()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("17. Get a metric image from CloudWatch.");

    Console.WriteLine($"\\tGetting Image data for custom metric.");
    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
    var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

    ProcessStartInfo info = new ProcessStartInfo();

    Console.WriteLine($"\\tFile saved as {Path.GetFileName(file)}.");
    Console.WriteLine($"\\tPress enter to open the image.");
    Console.ReadLine();
    info.FileName = Path.Combine("ms-photos://", file);
    info.UseShellExecute = true;
    info.CreateNoWindow = true;
    info.Verb = string.Empty;

    Process.Start(info);

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up created resources.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <param name="metric">The CloudWatch metric.</param>
/// <returns>Async task.</returns>
private static async Task CleanupResources()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"18. Clean up resources.");
}
```

```
var dashboardName = _configuration["dashboardName"];
if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
{
    Console.WriteLine($"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;
}
}
```

Metodi wrapper utilizzati dallo scenario per CloudWatch le azioni.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [DeleteAlarms](#)

- [DeleteAnomalyDetector](#)
- [DeleteDashboards](#)
- [DescribeAlarmHistory](#)
- [DescribeAlarms](#)
- [DescribeAlarmsForMetric](#)
- [DescribeAnomalyDetectors](#)
- [GetMetricData](#)
- [GetMetricStatistics](#)
- [GetMetricWidgetImage](#)
- [ListMetrics](#)
- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)
- [PutMetricData](#)

CloudWatch Registra esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with CloudWatch Logs.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

AssociateKmsKey

Il seguente esempio di codice mostra come utilizzare `AssociateKmsKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AssociateKmsKey](#) consulta AWS SDK for .NET API Reference.

CancelExportTask

Il seguente esempio di codice mostra come utilizzare `CancelExportTask`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CancelExportTask](#) consulta AWS SDK for .NET API Reference.

CreateExportTask

Il seguente esempio di codice mostra come utilizzare `CreateExportTask`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [CreateExportTask](#) consulta AWS SDK for .NET API Reference.

CreateLogGroup

Il seguente esempio di codice mostra come utilizzare `CreateLogGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateLogGroup](#) consulta AWS SDK for .NET API Reference.

CreateLogStream

Il seguente esempio di codice mostra come utilizzare `CreateLogStream`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateLogStream](#) consulta AWS SDK for .NET API Reference.

DeleteLogGroup

Il seguente esempio di codice mostra come utilizzare `DeleteLogGroup`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteLogGroup](#) consulta AWS SDK for .NET API Reference.

DescribeExportTasks

Il seguente esempio di codice mostra come utilizzare `DescribeExportTasks`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
}
```

- Per i dettagli sull'API, [DescribeExportTasks](#) consulta AWS SDK for .NET API Reference.

DescribeLogGroups

Il seguente esempio di codice mostra come utilizzare `DescribeLogGroups`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

- Per i dettagli sull'API, [DescribeLogGroups](#) consulta AWS SDK for .NET API Reference.

StartLiveTail

Il seguente esempio di codice mostra come utilizzare `StartLiveTail`.

AWS SDK for .NET

Includere i file richiesti.

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Avvia la sessione Live Tail.

```
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;
}
```

Puoi gestire gli eventi della sessione di Live Tail in due modi:

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

- Per i dettagli sulle API, consulta la [StartLiveTail](#) sezione AWS SDK for .NET API Reference.

Esempi di Amazon Cognito Identity Provider che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Cognito Identity Provider. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

AdminGetUser

Il seguente esempio di codice mostra come utilizzare `AdminGetUser`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AdminGetUser](#) consulta AWS SDK for .NET API Reference.

AdminInitiateAuth

Il seguente esempio di codice mostra come utilizzare `AdminInitiateAuth`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [AdminInitiateAuth](#) consulta AWS SDK for .NET API Reference.

AdminRespondToAuthChallenge

Il seguente esempio di codice mostra come utilizzare `AdminRespondToAuthChallenge`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [AdminRespondToAuthChallenge](#) consulta AWS SDK for .NET API Reference.

AssociateSoftwareToken

Il seguente esempio di codice mostra come utilizzare `AssociateSoftwareToken`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [AssociateSoftwareToken](#) consulta AWS SDK for .NET API Reference.

ConfirmDevice

Il seguente esempio di codice mostra come utilizzare `ConfirmDevice`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;  
}
```

- Per i dettagli sull'API, [ConfirmDevice](#) consulta AWS SDK for .NET API Reference.

ConfirmSignUp

Il seguente esempio di codice mostra come utilizzare `ConfirmSignUp`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ConfirmSignUp](#) consulta AWS SDK for .NET API Reference.

InitiateAuth

Il seguente esempio di codice mostra come utilizzare `InitiateAuth`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [InitiateAuth](#) consulta AWS SDK for .NET API Reference.

ListUserPools

Il seguente esempio di codice mostra come utilizzare `ListUserPools`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListUserPools](#) consulta AWS SDK for .NET API Reference.

ListUsers

Il seguente esempio di codice mostra come utilizzare `ListUsers`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListUsers](#) consulta AWS SDK for .NET API Reference.

ResendConfirmationCode

Il seguente esempio di codice mostra come utilizzare `ResendConfirmationCode`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ResendConfirmationCode](#) consulta AWS SDK for .NET API Reference.

SignUp

Il seguente esempio di codice mostra come utilizzare `SignUp`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [SignUp](#) consulta AWS SDK for .NET API Reference.

VerifySoftwareToken

Il seguente esempio di codice mostra come utilizzare `VerifySoftwareToken`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [VerifySoftwareToken](#) consulta AWS SDK for .NET API Reference.

Scenari

Registrazione di un utente a un pool di utenti che richiede l'autenticazione MFA

L'esempio di codice seguente mostra come:

- Registra e conferma un utente con nome utente, password e indirizzo e-mail.
- Configura l'autenticazione a più fattori associando un'applicazione MFA all'utente.
- Accedi utilizzando una password e un codice MFA.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.Write("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($"Confirmation code sent to {userName}.");
Console.Write("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.WriteLine("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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)

- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Esempi di Amazon Comprehend con AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Comprehend. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

DetectDominantLanguage

Il seguente esempio di codice mostra come utilizzare `DetectDominantLanguage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");
    }
}
```

- Per i dettagli sull'API, [DetectDominantLanguage](#) consulta AWS SDK for .NET API Reference.

DetectEntities

Il seguente esempio di codice mostra come utilizzare `DetectEntities`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");
    }
}
```

- Per i dettagli sull'API, [DetectEntities](#) consulta AWS SDK for .NET API Reference.

DetectKeyPhrases

Il seguente esempio di codice mostra come utilizzare `DetectKeyPhrases`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");
    }
}
```

- Per i dettagli sull'API, [DetectKeyPhrases](#) consulta AWS SDK for .NET API Reference.

DetectPiiEntities

Il seguente esempio di codice mostra come utilizzare `DetectPiiEntities`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DetectPiiEntities](#) consulta AWS SDK for .NET API Reference.

DetectSentiment

Il seguente esempio di codice mostra come utilizzare `DetectSentiment`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");  
    }  
}
```

- Per i dettagli sull'API, [DetectSentiment](#) consulta AWS SDK for .NET API Reference.

DetectSyntax

Il seguente esempio di codice mostra come utilizzare `DetectSyntax`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");
}
}

```

- Per i dettagli sull'API, [DetectSyntax](#) consulta AWS SDK for .NET API Reference.

StartTopicsDetectionJob

Il seguente esempio di codice mostra come utilizzare `StartTopicsDetectionJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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 method 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}");
    }
}

```

- Per i dettagli sull'API, [StartTopicsDetectionJob](#) consulta AWS SDK for .NET API Reference.

Esempi di utilizzo di DynamoDB AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con DynamoDB.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello DynamoDB

Gli esempi di codice seguenti mostrano come iniziare a utilizzare DynamoDB.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListTables](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Azioni

BatchExecuteStatement

Il seguente esempio di codice mostra come utilizzare `BatchExecuteStatement`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizzo di batch di istruzioni INSERT per aggiungere elementi.

```
/// <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!;
    }
}

```

Utilizzo di batch di istruzioni SELECT per ottenere elementi.

```

/// <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;
    }
}
```

Utilizzo di batch di istruzioni UPDATE per aggiornare elementi.

```

    /// <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;
}

```

Utilizzo di batch di istruzioni DELETE per eliminare elementi.

```

/// <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;
    }

```

- Per i dettagli sull'API, [BatchExecuteStatement](#) consulta AWS SDK for .NET API Reference.

BatchGetItem

Il seguente esempio di codice mostra come utilizzare `BatchGetItem`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
}
}

```

- Per i dettagli sull'API, [BatchGetItem](#) consulta AWS SDK for .NET API Reference.

BatchWriteItem

Il seguente esempio di codice mostra come utilizzare `BatchWriteItem`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Scrive un batch di elementi nella tabella del filmato.

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

- Per i dettagli sull'API, [BatchWriteItem](#) consulta AWS SDK for .NET API Reference.

CreateTable

Il seguente esempio di codice mostra come utilizzare `CreateTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
    /// <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.Write(".");
        }
        while (status != "ACTIVE");

        return status == TableStatus.ACTIVE;
    }
}
```

- Per i dettagli sull'API, [CreateTable](#) consulta AWS SDK for .NET API Reference.

DeleteItem

Il seguente esempio di codice mostra come utilizzare `DeleteItem`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteItem](#) consulta AWS SDK for .NET API Reference.

DeleteTable

Il seguente esempio di codice mostra come utilizzare `DeleteTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [DeleteTable](#) consulta AWS SDK for .NET API Reference.

DescribeTable

Il seguente esempio di codice mostra come utilizzare `DescribeTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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}");
}
```

- Per i dettagli sull'API, [DescribeTable](#) consulta AWS SDK for .NET API Reference.

ExecuteStatement

Il seguente esempio di codice mostra come utilizzare `ExecuteStatement`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Utilizzo di un'istruzione INSERT per aggiungere un elemento.

```
/// <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;
    }

```

Utilizzo di un'istruzione SELECT per ottenere un elemento.

```

/// <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;
}

```

Utilizzo di un'istruzione SELECT per ottenere un elenco di elementi.

```

/// <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;
}

```

Utilizzo di un'istruzione UPDATE per aggiornare un elemento.

```

    /// <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;
    }

```

Utilizzo di un'istruzione DELETE per eliminare un singolo filmato.

```

    /// <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;
    }

```

- Per i dettagli sull'API, [ExecuteStatement](#) consulta AWS SDK for .NET API Reference.

GetItem

Il seguente esempio di codice mostra come utilizzare `GetItem`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    /// <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;
    }
}
```

- Per i dettagli sull'API, [GetItem](#) consulta AWS SDK for .NET API Reference.

ListTables

Il seguente esempio di codice mostra come utilizzare `ListTables`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

- Per i dettagli sull'API, [ListTables](#) consulta AWS SDK for .NET API Reference.

PutItem

Il seguente esempio di codice mostra come utilizzare `PutItem`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [PutItem](#) consulta AWS SDK for .NET API Reference.

Query

Il seguente esempio di codice mostra come utilizzare `Query`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per ulteriori informazioni sulle API, consulta [Query](#) nella Documentazione di riferimento delle API AWS SDK for .NET .

Scan

Il seguente esempio di codice mostra come usare Scan.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
}
```

- Per informazioni dettagliate sulle API, consulta [Scan](#) nella Documentazione di riferimento per le API AWS SDK for .NET .

UpdateItem

Il seguente esempio di codice mostra come usare `UpdateItem`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateItem](#) consulta AWS SDK for .NET API Reference.

Scenari

Nozioni di base sull'utilizzo di tabelle, elementi e query

L'esempio di codice seguente mostra come:

- Crea una tabella in grado di contenere i dati del filmato.
- Inserisci, ottieni e aggiorna un singolo filmato nella tabella.
- Scrivi i dati del filmato nella tabella da un file JSON di esempio.
- Esegui una query sui filmati che sono stati rilasciati in un dato anno.
- Cerca i filmati che sono stati distribuiti in diversi anni.
- Elimina un filmato dalla tabella, quindi elimina la tabella.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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($"
Table: {tableName} successfully created.");
        }
        else
        {
            Console.WriteLine($"
Could 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();
}
}
```

Crea una tabella per contenere i dati del filmato.

```
    /// <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;
}

```

Aggiunge un singolo filmato alla tabella.

```

/// <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;
    }

```

Aggiorna un singolo elemento in una tabella.

```

    /// <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;
}
```

Recupera un singolo elemento dalla tabella del filmato.

```
/// <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;
    }

```

Scrivi un batch di elementi nella tabella del filmato.

```

    /// <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;
}
```

```
}
```

Elimina un singolo elemento dalla tabella.

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

Esegue una query sulla tabella per i filmati usciti in un determinato anno.


```
/// <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;
}

```

Esegue la ricerca dei filmati che sono usciti in un intervallo di anni.

```

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;
}
```

Elimina la tabella del filmato.

```
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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)


- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un batch di elementi mediante più istruzioni SELECT.
- Aggiunta di un batch di articoli eseguendo più istruzioni INSERT.
- Aggiornamento di un batch di elementi mediante più istruzioni UPDATE.
- Eliminazione di un batch di elementi mediante più istruzioni DELETE.

AWS SDK for .NET

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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.WriteLine(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;
}
```

- Per i dettagli sull'API, [BatchExecuteStatement](#) consulta AWS SDK for .NET API Reference.

Esecuzione di una query mediante PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un articolo eseguendo un'istruzione SELECT.
- Aggiunta di un elemento eseguendo un'istruzione INSERT.
- Aggiornamento di un elemento eseguendo un'istruzione UPDATE.
- Eliminazione di un elemento eseguendo un'istruzione DELETE.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
}

```

- Per i dettagli sull'API, [ExecuteStatement](#) consulta AWS SDK for .NET API Reference.

Utilizzo di un modello di documento

Il seguente esempio di codice mostra come eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) e batch utilizzando un modello di documento per DynamoDB e un SDK.

AWS

Per ulteriori informazioni, consulta [Modello di documento](#).

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esecuzione di operazioni CRUD utilizzando un modello di documento.

```
/// <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.ExpressionAttributeValues[":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);
        }
    }
}

```

Esecuzione di operazioni di scrittura in batch utilizzando un modello di documento.

```

    /// <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();
}
}
```

Scansione di una tabella utilizzando un modello di documento.

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

```

Esecuzione di query e scansione di una tabella utilizzando un modello di documento.

```

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

Utilizzo di un modello di persistenza degli oggetti di alto livello

Il seguente esempio di codice mostra come eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) e in batch utilizzando un modello di persistenza a oggetti per DynamoDB e un SDK. AWS

Per ulteriori informazioni, consulta [Modello di persistenza degli oggetti](#).

AWS SDK for .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esecuzione di operazioni CRUD utilizzando un modello di persistenza degli oggetti di alto livello.

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

```

Esecuzione di operazioni di scrittura in batch utilizzando un modello di persistenza degli oggetti di alto livello.

```

/// <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);
    }
}

```

Mappatura dei dati arbitrari su una tabella utilizzando un modello di persistenza degli oggetti di alto livello.

```

/// <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);
}
}
```

Esecuzione di query e scansione di una tabella utilizzando un modello di persistenza degli oggetti di alto livello.

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

Esempi serverless

Richiama una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

AWS SDK for .NET

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento DynamoDB con Lambda utilizzando .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// 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 AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Segnalazione degli errori degli elementi batch per le funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione degli errori degli elementi batch di DynamoDB con Lambda utilizzando .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// 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 AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
    }
}
```

```
List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>();
StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

foreach (var record in dynamoEvent.Records)
{
    try
    {
        var sequenceNumber = record.Dynamodb.SequenceNumber;
        context.Logger.LogInformation(sequenceNumber);
    }
    catch (Exception ex)
    {
        context.Logger.LogError(ex.Message);
        batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
{ ItemIdentifier = record.Dynamodb.SequenceNumber });
    }
}

if (batchItemFailures.Count > 0)
{
    streamsEventResponse.BatchItemFailures = batchItemFailures;
}

context.Logger.LogInformation("Stream processing complete.");
return streamsEventResponse;
}
}
```

Esempi di utilizzo di Amazon EC2 AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon EC2. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.


Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon EC2

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon EC2.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeSecurityGroups](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

AllocateAddress

Il seguente esempio di codice mostra come utilizzare `AllocateAddress`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [AllocateAddress](#) consulta AWS SDK for .NET API Reference.

AssociateAddress

Il seguente esempio di codice mostra come utilizzare `AssociateAddress`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [AssociateAddress](#) consulta AWS SDK for .NET API Reference.

AuthorizeSecurityGroupIngress

Il seguente esempio di codice mostra come utilizzare `AuthorizeSecurityGroupIngress`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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();
}

```

- Per i dettagli sull'API, [AuthorizeSecurityGroupIngress](#) consulta AWS SDK for .NET API Reference.

CreateKeyPair

Il seguente esempio di codice mostra come utilizzare `CreateKeyPair`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateKeyPair](#) consulta AWS SDK for .NET API Reference.

CreateLaunchTemplate

Il seguente esempio di codice mostra come utilizzare `CreateLaunchTemplate`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateLaunchTemplate](#) consulta AWS SDK for .NET API Reference.

CreateSecurityGroup

Il seguente esempio di codice mostra come utilizzare `CreateSecurityGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateSecurityGroup](#) consulta AWS SDK for .NET API Reference.

DeleteKeyPair

Il seguente esempio di codice mostra come utilizzare `DeleteKeyPair`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);
        }
    }
}
```

- Per i dettagli sull'API, [DeleteKeyPair](#) consulta AWS SDK for .NET API Reference.

DeleteLaunchTemplate

Il seguente esempio di codice mostra come utilizzare `DeleteLaunchTemplate`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteLaunchTemplate](#) consulta AWS SDK for .NET API Reference.

DeleteSecurityGroup

Il seguente esempio di codice mostra come utilizzare `DeleteSecurityGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteSecurityGroup](#) consulta AWS SDK for .NET API Reference.

DescribeAvailabilityZones

Il seguente esempio di codice mostra come utilizzare `DescribeAvailabilityZones`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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();
}
```

- Per i dettagli sull'API, [DescribeAvailabilityZones](#) consulta AWS SDK for .NET API Reference.

DescribeIamInstanceProfileAssociations

Il seguente esempio di codice mostra come utilizzare `DescribeIamInstanceProfileAssociations`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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];
}
```

- Per i dettagli sull'API, [DescribeIamInstanceProfileAssociations](#) consulta AWS SDK for .NET API Reference.

DescribeInstanceTypes

Il seguente esempio di codice mostra come utilizzare `DescribeInstanceTypes`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeInstanceTypes](#) consulta AWS SDK for .NET API Reference.

DescribeInstances

Il seguente esempio di codice mostra come utilizzare `DescribeInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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.WriteLine($"Instance ID: {instance.InstanceId} ");
            Console.WriteLine($"Current State: {instance.State.Name}");
        }
    }
}
}
```

- Per i dettagli sull'API, [DescribeInstances](#) consulta AWS SDK for .NET API Reference.

DescribeKeyPairs

Il seguente esempio di codice mostra come utilizzare `DescribeKeyPairs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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();
}

```

- Per i dettagli sull'API, [DescribeKeyPairs](#) consulta AWS SDK for .NET API Reference.

DescribeSecurityGroups

Il seguente esempio di codice mostra come utilizzare `DescribeSecurityGroups`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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.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}");
    });
}

```

- Per i dettagli sull'API, [DescribeSecurityGroups](#) consulta AWS SDK for .NET API Reference.

DescribeSubnets

Il seguente esempio di codice mostra come utilizzare `DescribeSubnets`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, [DescribeSubnets](#) consulta AWS SDK for .NET API Reference.

DescribeVpcs

Il seguente esempio di codice mostra come utilizzare `DescribeVpcs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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];
}

```

- Per i dettagli sull'API, [DescribeVpcs](#) consulta AWS SDK for .NET API Reference.

DisassociateAddress

Il seguente esempio di codice mostra come utilizzare `DisassociateAddress`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [DisassociateAddress](#) consulta AWS SDK for .NET API Reference.

RebootInstances

Il seguente esempio di codice mostra come utilizzare `RebootInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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.");
    }
}

```

Sostituisci il profilo per un'istanza, riavvia e riavvia un server Web.

```

/// <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}");
    }

```

- Per i dettagli sull'API, [RebootInstances](#) consulta AWS SDK for .NET API Reference.

ReleaseAddress

Il seguente esempio di codice mostra come utilizzare `ReleaseAddress`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }
```

- Per i dettagli sull'API, [ReleaseAddress](#) consulta AWS SDK for .NET API Reference.

ReplaceIamInstanceProfileAssociation

Il seguente esempio di codice mostra come utilizzare `ReplaceIamInstanceProfileAssociation`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```

- Per i dettagli sull'API, [ReplacelamInstanceProfileAssociation](#) consulta AWS SDK for .NET API Reference.

RunInstances

Il seguente esempio di codice mostra come utilizzare `RunInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [RunInstances](#) consulta AWS SDK for .NET API Reference.

StartInstances

Il seguente esempio di codice mostra come utilizzare `StartInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [StartInstances](#) consulta AWS SDK for .NET API Reference.

StopInstances

Il seguente esempio di codice mostra come utilizzare `StopInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [StopInstances](#) consulta AWS SDK for .NET API Reference.

TerminateInstances

Il seguente esempio di codice mostra come utilizzare `TerminateInstances`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [TerminateInstances](#) consulta AWS SDK for .NET API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisce le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui lo scenario interattivo al prompt dei comandi.

```
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($"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;
}

```

Crea una classe che racchiuda le operazioni di dimensionamento automatico e Amazon EC2.

```

/// <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 (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 }
            });
    }
}

```

Crea una classe che racchiuda le operazioni di Elastic Load Balancing.

```

/// <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);
    }
    }
}
}

```

Crea una classe che utilizzi DynamoDB per simulare un servizio di raccomandazione.

```

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

Crea una classe che racchiuda le operazioni di Systems Manager.

```
/// <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 });
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)

- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Nozioni di base sulle istanze

L'esempio di codice seguente mostra come:

- Creare una coppia di chiavi e un gruppo di sicurezza.
- Selezionare un'Amazon Machine Image (AMI) e un tipo di istanza compatibile e quindi creare un'istanza.
- Arrestare e riavviare l'istanza.
- Associazione di un indirizzo IP elastico all'istanza
- Connettiti alla tua istanza con SSH, quindi elimina le risorse.

AWS SDK for .NET

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario al prompt dei comandi.

```
/// <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.WriteLine("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.WriteLine("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.Write("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();
    }
}

```

Definisci una classe che racchiude le operazioni EC2.

```

/// <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($"Instance ID: {instance.InstanceId}");
                Console.WriteLine($"    \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.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($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{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($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{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($"{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;
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AllocateAddress](#)
 - [AssociateAddress](#)

- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Esempi di utilizzo di Amazon ECS AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon ECS. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Salve Amazon ECS

Il seguente esempio di codice mostra come iniziare a usare Amazon ECS.

AWS SDK for .NET

Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}  
}
```

- Per i dettagli sull'API, [ListClusters](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

ListClusters

Il seguente esempio di codice mostra come utilizzare `ListClusters`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListClusters](#) consulta AWS SDK for .NET API Reference.

ListServices

Il seguente esempio di codice mostra come utilizzare `ListServices`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListServices](#) consulta AWS SDK for .NET API Reference.

ListTasks

Il seguente esempio di codice mostra come utilizzare `ListTasks`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListTasks](#) consulta AWS SDK for .NET API Reference.

Scenari

Ottieni informazioni ARN per cluster, servizi e attività

L'esempio di codice seguente mostra come:

- Ottieni un elenco di tutti i cluster.
- Ottieni servizi per un cluster.
- Ottieni attività per un cluster.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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));
}
}

```

Metodi wrapper richiamati dallo scenario per gestire le azioni di Amazon ECS.

```

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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [ListClusters](#)
 - [ListServices](#)
 - [ListTasks](#)

Elastic Load Balancing - Esempi di utilizzo della versione 2 AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Elastic Load Balancing - Versione 2.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateListener

Il seguente esempio di codice mostra come utilizzare `CreateListener`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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];
}
```

- Per i dettagli sull'API, [CreateListener](#) consulta AWS SDK for .NET API Reference.

CreateLoadBalancer

Il seguente esempio di codice mostra come utilizzare `CreateLoadBalancer`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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];
}
```

- Per i dettagli sull'API, [CreateLoadBalancer](#) consulta AWS SDK for .NET API Reference.

CreateTargetGroup

Il seguente esempio di codice mostra come utilizzare `CreateTargetGroup`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateTargetGroup](#) consulta AWS SDK for .NET API Reference.

DeleteLoadBalancer

Il seguente esempio di codice mostra come utilizzare `DeleteLoadBalancer`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteLoadBalancer](#) consulta AWS SDK for .NET API Reference.

DeleteTargetGroup

Il seguente esempio di codice mostra come utilizzare `DeleteTargetGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);
    }
}
}
```

- Per i dettagli sull'API, [DeleteTargetGroup](#) consulta AWS SDK for .NET API Reference.

DescribeLoadBalancers

Il seguente esempio di codice mostra come utilizzare `DescribeLoadBalancers`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeLoadBalancers](#) consulta AWS SDK for .NET API Reference.

DescribeTargetHealth

Il seguente esempio di codice mostra come utilizzare `DescribeTargetHealth`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeTargetHealth](#) consulta AWS SDK for .NET API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisce le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.

- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui lo scenario interattivo al prompt dei comandi.

```
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"
        + "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.TargetGroupM
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($"http://{endPoint}\n");
}
else
{
    Console.WriteLine(
        "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
        + "manually verifying that your VPC and security group are
configured correctly and that\n"
        + "you can successfully make a GET request to the load balancer
endpoint:\n");
    Console.WriteLine($"http://{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;
}
```

Crea una classe che racchiuda le operazioni di dimensionamento automatico e Amazon EC2.

```
/// <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="deleteKeyName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyName });
            File.Delete($"{deleteKeyName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyName} 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 (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 }
        });
}
}

```

Crea una classe che racchiuda le operazioni di Elastic Load Balancing.

```

/// <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);
        }
    }
}
}
```

Crea una classe che utilizzi DynamoDB per simulare un servizio di raccomandazione.

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

Crea una classe che racchiuda le operazioni di Systems Manager.

```
/// <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 });
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [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 esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with EventBridge.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.


Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve EventBridge

I seguenti esempi di codice mostrano come iniziare a utilizzare EventBridge.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListEventBuses](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

DeleteRule

Il seguente esempio di codice mostra come utilizzare `DeleteRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina una regola in base al nome della stessa.

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

- Per i dettagli sull'API, [DeleteRule](#) consulta AWS SDK for .NET API Reference.

DescribeRule

Il seguente esempio di codice mostra come utilizzare `DescribeRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ottieni lo stato di una regola utilizzando la descrizione della regola.

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

- Per i dettagli sull'API, [DescribeRule](#) consulta AWS SDK for .NET API Reference.

DisableRule

Il seguente esempio di codice mostra come utilizzare `DisableRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Disabilita una regola in base al nome della stessa.

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

- Per i dettagli sull'API, [DisableRule](#) consulta AWS SDK for .NET API Reference.

EnableRule

Il seguente esempio di codice mostra come utilizzare `EnableRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Abilita una regola in base al nome della stessa.

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

- Per i dettagli sull'API, [EnableRule](#) consulta AWS SDK for .NET API Reference.

ListRuleNamesByTarget

Il seguente esempio di codice mostra come utilizzare `ListRuleNamesByTarget`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca tutti i nomi delle regole utilizzando la destinazione.

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

- Per i dettagli sull'API, [ListRuleNamesByTarget](#) consulta AWS SDK for .NET API Reference.

ListRules

Il seguente esempio di codice mostra come utilizzare `ListRules`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca tutte le regole per un router di eventi.

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

- Per i dettagli sull'API, [ListRules](#) consulta AWS SDK for .NET API Reference.

ListTargetsByRule

Il seguente esempio di codice mostra come utilizzare `ListTargetsByRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca tutte le destinazioni di una regola utilizzando il nome della stessa.

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

- Per i dettagli sull'API, [ListTargetsByRule](#) consulta AWS SDK for .NET API Reference.

PutEvents

Il seguente esempio di codice mostra come utilizzare `PutEvents`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un evento che corrisponde a un modello personalizzato per una regola.

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

```
}

```

- Per i dettagli sull'API, [PutEvents](#) consulta AWS SDK for .NET API Reference.

PutRule

Il seguente esempio di codice mostra come utilizzare `PutRule`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una regola che si attiva quando un oggetto viene aggiunto a un bucket di Amazon Simple Storage Service.

```

    /// <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;
}

```

Crea una regola che utilizza un modello personalizzato.

```

/// <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;
}

```

- Per i dettagli sull'API, [PutRule](#) consulta AWS SDK for .NET API Reference.

PutTargets

Il seguente esempio di codice mostra come utilizzare PutTargets.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Aggiungi un argomento Amazon SNS come destinazione per una regola.

```
/// <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;
}

```

Aggiungi un trasformatore di input a una destinazione per una regola.

```

/// <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;
}

```

- Per i dettagli sull'API, [PutTargets](#) consulta AWS SDK for .NET API Reference.

RemoveTargets

Il seguente esempio di codice mostra come utilizzare `RemoveTargets`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Rimuovi tutte le destinazioni di una regola utilizzando il nome della stessa.

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

- Per i dettagli sull'API, [RemoveTargets](#) consulta AWS SDK for .NET API Reference.

Scenari

Nozioni di base su regole e destinazioni

L'esempio di codice seguente mostra come:

- Creare una regola e aggiungervi una destinazione.
- Abilitare e disabilitare regole.
- Elencare e aggiornare regole e destinazioni.
- Inviare eventi e quindi eliminare le risorse.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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($"\\tRule: {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($"\\tUpdated 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($"\tDelete all targets and event rule {eventRuleName}?
(y/n)"))
{
    Console.WriteLine($"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($"\tDelete Amazon SNS subscription topic {topicName}?
(y/n)"))
{
    Console.WriteLine($"Deleting topic.");
    await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
    {
        TopicArn = topicArn
    });
}

var bucketName = _configuration["testBucketName"];
if (GetYesNoResponse($"\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
{
    Console.WriteLine($"Deleting 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;
}
}

```

Crea una classe che racchiuda le operazioni. EventBridge

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

```

};
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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [DeleteRule](#)
 - [DescribeRule](#)
 - [DisableRule](#)
 - [EnableRule](#)

- [ListRuleNamesByTarget](#)
- [ListRules](#)
- [ListTargetsByRule](#)
- [PutEvents](#)
- [PutRule](#)
- [PutTargets](#)

AWS Glue esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with AWS Glue.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve AWS Glue

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Glue.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
        }
    }
}
```

```
}  
}
```

- Per i dettagli sull'API, [ListJobs](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateCrawler

Il seguente esempio di codice mostra come utilizzare `CreateCrawler`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateCrawler](#) consulta AWS SDK for .NET API Reference.

CreateJob

Il seguente esempio di codice mostra come utilizzare `CreateJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateJob](#) consulta AWS SDK for .NET API Reference.

DeleteCrawler

Il seguente esempio di codice mostra come utilizzare `DeleteCrawler`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteCrawler](#) consulta AWS SDK for .NET API Reference.

DeleteDatabase

Il seguente esempio di codice mostra come utilizzare `DeleteDatabase`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteDatabase](#) consulta AWS SDK for .NET API Reference.

DeleteJob

Il seguente esempio di codice mostra come utilizzare `DeleteJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteJob](#) consulta AWS SDK for .NET API Reference.

DeleteTable

Il seguente esempio di codice mostra come utilizzare `DeleteTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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


- Per i dettagli sull'API, [DeleteTable](#) consulta AWS SDK for .NET API Reference.

GetCrawler

Il seguente esempio di codice mostra come utilizzare `GetCrawler`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetCrawler](#) consulta AWS SDK for .NET API Reference.

GetDatabase

Il seguente esempio di codice mostra come utilizzare `GetDatabase`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetDatabase](#) consulta AWS SDK for .NET API Reference.

GetJobRun

Il seguente esempio di codice mostra come utilizzare `GetJobRun`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [GetJobRun](#) consulta AWS SDK for .NET API Reference.

GetJobRuns

Il seguente esempio di codice mostra come utilizzare `GetJobRuns`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetJobRuns](#) consulta AWS SDK for .NET API Reference.

GetTables

Il seguente esempio di codice mostra come utilizzare `GetTables`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetTables](#) consulta AWS SDK for .NET API Reference.

ListJobs

Il seguente esempio di codice mostra come utilizzare `ListJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListJobs](#) consulta AWS SDK for .NET API Reference.

StartCrawler

Il seguente esempio di codice mostra come utilizzare `StartCrawler`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [StartCrawler](#) consulta AWS SDK for .NET API Reference.

StartJobRun

Il seguente esempio di codice mostra come utilizzare `StartJobRun`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [StartJobRun](#) consulta AWS SDK for .NET API Reference.

Scenari

Nozioni di base su crawler e processi

L'esempio di codice seguente mostra come:

- Crea un crawler che esegue la scansione di un bucket Amazon S3 pubblico e genera un database di metadati in formato CSV.
- Elenca le informazioni su database e tabelle nel tuo AWS Glue Data Catalog.
- Crea un processo per estrarre i dati CSV dal bucket S3, trasformare i dati e caricare l'output in formato JSON in un altro bucket S3.
- Elenca le informazioni sulle esecuzioni dei processi, visualizza i dati trasformati e pulisci le risorse.

Per ulteriori informazioni, consulta [Tutorial: Guida introduttiva a AWS Glue Studio](#).

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una classe che racchiuda le AWS Glue funzioni utilizzate nello scenario.

```
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;
}
}
```


Creazione di una classe che esegue lo scenario.

```
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 jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
        var jobRunComplete = false;
        var jobRun = new JobRun();
        do
        {
            jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
            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);
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)

- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

Esempi IAM che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with IAM.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello IAM

Gli esempi di codice seguenti mostrano come iniziare a utilizzare IAM.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListPolicies](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

AddUserToGroup

Il seguente esempio di codice mostra come utilizzare `AddUserToGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AddUserToGroup](#) consulta AWS SDK for .NET API Reference.

AttachRolePolicy

Il seguente esempio di codice mostra come utilizzare `AttachRolePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AttachRolePolicy](#) consulta AWS SDK for .NET API Reference.

CreateAccessKey

Il seguente esempio di codice mostra come utilizzare `CreateAccessKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateAccessKey](#) consulta AWS SDK for .NET API Reference.

CreateGroup

Il seguente esempio di codice mostra come utilizzare `CreateGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateGroup](#) consulta AWS SDK for .NET API Reference.

CreateInstanceProfile

Il seguente esempio di codice mostra come utilizzare `CreateInstanceProfile`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, [CreateInstanceProfile](#) consulta AWS SDK for .NET API Reference.

CreatePolicy

Il seguente esempio di codice mostra come utilizzare `CreatePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreatePolicy](#) consulta AWS SDK for .NET API Reference.

CreateRole

Il seguente esempio di codice mostra come utilizzare `CreateRole`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateRole](#) consulta AWS SDK for .NET API Reference.

CreateServiceLinkedRole

Il seguente esempio di codice mostra come utilizzare `CreateServiceLinkedRole`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateServiceLinkedRole](#) consulta AWS SDK for .NET API Reference.

CreateUser

Il seguente esempio di codice mostra come utilizzare `CreateUser`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateUser](#) consulta AWS SDK for .NET API Reference.

DeleteAccessKey

Il seguente esempio di codice mostra come utilizzare `DeleteAccessKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteAccessKey](#) consulta AWS SDK for .NET API Reference.

DeleteGroup

Il seguente esempio di codice mostra come utilizzare `DeleteGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteGroup](#) consulta AWS SDK for .NET API Reference.

DeleteGroupPolicy

Il seguente esempio di codice mostra come utilizzare `DeleteGroupPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}

```

- Per i dettagli sull'API, [DeleteGroupPolicy](#) consulta AWS SDK for .NET API Reference.

DeleteInstanceProfile

Il seguente esempio di codice mostra come utilizzare `DeleteInstanceProfile`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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

- Per i dettagli sull'API, [DeleteInstanceProfile](#) consulta AWS SDK for .NET API Reference.

DeletePolicy

Il seguente esempio di codice mostra come utilizzare `DeletePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeletePolicy](#) consulta AWS SDK for .NET API Reference.

DeleteRole

Il seguente esempio di codice mostra come utilizzare `DeleteRole`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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


- Per i dettagli sull'API, [DeleteRole](#) consulta AWS SDK for .NET API Reference.

DeleteRolePolicy

Il seguente esempio di codice mostra come utilizzare `DeleteRolePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteRolePolicy](#) consulta AWS SDK for .NET API Reference.

DeleteUser

Il seguente esempio di codice mostra come utilizzare `DeleteUser`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteUser](#) consulta AWS SDK for .NET API Reference.

DeleteUserPolicy

Il seguente esempio di codice mostra come utilizzare `DeleteUserPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
    }

```

- Per i dettagli sull'API, [DeleteUserPolicy](#) consulta AWS SDK for .NET API Reference.

DetachRolePolicy

Il seguente esempio di codice mostra come utilizzare `DetachRolePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    /// <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;  
}
```

- Per i dettagli sull'API, [DetachRolePolicy](#) consulta AWS SDK for .NET API Reference.

GetAccountPasswordPolicy

Il seguente esempio di codice mostra come utilizzare `GetAccountPasswordPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetAccountPasswordPolicy](#) consulta AWS SDK for .NET API Reference.

GetPolicy

Il seguente esempio di codice mostra come utilizzare `GetPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetPolicy](#) consulta AWS SDK for .NET API Reference.

GetRole

Il seguente esempio di codice mostra come utilizzare `GetRole`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetRole](#) consulta AWS SDK for .NET API Reference.

GetUser

Il seguente esempio di codice mostra come utilizzare `GetUser`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetUser](#) consulta AWS SDK for .NET API Reference.

ListAttachedRolePolicies

Il seguente esempio di codice mostra come utilizzare `ListAttachedRolePolicies`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListAttachedRolePolicies](#) consulta AWS SDK for .NET API Reference.

ListGroups

Il seguente esempio di codice mostra come utilizzare `ListGroups`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListGroups](#) consulta AWS SDK for .NET API Reference.

ListPolicies

Il seguente esempio di codice mostra come utilizzare `ListPolicies`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListPolicies](#) consulta AWS SDK for .NET API Reference.

ListRolePolicies

Il seguente esempio di codice mostra come utilizzare `ListRolePolicies`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListRolePolicies](#) consulta AWS SDK for .NET API Reference.

ListRoles

Il seguente esempio di codice mostra come utilizzare `ListRoles`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListRoles](#) consulta AWS SDK for .NET API Reference.

ListSAMLProviders

Il seguente esempio di codice mostra come utilizzare `ListSAMLProviders`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta la sezione [ListSAMLProviders](#) nella Documentazione di riferimento dell'API AWS SDK for .NET .

ListUsers

Il seguente esempio di codice mostra come usare `ListUsers`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListUsers](#) consulta AWS SDK for .NET API Reference.

PutGroupPolicy

Il seguente esempio di codice mostra come utilizzare `PutGroupPolicy`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [PutGroupPolicy](#) consulta AWS SDK for .NET API Reference.

PutRolePolicy

Il seguente esempio di codice mostra come utilizzare `PutRolePolicy`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [PutRolePolicy](#) consulta AWS SDK for .NET API Reference.

RemoveUserFromGroup

Il seguente esempio di codice mostra come utilizzare `RemoveUserFromGroup`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [RemoveUserFromGroup](#) consulta AWS SDK for .NET API Reference.

Scenari

Creazione e gestione di un servizio resiliente

Il seguente esempio di codice mostra come creare un servizio Web con bilanciamento del carico che restituisca consigli su libri, film e canzoni. L'esempio mostra come il servizio risponde ai guasti e spiega come ristrutturarlo per una maggiore resilienza in caso di guasti.

- Utilizza un gruppo con dimensionamento automatico Amazon EC2 per creare istanze Amazon Elastic Compute Cloud (Amazon EC2) basate su un modello di avvio e per mantenere il numero di istanze entro un intervallo specificato.
- Gestisci e distribuisce le richieste HTTP con Elastic Load Balancing.
- Monitora lo stato delle istanze in un gruppo con dimensionamento automatico e inoltra le richieste soltanto alle istanze integre.
- Esegui un server Web Python su ogni istanza EC2 per gestire le richieste HTTP. Il server Web risponde con consigli e controlli dell'integrità.
- Simula un servizio di raccomandazione con una tabella Amazon DynamoDB.
- Controlla la risposta del server web alle richieste e ai controlli sanitari aggiornando AWS Systems Manager i parametri.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui lo scenario interattivo al prompt dei comandi.

```
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.TargetGroupName,
            protocol, port, defaultVpc.VpcId);

        await
            _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadBalancerName,
            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($"http://{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;
    }
}
```

Crea una classe che racchiuda le operazioni di dimensionamento automatico e Amazon EC2.

```
/// <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="newKeyName">The name of the new key pair.</param>
```



```
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyName });
        await File.WriteAllTextAsync($"{newKeyName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyName });
        File.Delete($"{deleteKeyName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyName} 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 (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 }
            });
    }

```

```
}  
}
```

Crea una classe che racchiuda le operazioni di Elastic Load Balancing.

```
/// <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 }
                    }
                );
        }
        catch { }
    }
}
```

```

        });

        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);
        }
    }
}
}

```

Crea una classe che utilizzi DynamoDB per simulare un servizio di raccomandazione.

```

/// <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");
        }
    }
}

```

Crea una classe che racchiuda le operazioni di Systems Manager.

```

/// <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 });  
    }  
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [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](#)

Creazione di un gruppo e aggiunta di un utente

L'esempio di codice seguente mostra come:

- Crea un gruppo e concedigli autorizzazioni di accesso complete ad Amazon S3.
- Crea un nuovo utente senza autorizzazioni per accedere ad Amazon S3.
- Aggiungi l'utente al gruppo e verifica che ora disponga delle autorizzazioni per Amazon S3, quindi ripulisci le risorse.

AWS SDK for .NET

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AddUserToGroup](#)
 - [AttachRolePolicy](#)

- [CreateAccessKey](#)
- [CreateGroup](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeleteGroup](#)
- [DeleteGroupPolicy](#)
- [DeleteUser](#)
- [PutGroupPolicy](#)
- [RemoveUserFromGroup](#)

Creazione di un utente e assunzione di un ruolo

Il seguente esempio di codice mostra come creare un utente e assumere un ruolo.

 Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

- Crea un utente che non disponga di autorizzazioni.
- Crea un ruolo che conceda l'autorizzazione per elencare i bucket Amazon S3 per l'account.
- Aggiungi una policy per consentire all'utente di assumere il ruolo.
- Assumi il ruolo ed elenca i bucket S3 utilizzando le credenziali temporanee, quindi ripulisci le risorse.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();  
    }  
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Esempi di Amazon Keyspaces utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Amazon Keyspaces.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Amazon Keyspaces

I seguenti esempi di codice mostrano come iniziare a utilizzare Amazon Keyspaces.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
    }
}
```

- Per i dettagli sull'API, [ListKeyspaces](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateKeyspace

Il seguente esempio di codice mostra come utilizzare `CreateKeyspace`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateKeyspace](#) consulta AWS SDK for .NET API Reference.

CreateTable

Il seguente esempio di codice mostra come utilizzare CreateTable.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateTable](#) consulta AWS SDK for .NET API Reference.

DeleteKeyspace

Il seguente esempio di codice mostra come utilizzare `DeleteKeyspace`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteKeyspace](#) consulta AWS SDK for .NET API Reference.

DeleteTable

Il seguente esempio di codice mostra come utilizzare `DeleteTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [DeleteTable](#) consulta AWS SDK for .NET API Reference.

GetKeyspace

Il seguente esempio di codice mostra come utilizzare `GetKeyspace`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetKeyspace](#) consulta AWS SDK for .NET API Reference.

GetTable

Il seguente esempio di codice mostra come utilizzare `GetTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetTable](#) consulta AWS SDK for .NET API Reference.

ListKeyspaces

Il seguente esempio di codice mostra come utilizzare `ListKeyspaces`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListKeyspaces](#) consulta AWS SDK for .NET API Reference.

ListTables

Il seguente esempio di codice mostra come utilizzare `ListTables`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [ListTables](#) consulta AWS SDK for .NET API Reference.

RestoreTable

Il seguente esempio di codice mostra come utilizzare `RestoreTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, [RestoreTable](#) consulta AWS SDK for .NET API Reference.

UpdateTable

Il seguente esempio di codice mostra come utilizzare `UpdateTable`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateTable](#) consulta AWS SDK for .NET API Reference.

Scenari

Inizia con gli spazi chiave e le tabelle

L'esempio di codice seguente mostra come:

- Crea uno spazio chiave e una tabella. Lo schema della tabella contiene i dati dei film e il point-in-time ripristino è abilitato.
- Connect al keyspace utilizzando una connessione TLS sicura con autenticazione SigV4.
- Esegui una query sulla tabella. Aggiungi, recupera e aggiorna i dati dei film.
- Aggiorna la tabella. Aggiungi una colonna per tenere traccia dei film guardati.
- Ripristina lo stato precedente della tabella e ripulisci le risorse.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 driver 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.WriteLine(".");
    }
}

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 keyspaces.</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 keyspaces 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!;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateKeyspace](#)
 - [CreateTable](#)
 - [DeleteKeyspace](#)
 - [DeleteTable](#)
 - [GetKeyspace](#)
 - [GetTable](#)
 - [ListKeyspaces](#)
 - [ListTables](#)
 - [RestoreTable](#)
 - [UpdateTable](#)

Esempi di Kinesis che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Kinesis.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)
- [Esempi serverless](#)

Azioni

AddTagsToStream

Il seguente esempio di codice mostra come utilizzare `AddTagsToStream`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
```



```
public static async Task Main()
{
    IAmazonKinesis client = new AmazonKinesisClient();

    string streamName = "AmazonKinesisStream";
    var tags = new Dictionary<string, string>
    {
        { "Project", "Sample Kinesis Project" },
        { "Application", "Sample Kinesis App" },
    };

    var success = await ApplyTagsToStreamAsync(client, streamName, tags);

    if (success)
    {
        Console.WriteLine($"Taggs successfully added to {streamName}.");
    }
    else
    {
        Console.WriteLine("Tags were not added to the stream.");
    }
}

/// <summary>
/// Applies the set of tags to the named Kinesis stream.
/// </summary>
/// <param name="client">The initialized Kinesis client.</param>
/// <param name="streamName">The name of the Kinesis stream to which
/// the tags will be attached.</param>
/// <param name="tags">A sictionary containing key/value pairs which
/// will be used to create the Kinesis tags.</param>
/// <returns>A Boolean value which represents the success or failure
/// of AddTagsToStreamAsync.</returns>
public static async Task<bool> ApplyTagsToStreamAsync(
    IAmazonKinesis client,
    string streamName,
    Dictionary<string, string> tags)
{
    var request = new AddTagsToStreamRequest
    {
        StreamName = streamName,
        Tags = tags,
    };
};
```

```
        var response = await client.AddTagsToStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Per i dettagli sull'API, [AddTagsToStream](#) consulta AWS SDK for .NET API Reference.

CreateStream

Il seguente esempio di codice mostra come utilizzare `CreateStream`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [CreateStream](#) consulta AWS SDK for .NET API Reference.

DeleteStream

Il seguente esempio di codice mostra come utilizzare `DeleteStream`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
}
}
```

- Per i dettagli sull'API, [DeleteStream](#) consulta AWS SDK for .NET API Reference.

DeregisterStreamConsumer

Il seguente esempio di codice mostra come utilizzare `DeregisterStreamConsumer`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;  
    }  
}
```

- Per i dettagli sull'API, [DeregisterStreamConsumer](#) consulta AWS SDK for .NET API Reference.

ListStreamConsumers

Il seguente esempio di codice mostra come utilizzare `ListStreamConsumers`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```


- Per i dettagli sull'API, [ListStreamConsumers](#) consulta AWS SDK for .NET API Reference.

ListStreams

Il seguente esempio di codice mostra come utilizzare `ListStreams`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [ListStreams](#) consulta AWS SDK for .NET API Reference.

ListTagsForStream

Il seguente esempio di codice mostra come utilizzare `ListTagsForStream`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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}"));
    }
}


```

- Per i dettagli sull'API, [ListTagsForStream](#) consulta AWS SDK for .NET API Reference.

RegisterStreamConsumer

Il seguente esempio di codice mostra come utilizzare `RegisterStreamConsumer`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [RegisterStreamConsumer](#) consulta AWS SDK for .NET API Reference.

Esempi serverless

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

AWS SDK for .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
```

```
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.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;
}
}

```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .NET.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.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 esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with AWS KMS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

CreateAlias

Il seguente esempio di codice mostra come utilizzare `CreateAlias`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateAlias](#) consulta AWS SDK for .NET API Reference.

CreateGrant

Il seguente esempio di codice mostra come utilizzare `CreateGrant`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateGrant](#) consulta AWS SDK for .NET API Reference.

CreateKey

Il seguente esempio di codice mostra come utilizzare `CreateKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateKey](#) consulta AWS SDK for .NET API Reference.

DescribeKey

Il seguente esempio di codice mostra come utilizzare `DescribeKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeKey](#) consulta AWS SDK for .NET API Reference.

DisableKey

Il seguente esempio di codice mostra come utilizzare `DisableKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DisableKey](#) consulta AWS SDK for .NET API Reference.

EnableKey

Il seguente esempio di codice mostra come utilizzare `EnableKey`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [EnableKey](#) consulta AWS SDK for .NET API Reference.

ListAliases

Il seguente esempio di codice mostra come utilizzare `ListAliases`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

```
}
```

- Per i dettagli sull'API, [ListAliases](#) consulta AWS SDK for .NET API Reference.

ListGrants

Il seguente esempio di codice mostra come utilizzare `ListGrants`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

- Per i dettagli sull'API, [ListGrants](#) consulta AWS SDK for .NET API Reference.

ListKeys

Il seguente esempio di codice mostra come utilizzare `ListKeys`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

- Per i dettagli sull'API, [ListKeys](#) consulta AWS SDK for .NET API Reference.

Esempi di utilizzo di Lambda AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Lambda.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Lambda

L'esempio di codice seguente mostra come iniziare a utilizzare Lambda.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListFunctions](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Azioni

CreateFunction

Il seguente esempio di codice mostra come utilizzare `CreateFunction`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateFunction](#) consulta AWS SDK for .NET API Reference.

DeleteFunction

Il seguente esempio di codice mostra come utilizzare `DeleteFunction`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteFunction](#) consulta AWS SDK for .NET API Reference.

GetFunction

Il seguente esempio di codice mostra come utilizzare `GetFunction`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetFunction](#) consulta AWS SDK for .NET API Reference.

Invoke

Il seguente esempio di codice mostra come utilizzare `Invoke`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sulle API, consulta [Invoke](#) nella Documentazione di riferimento delle API AWS SDK for .NET .

ListFunctions

Il seguente esempio di codice mostra come usare `ListFunctions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListFunctions](#) consulta AWS SDK for .NET API Reference.

UpdateFunctionCode

Il seguente esempio di codice mostra come utilizzare `UpdateFunctionCode`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateFunctionCode](#) consulta AWS SDK for .NET API Reference.

UpdateFunctionConfiguration

Il seguente esempio di codice mostra come utilizzare `UpdateFunctionConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateFunctionConfiguration](#) consulta AWS SDK for .NET API Reference.

Scenari

Nozioni di base sulle funzioni

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Richiamare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Richiamare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni sull'utilizzo di Lambda, consulta [Creare una funzione Lambda con la console](#).

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Creare metodi che eseguono operazioni Lambda.

```
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;
    }
}
```

Creare una funzione che esegue lo scenario.

```
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.Write("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.Write(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

await lambdaWrapper.UpdateFunctionConfigurationAsync(
    functionName,
    calculatorHandler,
    new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.Write(".");
}
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.WriteLine("Enter the first value: ");
            value = Console.ReadLine();
        }
        while (value == string.Empty);

        string? value2;
        do
        {
            Console.WriteLine("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();
    }
}
```

Definire un gestore Lambda che incrementa un numero.

```
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;
        }
    }
}

```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```

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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateFunction](#)

- [DeleteFunction](#)
- [GetFunction](#)
- [Invoke](#)
- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

Esempi serverless

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.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;
    }
}
```

Richiama una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

AWS SDK for .NET

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento DynamoDB con Lambda utilizzando .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// 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 AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");
        }
    }
}
```

```
        context.Logger.LogInformation(JsonSerializer.Serialize(record));
    }

    context.Logger.LogInformation("Stream processing complete.");
}
}
```

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
        }
    }
}
```

Richiamo di una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
    }
}
}
```

Richiamo di una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

AWS SDK for .NET

Note

C'è altro su GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
        }
    }
}
```


Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.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; }
    }
}

```

Segnalazione degli errori degli elementi batch per le funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione degli errori degli elementi batch di DynamoDB con Lambda utilizzando .NET.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// 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 AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with MediaConvert.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve MediaConvert

Il seguente esempio di codice mostra come iniziare a utilizzare AWS Elemental MediaConvert.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 Jobs
are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get some MediaConvert jobs.
        var response = await mediaConvertClient.ListJobsAsync(
            new ListJobsRequest()
            {
                MaxResults = 10
            }
        );

        foreach (var job in response.Jobs)
        {
            Console.WriteLine($"\\tJob: {job.Id} status {job.Status}");
            Console.WriteLine();
        }
    }
}
```

- Per i dettagli sull'API, [DescribeEndpoints](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)

Azioni

CreateJob

Il seguente esempio di codice mostra come utilizzare `CreateJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta le posizioni dei file, il client e il wrapper.

```
// 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"];

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

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));

```

Crea il lavoro utilizzando il metodo wrapper e restituisci l'ID del lavoro.

```

/// <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

CreateJobResponse createJobResponse =
    await _amazonMediaConvert.CreateJobAsync(createJobRequest);

var jobId = createJobResponse.Job.Id;

return jobId;
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [CreateJob](#) Reference.

GetJob

Il seguente esempio di codice mostra come utilizzare `GetJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta le posizioni dei file, il client e il wrapper.

```
// 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"];

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

var wrapper = new MediaConvertWrapper(mcClient);
```

Ottieni un lavoro tramite il suo ID.

```
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;
}
```

- Per i dettagli sull'API, consulta la [GetJob](#) sezione AWS SDK for .NET API Reference.

ListJobs

Il seguente esempio di codice mostra come utilizzare `ListJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta le posizioni dei file, il client e il wrapper.

```
// 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"];

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

var wrapper = new MediaConvertWrapper(mcClient);

```

Elenca i lavori con uno stato particolare.

```

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}.");
});

```

Elenca i lavori usando un impaginatore.

```

/// <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;
}

```



```
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [ListJobsReference](#).

Organizations: esempi che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Organizations.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

AttachPolicy

Il seguente esempio di codice mostra come utilizzare `AttachPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AttachPolicy](#) consulta AWS SDK for .NET API Reference.

CreateAccount

Il seguente esempio di codice mostra come utilizzare `CreateAccount`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}  
}
```

- Per i dettagli sull'API, [CreateAccount](#) consulta AWS SDK for .NET API Reference.

CreateOrganization

Il seguente esempio di codice mostra come utilizzare `CreateOrganization`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateOrganization](#) consulta AWS SDK for .NET API Reference.

CreateOrganizationalUnit

Il seguente esempio di codice mostra come utilizzare `CreateOrganizationalUnit`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateOrganizationalUnit](#) consulta AWS SDK for .NET API Reference.

CreatePolicy

Il seguente esempio di codice mostra come utilizzare `CreatePolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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);
        }
    }
}
```

- Per i dettagli sull'API, [CreatePolicy](#) consulta AWS SDK for .NET API Reference.

DeleteOrganization

Il seguente esempio di codice mostra come utilizzare `DeleteOrganization`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteOrganization](#) consulta AWS SDK for .NET API Reference.

DeleteOrganizationalUnit

Il seguente esempio di codice mostra come utilizzare `DeleteOrganizationalUnit`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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-0000-000000000";

        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}.");
        }
    }
}
```

- Per i dettagli sull'API, [DeleteOrganizationalUnit](#) consulta AWS SDK for .NET API Reference.

DeletePolicy

Il seguente esempio di codice mostra come utilizzare `DeletePolicy`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeletePolicy](#) consulta AWS SDK for .NET API Reference.

DetachPolicy

Il seguente esempio di codice mostra come utilizzare `DetachPolicy`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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-000000000";
    }
}
```

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

- Per i dettagli sull'API, [DetachPolicy](#) consulta AWS SDK for .NET API Reference.

ListAccounts

Il seguente esempio di codice mostra come utilizzare `ListAccounts`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

- Per i dettagli sull'API, [ListAccounts](#) consulta AWS SDK for .NET API Reference.

ListOrganizationalUnitsForParent

Il seguente esempio di codice mostra come utilizzare `ListOrganizationalUnitsForParent`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

- Per i dettagli sull'API, [ListOrganizationalUnitsForParent](#) consulta AWS SDK for .NET API Reference.

ListPolicies

Il seguente esempio di codice mostra come utilizzare `ListPolicies`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

```
}
```

- Per i dettagli sull'API, [ListPolicies](#) consulta AWS SDK for .NET API Reference.

Esempi di utilizzo di Amazon Pinpoint AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Pinpoint. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

SendMessage

Il seguente esempio di codice mostra come utilizzare `SendMessage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio e-mail.

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendEmailMessage;

public class SendEmailMainClass
{
    public static async Task Main(string[] args)
    {
        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();

        // The AWS Region that you want to use to send the email. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The "From" address. This address has to be verified in Amazon Pinpoint
        // in the region you're using to send email.
        string senderAddress = configuration["SenderAddress"]!;

        // The address on the "To" line. If your Amazon Pinpoint account is in
        // the sandbox, this address also has to be verified.
        string toAddress = configuration["ToAddress"]!;

        // The Amazon Pinpoint project/application ID to use when you send this
        message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        try
        {
            await SendEmailMessage(region, appId, toAddress, senderAddress);
        }
        catch (Exception ex)
```

```

    {
        Console.WriteLine("The message wasn't sent. Error message: " +
ex.Message);
    }
}

public static async Task<MessageResponse> SendEmailMessage(
    string region, string appId, string toAddress, string senderAddress)
{
    var client = new
AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

    // The subject line of the email.
    string subject = "Amazon Pinpoint Email test";

    // The body of the email for recipients whose email clients don't
    // support HTML content.
    string textBody = @"Amazon Pinpoint Email Test (.NET)"
        + "\n-----"
        + "\nThis email was sent using the Amazon Pinpoint API
using the AWS SDK for .NET.";

    // The body of the email for recipients whose email clients support
    // HTML content.
    string htmlBody = @"<html>"
        + "\n<head></head>"
        + "\n<body>"
        + "\n  <h1>Amazon Pinpoint Email Test (AWS SDK for .NET)</
h1>"
        + "\n  <p>This email was sent using the "
        + "\n    <a href='https://aws.amazon.com/pinpoint/'>Amazon
Pinpoint</a> API "
        + "\n    using the <a href='https://aws.amazon.com/sdk-
for-net/'>AWS SDK for .NET</a>"
        + "\n  </p>"
        + "\n</body>"
        + "\n</html>";

    // The character encoding the you want to use for the subject line and
    // message body of the email.
    string charset = "UTF-8";

    var sendRequest = new SendMessagesRequest
    {

```

```
ApplicationId = appId,
MessageRequest = new MessageRequest
{
    Addresses = new Dictionary<string, AddressConfiguration>
    {
        {
            toAddress,
            new AddressConfiguration
            {
                ChannelType = ChannelType.EMAIL
            }
        },
    },
    MessageConfiguration = new DirectMessageConfiguration
    {
        EmailMessage = new EmailMessage
        {
            FromAddress = senderAddress,
            SimpleEmail = new SimpleEmail
            {
                HtmlPart = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = htmlBody
                },
                TextPart = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = textBody
                },
                Subject = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = subject
                }
            }
        }
    }
};
Console.WriteLine("Sending message...");
SendMessageResponse response = await client.SendMessageAsync(sendRequest);
Console.WriteLine("Message sent!");
return response.MessageResponse;
```

```
}  
}
```

Invia un messaggio SMS.

```
using Amazon;  
using Amazon.Pinpoint;  
using Amazon.Pinpoint.Model;  
using Microsoft.Extensions.Configuration;  
  
namespace SendSmsMessage;  
  
public class SendSmsMessageMainClass  
{  
    public static async Task Main(string[] args)  
    {  
        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();  
  
        // The AWS Region that you want to use to send the message. For a list of  
        // AWS Regions where the Amazon Pinpoint API is available, see  
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/  
        string region = "us-east-1";  
  
        // The phone number or short code to send the message from. The phone number  
        // or short code that you specify has to be associated with your Amazon  
        Pinpoint  
        // account. For best results, specify long codes in E.164 format.  
        string originationNumber = configuration["OriginationNumber"]!;  
  
        // The recipient's phone number. For best results, you should specify the  
        // phone number in E.164 format.  
        string destinationNumber = configuration["DestinationNumber"]!;  
  
        // The Pinpoint project/ application ID to use when you send this message.  
        // Make sure that the SMS channel is enabled for the project or application
```

```
// that you choose.
string appId = configuration["AppId"]!;

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
MessageType messageType = MessageType.TRANSACTIONAL;

// The registered keyword associated with the originating short code.
string? registeredKeyword = configuration["RegisteredKeyword"];

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-
countries.html
string? senderId = configuration["SenderId"];

try
{
    var response = await SendSmsMessage(region, appId, destinationNumber,
        originationNumber, registeredKeyword, senderId, messageType);
    Console.WriteLine($"Message sent to
{response.MessageResponse.Result.Count} recipient(s).");
    foreach (var messageResultValue in
        response.MessageResponse.Result.Select(r => r.Value))
    {
        Console.WriteLine($"{messageResultValue.MessageId} Status:
{messageResultValue.DeliveryStatus}");
    }
}
catch (Exception ex)
{
    Console.WriteLine("The message wasn't sent. Error message: " +
ex.Message);
}
}

public static async Task<SendMessagesResponse> SendSmsMessage(
    string region, string appId, string destinationNumber, string
originationNumber,
    string? keyword, string? senderId, MessageType messageType)
{
    // The content of the SMS message.
```



```
string message = "This message was sent through Amazon Pinpoint using" +
    " the AWS SDK for .NET. Reply STOP to opt out.";

var client = new
AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

SendMessagesRequest sendRequest = new SendMessagesRequest
{
    ApplicationId = appId,
    MessageRequest = new MessageRequest
    {
        Addresses =
            new Dictionary<string, AddressConfiguration>
            {
                {
                    destinationNumber,
                    new AddressConfiguration { ChannelType =
ChannelType.SMS }
                }
            },
        MessageConfiguration = new DirectMessageConfiguration
        {
            SMSMessage = new SMSMessage
            {
                Body = message,
                MessageType = MessageType.TRANSACTIONAL,
                OriginationNumber = originationNumber,
                SenderId = senderId,
                Keyword = keyword
            }
        }
    }
};
SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
return response;
}
```

- Per i dettagli sull'API, [SendMessages](#) consulta AWS SDK for .NET API Reference.

Esempi di utilizzo di Amazon Polly AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Polly. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

DeleteLexicon

Il seguente esempio di codice mostra come utilizzare DeleteLexicon.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [DeleteLexicon](#) consulta AWS SDK for .NET API Reference.

DescribeVoices

Il seguente esempio di codice mostra come utilizzare `DescribeVoices`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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}");
}
}

```

- Per i dettagli sull'API, [DescribeVoices](#) consulta AWS SDK for .NET API Reference.

GetLexicon

Il seguente esempio di codice mostra come utilizzare `GetLexicon`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);  
        }  
    }  
}
```

- Per i dettagli sull'API, [GetLexicon](#) consulta AWS SDK for .NET API Reference.

ListLexicons

Il seguente esempio di codice mostra come utilizzare `ListLexicons`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListLexicons](#) consulta AWS SDK for .NET API Reference.

PutLexicon

Il seguente esempio di codice mostra come utilizzare `PutLexicon`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
}
```

- Per i dettagli sull'API, [PutLexicon](#) consulta AWS SDK for .NET API Reference.

SynthesizeSpeech

Il seguente esempio di codice mostra come utilizzare `SynthesizeSpeech`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

Sintetizza il parlato dal testo utilizzando i segni vocali con Amazon Polly utilizzando un SDK. AWS

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

- Per i dettagli sulle API, consulta [SynthesizeSpeech](#) la sezione API Reference.AWS SDK for .NET

Esempi di utilizzo di Amazon RDS AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon RDS. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.


Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon RDS

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon RDS.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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($"DB name: {instance.DBName}");
    Console.WriteLine($"DB Instance Arn: {instance.DBInstanceArn}");
    Console.WriteLine($"DB Instance Identifier: {instance.DBInstanceIdentifier}");
    Console.WriteLine();
}
}
```

- Per informazioni dettagliate sull'API, consulta [DescribeDBInstances](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateDBInstance

Il seguente esempio di codice mostra come usare `CreateDBInstance`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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


- Per informazioni dettagliate sull'API, consulta [CreateDBInstance](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

CreateDBParameterGroup

Il seguente esempio di codice mostra come usare `CreateDBParameterGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [CreateDB ParameterGroup](#) in AWS SDK for .NET API Reference.

CreateDBSnapshot

Il seguente esempio di codice mostra come utilizzare CreateDBSnapshot

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [CreateDBSnapshot](#) nella Documentazione di riferimento dell'API AWS SDK for .NET .

DeleteDBInstance

Il seguente esempio di codice mostra come usare `DeleteDBInstance`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DeleteDBInstance](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DeleteDBParameterGroup

Il seguente esempio di codice mostra come usare `DeleteDBParameterGroup`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, consulta [DeleteDB ParameterGroup](#) in AWS SDK for .NET API Reference.

DescribeDBEngineVersions

Il seguente esempio di codice mostra come utilizzare `DescribeDBEngineVersions`

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeDB EngineVersions](#) in AWS SDK for .NET API Reference.

DescribeDBInstances

Il seguente esempio di codice mostra come utilizzare. DescribeDBInstances

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per informazioni dettagliate sull'API, consulta [DescribeDBInstances](#) nella Documentazione di riferimento delle API di AWS SDK for .NET .

DescribeDBParameterGroups

Il seguente esempio di codice mostra come usare `DescribeDBParameterGroups`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, consulta [DescribeDB ParameterGroups](#) in AWS SDK for .NET API Reference.

DescribeDBParameters

Il seguente esempio di codice mostra come utilizzare `DescribeDBParameters`

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni sull'API, consulta [DescribeDBParameters](#) nella Documentazione di riferimento dell'API AWS SDK for .NET .

DescribeDBSnapshots

Il seguente esempio di codice mostra come usare `DescribeDBSnapshots`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta [DescribeDBSnapshots](#) nella Documentazione di riferimento dell'API AWS SDK for .NET .

DescribeOrderableDBInstanceOptions

Il seguente esempio di codice mostra come usare `DescribeOrderableDBInstanceOptions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [DescribeOrderableDB InstanceOptions](#) in AWS SDK for .NET API Reference.

ModifyDBParameterGroup

Il seguente esempio di codice mostra come utilizzare `ModifyDBParameterGroup`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta [ModifyDB ParameterGroup](#) in AWS SDK for .NET API Reference.

Scenari

Nozioni di base sulle istanze DB

L'esempio di codice seguente mostra come:

- Creare un gruppo di parametri database personalizzati e imposta i relativi valori.
- Creare un'istanza database configurata per utilizzare il gruppo di parametri. L'istanza DB contiene anche un database.
- Acquisire uno snapshot dell'istanza.
- Eliminare l'istanza e il gruppo di parametri.

AWS SDK for .NET

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
/// <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}. 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.Description}." +
            $"{p.AllowedValues}." +
            $"{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(
        $"{instance.Engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
"
        + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");

    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);
    }

```

Metodi wrapper utilizzati dallo scenario per operazioni delle istanze DB.

```

    /// <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;
}

```

Metodi wrapper utilizzati dallo scenario per gruppi di parametri database.

```

/// <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;
    }
}
```

Metodi wrapper utilizzati dallo scenario per operazioni degli snapshot database.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateDBInstance](#)
 - [Creato B ParameterGroup](#)
 - [CreateDBSnapshot](#)
 - [DeleteDBInstance](#)
 - [Eliminare DB ParameterGroup](#)
 - [Descritto B EngineVersions](#)
 - [DescribeDBInstances](#)
 - [Descritto B ParameterGroups](#)
 - [DescribeDBParameters](#)
 - [DescribeDBSnapshots](#)
 - [DescribeOrderableDB InstanceOptions](#)
 - [Modifica DB ParameterGroup](#)

Esempi di utilizzo di Amazon Rekognition AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon AWS SDK for .NET Rekognition.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

CompareFaces

Il seguente esempio di codice mostra come utilizzare `CompareFaces`.

Per ulteriori informazioni, consulta [Confronto dei volti nelle immagini](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CompareFaces](#) consulta AWS SDK for .NET API Reference.

CreateCollection

Il seguente esempio di codice mostra come utilizzare `CreateCollection`.

Per ulteriori informazioni, consulta [Creazione di una raccolta](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateCollection](#) consulta AWS SDK for .NET API Reference.

DeleteCollection

Il seguente esempio di codice mostra come utilizzare `DeleteCollection`.

Per ulteriori informazioni, consulta [Eliminazione di una raccolta](#).

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteCollection](#) consulta AWS SDK for .NET API Reference.

DeleteFaces

Il seguente esempio di codice mostra come utilizzare `DeleteFaces`.

Per ulteriori informazioni, consulta [Eliminazione dei volti da una raccolta](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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-xxxxxxxxxxxxxxxx" };

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

- Per i dettagli sull'API, [DeleteFaces](#) consulta AWS SDK for .NET API Reference.

DescribeCollection

Il seguente esempio di codice mostra come utilizzare `DescribeCollection`.

Per ulteriori informazioni, consulta [Descrizione di una raccolta](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeCollection](#) consulta AWS SDK for .NET API Reference.

DetectFaces

Il seguente esempio di codice mostra come utilizzare DetectFaces.

Per ulteriori informazioni, consulta [Rilevamento dei volti in un'immagine](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
}

```

Visualizza le informazioni del riquadro di delimitazione per tutti i volti di un'immagine.

```

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}");
}
}
```

- Per i dettagli sull'API, [DetectFaces](#) consulta AWS SDK for .NET API Reference.

DetectLabels

Il seguente esempio di codice mostra come utilizzare `DetectLabels`.

Per ulteriori informazioni, consulta [Rilevamento delle etichette in un'immagine](#).

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
}
```

Rileva le etichette in un file di immagine archiviato sul tuo computer.

```
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);
    }
}
}
```

- Per i dettagli sull'API, [DetectLabels](#) consulta AWS SDK for .NET API Reference.

DetectModerationLabels

Il seguente esempio di codice mostra come utilizzare `DetectModerationLabels`.

Per ulteriori informazioni, consulta [Rilevamento di immagini non appropriate](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
}
```

- Per i dettagli sull'API, [DetectModerationLabels](#) consulta AWS SDK for .NET API Reference.

DetectText

Il seguente esempio di codice mostra come utilizzare `DetectText`.

Per ulteriori informazioni, consulta [Rilevamento del testo in un'immagine](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
        }
    }
}
```

```
}  
}
```

- Per i dettagli sull'API, [DetectText](#) consulta AWS SDK for .NET API Reference.

GetCelebrityInfo

Il seguente esempio di codice mostra come utilizzare `GetCelebrityInfo`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetCelebrityInfo](#) consulta AWS SDK for .NET API Reference.

IndexFaces

Il seguente esempio di codice mostra come utilizzare `IndexFaces`.

Per ulteriori informazioni, consulta [Indicizzazione dei volti in una raccolta](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [IndexFaces](#) consulta AWS SDK for .NET API Reference.

ListCollections

Il seguente esempio di codice mostra come utilizzare `ListCollections`.

Per ulteriori informazioni, consulta [Creazione dell'elenco delle raccolte](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
}
```

- Per i dettagli sull'API, [ListCollections](#) consulta AWS SDK for .NET API Reference.

ListFaces

Il seguente esempio di codice mostra come utilizzare `ListFaces`.

Per ulteriori informazioni, consulta [Creazione dell'elenco dei volti in una raccolta](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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));
    }
}
```

- Per i dettagli sull'API, [ListFaces](#) consulta AWS SDK for .NET API Reference.

RecognizeCelebrities

Il seguente esempio di codice mostra come utilizzare `RecognizeCelebrities`.

Per ulteriori informazioni, consulta [Riconoscimento delle celebrità in un'immagine](#).

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.UrlsWithEach(url =>
    {
        Console.WriteLine(url);
    });
});

Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
}
```

- Per i dettagli sull'API, [RecognizeCelebrities](#) consulta AWS SDK for .NET API Reference.

SearchFaces

Il seguente esempio di codice mostra come utilizzare `SearchFaces`.

Per ulteriori informazioni, consulta [Ricerca di un volto \(ID volto\)](#).

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [SearchFaces](#) consulta AWS SDK for .NET API Reference.

SearchFacesByImage

Il seguente esempio di codice mostra come utilizzare `SearchFacesByImage`.

Per ulteriori informazioni, consulta [Ricerca di un volto \(immagine\)](#).

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [SearchFacesByImage](#) consulta AWS SDK for .NET API Reference.

Esempi di registrazione del dominio Route 53 utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la registrazione del dominio AWS SDK for .NET with Route 53.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Registrazione domini Hello Route 53

Gli esempi di codice seguenti mostrano come iniziare a usare la registrazione di domini Route 53.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListPrices](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CheckDomainAvailability

Il seguente esempio di codice mostra come utilizzare `CheckDomainAvailability`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CheckDomainAvailability](#) consulta AWS SDK for .NET API Reference.

CheckDomainTransferability

Il seguente esempio di codice mostra come utilizzare `CheckDomainTransferability`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [CheckDomainTransferability](#) consulta AWS SDK for .NET API Reference.

GetDomainDetail

Il seguente esempio di codice mostra come utilizzare `GetDomainDetail`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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.";
    }
}
```

- Per i dettagli sull'API, [GetDomainDetail](#) consulta AWS SDK for .NET API Reference.

GetDomainSuggestions

Il seguente esempio di codice mostra come utilizzare `GetDomainSuggestions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetDomainSuggestions](#) consulta AWS SDK for .NET API Reference.

GetOperationDetail

Il seguente esempio di codice mostra come utilizzare `GetOperationDetail`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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}.";
}
}

```

- Per i dettagli sull'API, [GetOperationDetail](#) consulta AWS SDK for .NET API Reference.

ListDomains

Il seguente esempio di codice mostra come utilizzare `ListDomains`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, [ListDomains](#) consulta AWS SDK for .NET API Reference.

ListOperations

Il seguente esempio di codice mostra come utilizzare `ListOperations`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListOperations](#) consulta AWS SDK for .NET API Reference.

ListPrices

Il seguente esempio di codice mostra come utilizzare `ListPrices`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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();
}
```

- Per i dettagli sull'API, [ListPrices](#) consulta AWS SDK for .NET API Reference.

RegisterDomain

Il seguente esempio di codice mostra come utilizzare `RegisterDomain`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [RegisterDomain](#) consulta AWS SDK for .NET API Reference.

ViewBilling

Il seguente esempio di codice mostra come utilizzare `ViewBilling`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ViewBilling](#) consulta AWS SDK for .NET API Reference.

Scenari

Nozioni di base sui domini

L'esempio di codice seguente mostra come:

- Elenca i domini correnti ed elenca le operazioni dell'anno scorso.
- Visualizza la fatturazione dell'anno scorso e visualizza i prezzi per i tipi di dominio.
- Ricevi suggerimenti sui domini.
- Verifica la disponibilità e la trasferibilità dei domini.
- Facoltativamente, richiedi la registrazione di un dominio.
- Ottieni informazioni dettagliate di un'operazione.
- Facoltativamente, ottieni informazioni dettagliate di un dominio.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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($"{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($"2. List account domain operations in the past year.");
    var operations = await _route53Wrapper.ListOperations(
        DateTime.Today.AddYears(-1));
    for (int i = 0; i < operations.Count; i++)
    {
        Console.WriteLine($"{i}\tOperation Id: {operations[i].OperationId}");
        Console.WriteLine($"{i}\tStatus: {operations[i].Status}");
        Console.WriteLine($"{i}\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($"Transferability: {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($"Note: This example uses domain request settings in
settings.json.");
    Console.WriteLine($"To change the domain registration settings, set the
values in that file.");
    Console.WriteLine($"Remember, registering an actual domain will incur an
account billing cost.");
    Console.WriteLine($"Would 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(
                $"{Environment.NewLine}Registration 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;
    }
}

```

Metodi wrapper utilizzati dallo scenario per le operazioni di registrazione di domini Route 53.

```

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 = $"{\t}Operation {operationId}:\n" +
                $"{\t}For domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}. \n" +
                $"{\t}Message is {operationDetails.Message}. \n" +
                $"{\t}Status 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.";
    }
}
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CheckDomainAvailability](#)
 - [CheckDomainTransferability](#)
 - [GetDomainDetail](#)
 - [GetDomainSuggestions](#)

- [GetOperationDetail](#)
- [ListDomains](#)
- [ListOperations](#)
- [ListPrices](#)
- [RegisterDomain](#)
- [ViewBilling](#)

Esempi di utilizzo di Amazon S3 AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon S3. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti


- [Azioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Azioni

AbortMultipartUploads

Il seguente esempio di codice mostra come utilizzare `AbortMultipartUploads`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AbortMultipartUploads](#) consulta AWS SDK for .NET API Reference.

CopyObject

Il seguente esempio di codice mostra come utilizzare `CopyObject`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [CopyObject](#) consulta AWS SDK for .NET API Reference.

CreateBucket

Il seguente esempio di codice mostra come utilizzare `CreateBucket`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```

Crea un bucket con il blocco degli oggetti abilitato.

```

    /// <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;
}
}
```

- Per i dettagli sull'API, consulta la sezione [CreateBucket AWS SDK for .NET API Reference](#).

DeleteBucket

Il seguente esempio di codice mostra come utilizzare `DeleteBucket`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteBucket](#) consulta AWS SDK for .NET API Reference.

DeleteBucketCors

Il seguente esempio di codice mostra come utilizzare `DeleteBucketCors`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);  
}
```

- Per i dettagli sull'API, [DeleteBucketCors](#) consulta AWS SDK for .NET API Reference.

DeleteBucketLifecycle

Il seguente esempio di codice mostra come utilizzare `DeleteBucketLifecycle`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);  
}
```

- Per i dettagli sull'API, [DeleteBucketLifecycle](#) consulta AWS SDK for .NET API Reference.

DeleteObject

Il seguente esempio di codice mostra come utilizzare `DeleteObject`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina un oggetto in un bucket S3 senza controllo delle versioni.

```
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.");
        }
    }
}

```

Elimina un oggetto in un bucket S3 con controllo delle versioni.

```

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 = "versioned-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;
}
}

```

- Per i dettagli sull'API, [DeleteObject](#) consulta AWS SDK for .NET API Reference.

DeleteObjects

Il seguente esempio di codice mostra come utilizzare `DeleteObjects`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Eliminazione di tutti gli oggetti da un bucket Amazon S3.

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

Eliminare più oggetti da un bucket S3 senza controllo delle versioni.

```
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;
}
}
```

Eliminare più oggetti da un bucket S3 con controllo delle versioni.

```
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
    /// PubObjectsAsync 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;
    }
}

```

- Per i dettagli sull'API, consulta la [DeleteObjects](#) sezione AWS SDK for .NET API Reference.

GetBucketAcl

Il seguente esempio di codice mostra come utilizzare `GetBucketAcl`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    /// <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;
    }
```

- Per i dettagli sull'API, consulta la [GetBucketAcl](#) sezione AWS SDK for .NET API Reference.

GetBucketCors

Il seguente esempio di codice mostra come utilizzare `GetBucketCors`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
    PrintCORSCONFIGURATIONRules(configuration);
    return configuration;
}
```

- Per i dettagli sull'API, consulta la [GetBucketCors](#) sezione AWS SDK for .NET API Reference.

GetBucketLifecycleConfiguration

Il seguente esempio di codice mostra come utilizzare `GetBucketLifecycleConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [GetBucketLifecycleConfiguration](#) sezione AWS SDK for .NET API Reference.

GetBucketWebsite

Il seguente esempio di codice mostra come utilizzare `GetBucketWebsite`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [GetBucketWebsite](#) sezione AWS SDK for .NET API Reference.

GetObject

Il seguente esempio di codice mostra come utilizzare `GetObject`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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($"{filePath}\
\{objectName}", true, CancellationTokens.None);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error saving {objectName}: {ex.Message}");
        return false;
    }
}
```

- Per i dettagli sull'API, consulta la [GetObject](#) sezione AWS SDK for .NET API Reference.

GetObjectLegalHold

Il seguente esempio di codice mostra come utilizzare `GetObjectLegalHold`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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($"Unable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}

```

- Per i dettagli sull'API, consulta la [GetObjectLegalHold](#) sezione AWS SDK for .NET API Reference.

GetObjectLockConfiguration

Il seguente esempio di codice mostra come utilizzare `GetObjectLockConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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>
GetObjectLockConfiguration(string bucketName)
{
    try
    {
        var request = new GetObjectLockConfigurationRequest()
        {
            BucketName = bucketName
        };

        var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
        Console.WriteLine($"Bucket 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();
    }
}

```

- Per i dettagli sull'API, consulta la [GetObjectLockConfiguration](#) sezione AWS SDK for .NET API Reference.

GetObjectRetention

Il seguente esempio di codice mostra come utilizzare `GetObjectRetention`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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($"Object retention for {objectKey} in {bucketName}:
" +
            $"{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}");
        return response.Retention;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Unable to fetch object lock retention:
'{ex.Message}'");
        return new ObjectLockRetention();
    }
}
```

- Per i dettagli sull'API, consulta la [GetObjectRetention](#) sezione AWS SDK for .NET API Reference.

ListBuckets

Il seguente esempio di codice mostra come utilizzare `ListBuckets`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

```
}  
}
```

- Per i dettagli sull'API, consulta la [ListBuckets](#) sezione AWS SDK for .NET API Reference.

ListObjectVersions

Il seguente esempio di codice mostra come utilizzare `ListObjectVersions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [ListObjectVersions](#) sezione AWS SDK for .NET API Reference.

ListObjectsV2

Il seguente esempio di codice mostra come utilizzare `ListObjectsV2`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

Elencare gli oggetti con un impaginatore.

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

- Per i dettagli sull'API, consulta la [ListObjectsversione V2](#) in AWS SDK for .NET API Reference.

PutBucketAccelerateConfiguration

Il seguente esempio di codice mostra come utilizzare `PutBucketAccelerateConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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");
    }
}
```

```
    }
}
```

- Per i dettagli sull'API, consulta la [PutBucketAccelerateConfiguration](#) sezione AWS SDK for .NET API Reference.

PutBucketAcl

Il seguente esempio di codice mostra come utilizzare `PutBucketAcl`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    /// <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;
}
```

- Per i dettagli sull'API, consulta la [PutBucketAcl](#) sezione AWS SDK for .NET API Reference.

PutBucketCors

Il seguente esempio di codice mostra come utilizzare `PutBucketCors`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);
}
```

- Per i dettagli sull'API, consulta la [PutBucketCors](#) sezione AWS SDK for .NET API Reference.

PutBucketLifecycleConfiguration

Il seguente esempio di codice mostra come utilizzare `PutBucketLifecycleConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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);
}
```

- Per i dettagli sull'API, consulta la [PutBucketLifecycleConfiguration](#) sezione AWS SDK for .NET API Reference.

PutBucketLogging

Il seguente esempio di codice mostra come utilizzare PutBucketLogging.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}
}
```

- Per i dettagli sull'API, consulta la [PutBucketLogging](#) sezione AWS SDK for .NET API Reference.

PutBucketNotificationConfiguration

Il seguente esempio di codice mostra come utilizzare `PutBucketNotificationConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [PutBucketNotificationConfiguration](#) sezione AWS SDK for .NET API Reference.

PutBucketWebsite

Il seguente esempio di codice mostra come utilizzare `PutBucketWebsite`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// Put the website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
```

```

        {
            BucketName = bucketName,
            WebsiteConfiguration = new WebsiteConfiguration()
            {
                IndexDocumentSuffix = indexDocumentSuffix,
                ErrorDocument = errorDocument,
            },
        };
        PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);

```

- Per i dettagli sull'API, consulta la [PutBucketWebsite](#) sezione AWS SDK for .NET API Reference.

PutObject

Il seguente esempio di codice mostra come utilizzare `PutObject`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
        }
    }
}
```

Caricare un oggetto con la crittografia lato server.

```
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");
    }
}
}

```

- Per i dettagli sull'API, consulta la [PutObject](#) sezione AWS SDK for .NET API Reference.

PutObjectLegalHold

Il seguente esempio di codice mostra come utilizzare `PutObjectLegalHold`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }
}
```

- Per i dettagli sull'API, consulta la [PutObjectLegalHold](#) sezione AWS SDK for .NET API Reference.

PutObjectLockConfiguration

Il seguente esempio di codice mostra come utilizzare `PutObjectLockConfiguration`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta la configurazione di blocco degli oggetti di un bucket.

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

Imposta il periodo di conservazione predefinito di un bucket.

```

/// <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;
    }
}

```

- Per i dettagli sull'API, consulta la sezione [PutObjectLockConfiguration AWS SDK for .NET API Reference](#).

PutObjectRetention

Il seguente esempio di codice mostra come utilizzare `PutObjectRetention`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
}
```

- Per i dettagli sull'API, consulta la [PutObjectRetention](#) sezione AWS SDK for .NET API Reference.

RestoreObject

Il seguente esempio di codice mostra come utilizzare `RestoreObject`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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}");
}
}

```

- Per i dettagli sull'API, consulta la [RestoreObject](#) sezione AWS SDK for .NET API Reference.

Scenari

Creazione di un URL prefirmato

Il seguente esempio di codice mostra come creare un URL predefinito per Amazon S3 e caricare un oggetto.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Genera un URL prefirmato in grado di eseguire un'operazione Amazon S3 per un periodo di tempo limitato.

```
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;
    }
}
```

Genera un URL prefirmato ed esegui un caricamento utilizzando quell'URL.

```
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 name="objectKey">The name of the file that will be uploaded.</
param>
    /// <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;
    }
}

```

Nozioni di base su bucket e oggetti

L'esempio di codice seguente mostra come:

- Crea un bucket e carica un file in tale bucket.

- Scaricare un oggetto da un bucket.
- Copiare un oggetto in una sottocartella in un bucket.
- Elencare gli oggetti in un bucket.
- Elimina il bucket e tutti gli oggetti in esso contenuti.

AWS SDK for .NET

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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($"\\n{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.WriteLine("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.WriteLine("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);
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CopyObject](#)
 - [CreateBucket](#)

- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

Nozioni di base sulla crittografia

L'esempio di codice seguente mostra come iniziare a utilizzare la crittografia per gli oggetti Amazon S3.

AWS SDK for .NET

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}

```


- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .

- [CopyObject](#)
- [GetObject](#)
- [GetObjectMetadata](#)

Nozioni di base sui tag

L'esempio di codice seguente mostra come utilizzare i tag con gli oggetti Amazon S3.

AWS SDK for .NET

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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



```

    }
  }
}

```

- Per i dettagli sull'API, consulta la [GetObjectTagging](#) sezione AWS SDK for .NET API Reference.

Ottieni la configurazione di conservazione legale di un oggetto

Il seguente esempio di codice mostra come ottenere la configurazione di conservazione legale di un bucket S3.

AWS SDK for .NET

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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($"Unable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}
```

- Per i dettagli sull'API, consulta la [GetObjectLegalHold](#) sezione AWS SDK for .NET API Reference.

Blocca oggetti Amazon S3

Il seguente esempio di codice mostra come utilizzare le funzionalità di blocco degli oggetti di S3.

AWS SDK for .NET

Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo che dimostri le funzionalità di blocco degli oggetti di Amazon S3.

```
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(
        "\nFor this workflow, we will use the AWS SDK for .NET to create several
S3\n" +
        "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.On);

                            }
                            break;
                        }
                    case 1:
                        {
                            var question =
                                $"{\nWould 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;
}
}
```

Una classe wrapper per le funzioni S3.

```
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(
            $"Deleted {objectKey} in {bucketName}.");
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable 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($"\\tDelete for {bucketName} complete.");
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to delete bucket {bucketName}: " +
ex.Message);
        return false;
    }
}
}

```


- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [GetObjectLegalHold](#)

- [GetObjectLockConfiguration](#)
- [GetObjectRetention](#)
- [PutObjectLegalHold](#)
- [PutObjectLockConfiguration](#)
- [PutObjectRetention](#)

Gestire le liste di controllo degli accessi (ACL)

Gli esempi di codice seguenti mostrano come gestire le liste di controllo degli accessi (ACL) per i bucket Amazon S3.

AWS SDK for .NET

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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,
    AccessControlList = newAcl,
});
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [GetBucketAcl](#)
 - [GetObjectAcl](#)
 - [PutBucketAcl](#)
 - [PutObjectAcl](#)

Eseguire una copia in più parti

L'esempio di codice seguente mostra come eseguire una copia in più parti di un oggetto Amazon S3.

AWS SDK for .NET

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CompleteMultipartUpload](#)
 - [CreateMultipartUpload](#)
 - [GetObjectMetadata](#)
 - [UploadPartCopy](#)

Caricamento o download di file di grandi dimensioni

Il seguente esempio di codice mostra come caricare o scaricare file di grandi dimensioni da e verso Amazon S3.

Per ulteriori informazioni, consulta [Caricamento di un oggetto utilizzando il caricamento in più parti](#).

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Chiama le funzioni che trasferiscono file da e verso un bucket S3 utilizzando Amazon S3.

TransferUtility

```
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);
}

```

Caricamento di un singolo file.

```

/// <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;
        }
    }
}
```

Caricamento di un'intera directory locale.

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

Download di un singolo file.

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

Download dei contenuti di un bucket S3.

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

Tieni traccia dello stato di avanzamento di un caricamento utilizzando `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}");
}
}
```

Carica un oggetto con la crittografia.

```
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(initWithRequest);

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);
    }
}
}
```

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

AWS SDK for .NET

 Note

C'è altro su GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
    }
}
}
```

Esempi di utilizzo di S3 Glacier AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando S3 Glacier AWS SDK for .NET .

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon S3 Glacier

L'esempio di codice seguente mostra come iniziare a utilizzare Amazon S3 Glacier.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [ListVaults](#) sezione AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)

Azioni

AddTagsToVault

Il seguente esempio di codice mostra come utilizzare `AddTagsToVault`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [AddTagsToVault](#) sezione AWS SDK for .NET API Reference.

CreateVault

Il seguente esempio di codice mostra come utilizzare `CreateVault`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [CreateVault](#) sezione AWS SDK for .NET API Reference.

DescribeVault

Il seguente esempio di codice mostra come utilizzare `DescribeVault`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [DescribeVault](#) sezione AWS SDK for .NET API Reference.

InitiateJob

Il seguente esempio di codice mostra come utilizzare `InitiateJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Recupera un archivio da un caveau. Questo esempio utilizza la classe `ArchiveTransferManager`. Per i dettagli sull'API, vedere [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}%");
    }
}
```

- Per i dettagli sulle API, consulta la [InitiateJob](#) sezione AWS SDK for .NET API Reference.

ListJobs

Il seguente esempio di codice mostra come utilizzare `ListJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, consulta la [ListJobs](#) sezione AWS SDK for .NET API Reference.

ListTagsForVault

Il seguente esempio di codice mostra come utilizzare `ListTagsForVault`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
```

- Per i dettagli sull'API, consulta la [ListTagsForVault](#) sezione AWS SDK for .NET API Reference.

ListVaults

Il seguente esempio di codice mostra come utilizzare `ListVaults`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, consulta la [ListVaults](#) sezione AWS SDK for .NET API Reference.

UploadArchive

Il seguente esempio di codice mostra come utilizzare `UploadArchive`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }
}
```

- Per i dettagli sull'API, consulta la [UploadArchive](#) sezione AWS SDK for .NET API Reference.

SageMaker esempi che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with SageMaker.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve SageMaker

I seguenti esempi di codice mostrano come iniziare a utilizzare SageMaker.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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($"\\tInstance:
{notebookInstance.NotebookInstanceName}");
        Console.WriteLine($"\\tArn: {notebookInstance.NotebookInstanceArn}");
        Console.WriteLine($"\\tCreation Date:
{notebookInstance.CreationTime.ToShortDateString()}");
        Console.WriteLine();
    }
}
}
```

- Per i dettagli sull'API, consulta la [ListNotebookInstances](#) sezione AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreatePipeline

Il seguente esempio di codice mostra come utilizzare `CreatePipeline`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [CreatePipeline](#) sezione AWS SDK for .NET API Reference.

DeletePipeline

Il seguente esempio di codice mostra come utilizzare `DeletePipeline`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [DeletePipeline](#) sezione AWS SDK for .NET API Reference.

DescribePipelineExecution

Il seguente esempio di codice mostra come utilizzare `DescribePipelineExecution`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [DescribePipelineExecution](#) sezione AWS SDK for .NET API Reference.

StartPipelineExecution

Il seguente esempio di codice mostra come utilizzare `StartPipelineExecution`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, consulta la [StartPipelineExecution](#) sezione AWS SDK for .NET API Reference.

UpdatePipeline

Il seguente esempio di codice mostra come utilizzare `UpdatePipeline`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}
}
```

- Per i dettagli sull'API, consulta la [UpdatePipeline](#) sezione AWS SDK for .NET API Reference.

Scenari

Inizia con i lavori e le pipeline geospaziali

L'esempio di codice seguente mostra come:

- Imposta le risorse per una pipeline.
- Configura una pipeline che esegua un lavoro geospaziale.
- Avvio dell'esecuzione di una pipeline.
- Monitora lo stato dell'esecuzione.
- Visualizza l'output della pipeline.
- Pulisci le risorse.

Per ulteriori informazioni, consulta [Creare ed eseguire SageMaker pipeline utilizzando gli AWS SDK su Community.aws](#).

AWS SDK for .NET

Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una classe che racchiuda le operazioni. SageMaker

```
using System.Text.Json;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

namespace SageMakerActions;

/// <summary>
/// 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;
    }
}

```

Crea una funzione che gestisca i callback dalla pipeline. SageMaker

```

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.");
    }
}
}
}
}
}
}

```

Esegui uno scenario interattivo al prompt dei comandi.

```

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($"\\tRole 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($"Pipeline 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 = lambdaRoleName
        });
        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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Esempi di Secrets Manager che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Secrets Manager.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

GetSecretValue

Il seguente esempio di codice mostra come utilizzare `GetSecretValue`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
}
```

- Per i dettagli sull'API, consulta la [GetSecretValue](#) sezione AWS SDK for .NET API Reference.

Esempi di utilizzo di Amazon SES AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Amazon SES.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti


- [Azioni](#)

Azioni

CreateTemplate

Il seguente esempio di codice mostra come utilizzare `CreateTemplate`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [CreateTemplate](#) sezione AWS SDK for .NET API Reference.

DeleteIdentity

Il seguente esempio di codice mostra come utilizzare `DeleteIdentity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [DeleteIdentity](#) sezione AWS SDK for .NET API Reference.

DeleteTemplate

Il seguente esempio di codice mostra come utilizzare `DeleteTemplate`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [DeleteTemplate](#) sezione AWS SDK for .NET API Reference.

GetIdentityVerificationAttributes

Il seguente esempio di codice mostra come utilizzare `GetIdentityVerificationAttributes`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [GetIdentityVerificationAttributes](#) sezione AWS SDK for .NET API Reference.

GetSendQuota

Il seguente esempio di codice mostra come utilizzare `GetSendQuota`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [GetSendQuota](#) sezione AWS SDK for .NET API Reference.

ListIdentities

Il seguente esempio di codice mostra come utilizzare `ListIdentities`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [ListIdentities](#) sezione AWS SDK for .NET API Reference.

ListTemplates

Il seguente esempio di codice mostra come utilizzare `ListTemplates`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [ListTemplates](#) sezione AWS SDK for .NET API Reference.

SendEmail

Il seguente esempio di codice mostra come utilizzare `SendEmail`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [SendEmail](#) sezione AWS SDK for .NET API Reference.

SendTemplatedEmail

Il seguente esempio di codice mostra come utilizzare `SendTemplatedEmail`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [SendTemplatedEmail](#) sezione AWS SDK for .NET API Reference.

VerifyEmailIdentity

Il seguente esempio di codice mostra come utilizzare `VerifyEmailIdentity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [VerifyEmailIdentity](#) sezione AWS SDK for .NET API Reference.

Esempi di API Amazon SES v2 con AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'API AWS SDK for .NET with Amazon SES v2.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateContact

Il seguente esempio di codice mostra come utilizzare `CreateContact`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Creates a contact and adds it to the specified contact list.
/// </summary>
/// <param name="emailAddress">The email address of the contact.</param>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The response from the CreateContact operation.</returns>
public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
{
    var request = new CreateContactRequest
    {
        EmailAddress = emailAddress,
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
```

```
        Console.WriteLine($"Contact with email address {emailAddress} already  
exists in the contact list {contactListName}.");  
        Console.WriteLine(ex.Message);  
        return true;  
    }  
    catch (NotFoundException ex)  
    {  
        Console.WriteLine($"The contact list {contactListName} does not  
exist.");  
        Console.WriteLine(ex.Message);  
    }  
    catch (TooManyRequestsException ex)  
    {  
        Console.WriteLine("Too many requests were made. Please try again  
later.");  
        Console.WriteLine(ex.Message);  
    }  
    catch (Exception ex)  
    {  
        Console.WriteLine($"An error occurred while creating the contact:  
{ex.Message}");  
    }  
    return false;  
}
```

- Per i dettagli sull'API, consulta la [CreateContact](#) sezione AWS SDK for .NET API Reference.

CreateContactList

Il seguente esempio di codice mostra come utilizzare `CreateContactList`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
///  
/// <summary>
```

```
/// Creates a contact list with the specified name.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateContactListAsync(string contactListName)
{
    var request = new CreateContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Contact list with name {contactListName} already
exists.");
        Console.WriteLine(ex.Message);
        return true;
    }
    catch (LimitExceededException ex)
    {
        Console.WriteLine("The limit for contact lists has been exceeded.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
    }
    return false;
}
```

- Per i dettagli sull'API, consulta la [CreateContactList](#) sezione AWS SDK for .NET API Reference.

CreateEmailIdentity

Il seguente esempio di codice mostra come utilizzare `CreateEmailIdentity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Creates an email identity (email address or domain) and starts the
verification process.
/// </summary>
/// <param name="emailIdentity">The email address or domain to create and
verify.</param>
/// <returns>The response from the CreateEmailIdentity operation.</returns>
public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
{
    var request = new CreateEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.CreateEmailIdentityAsync(request);
        return response;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email identity {emailIdentity} already exists.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
}
```


```
        throw;
    }
    catch (LimitExceededException ex)
    {
        Console.WriteLine("The limit for email identities has been exceeded.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
        throw;
    }
}
```

- Per i dettagli sull'API, consulta la [CreateEmailIdentity](#) sezione AWS SDK for .NET API Reference.

CreateEmailTemplate

Il seguente esempio di codice mostra come utilizzare `CreateEmailTemplate`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Creates an email template with the specified content.
/// </summary>
/// <param name="templateName">The name of the email template.</param>
/// <param name="subject">The subject of the email template.</param>
/// <param name="htmlContent">The HTML content of the email template.</param>
/// <param name="textContent">The text content of the email template.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
{
    var request = new CreateEmailTemplateRequest
    {
        TemplateName = templateName,
        TemplateContent = new EmailTemplateContent
        {
            Subject = subject,
            Html = htmlContent,
            Text = textContent
        }
    };

    try
    {
        var response = await _sesClient.CreateEmailTemplateAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email template with name {templateName} already
exists.");
        Console.WriteLine(ex.Message);
    }
    catch (LimitExceededException ex)
```

```
    {
        Console.WriteLine("The limit for email templates has been exceeded.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email template:
{ex.Message}");
    }

    return false;
}
```

- Per i dettagli sull'API, consulta la [CreateEmailTemplate](#) sezione AWS SDK for .NET API Reference.

DeleteContactList

Il seguente esempio di codice mostra come utilizzare `DeleteContactList`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Deletes a contact list and all contacts within it.
/// </summary>
/// <param name="contactListName">The name of the contact list to delete.</
param>
/// <returns>True if successful.</returns>
```

```
public async Task<bool> DeleteContactListAsync(string contactListName)
{
    var request = new DeleteContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.DeleteContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
    }

    return false;
}
```

- Per i dettagli sull'API, consulta la [DeleteContactList](#) sezione AWS SDK for .NET API Reference.

DeleteEmailIdentity

Il seguente esempio di codice mostra come utilizzare `DeleteEmailIdentity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Deletes an email identity (email address or domain).
/// </summary>
/// <param name="emailIdentity">The email address or domain to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
{
    var request = new DeleteEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.DeleteEmailIdentityAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {

```

```
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
    }

    return false;
}
```

- Per i dettagli sull'API, consulta la [DeleteEmailIdentity](#) sezione AWS SDK for .NET API Reference.

DeleteEmailTemplate

Il seguente esempio di codice mostra come utilizzare `DeleteEmailTemplate`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Deletes an email template.
/// </summary>
/// <param name="templateName">The name of the email template to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailTemplateAsync(string templateName)
{
    var request = new DeleteEmailTemplateRequest
    {
        TemplateName = templateName
    };
};
```

```
try
{
    var response = await _sesClient.DeleteEmailTemplateAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
catch (NotFoundException ex)
{
    Console.WriteLine($"The email template {templateName} does not exist.");
    Console.WriteLine(ex.Message);
}
catch (TooManyRequestsException ex)
{
    Console.WriteLine("Too many requests were made. Please try again
later.");
    Console.WriteLine(ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
}

return false;
}
```

- Per i dettagli sull'API, consulta la [DeleteEmailTemplate](#) sezione AWS SDK for .NET API Reference.

ListContacts

Il seguente esempio di codice mostra come utilizzare `ListContacts`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
/// <summary>
/// Lists the contacts in the specified contact list.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The list of contacts response from the ListContacts operation.</
returns>
public async Task<List<Contact>> ListContactsAsync(string contactListName)
{
    var request = new ListContactsRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.ListContactsAsync(request);
        return response.Contacts;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
    }

    return new List<Contact>();
}
```

- Per i dettagli sull'API, consulta la [ListContacts](#) sezione AWS SDK for .NET API Reference.

SendEmail

Il seguente esempio di codice mostra come utilizzare `SendEmail`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Sends an email with the specified content and options.
/// </summary>
/// <param name="fromEmailAddress">The email address to send the email from.</
param>
/// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
/// <param name="subject">The subject of the email.</param>
/// <param name="htmlContent">The HTML content of the email.</param>
/// <param name="textContent">The text content of the email.</param>
/// <param name="templateName">The name of the email template to use
(optional).</param>
/// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
/// <param name="contactListName">The name of the contact list for unsubscribe
functionality (optional).</param>
/// <returns>The MessageId response from the SendEmail operation.</returns>
public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,
    string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
{
    var request = new SendEmailRequest
    {
        FromEmailAddress = fromEmailAddress
    };

    if (toEmailAddresses.Any())
    {
        request.Destination = new Destination { ToAddresses =
toEmailAddresses };
    }
}
```

```
    }

    if (!string.IsNullOrEmpty(templateName))
    {
        request.Content = new EmailContent()
        {
            Template = new Template
            {
                TemplateName = templateName,
                TemplateData = templateData
            }
        };
    }
    else
    {
        request.Content = new EmailContent
        {
            Simple = new Message
            {
                Subject = new Content { Data = subject },
                Body = new Body
                {
                    Html = new Content { Data = htmlContent },
                    Text = new Content { Data = textContent }
                }
            }
        };
    }

    if (!string.IsNullOrEmpty(contactListName))
    {
        request.ListManagementOptions = new ListManagementOptions
        {
            ContactListName = contactListName
        };
    }

    try
    {
        var response = await _sesClient.SendEmailAsync(request);
        return response.MessageId;
    }
    catch (AccountSuspendedException ex)
    {
```

```
        Console.WriteLine("The account's ability to send email has been
permanently restricted.");
        Console.WriteLine(ex.Message);
    }
    catch (MailFromDomainNotVerifiedException ex)
    {
        Console.WriteLine("The sending domain is not verified.");
        Console.WriteLine(ex.Message);
    }
    catch (MessageRejectedException ex)
    {
        Console.WriteLine("The message content is invalid.");
        Console.WriteLine(ex.Message);
    }
    catch (SendingPausedException ex)
    {
        Console.WriteLine("The account's ability to send email is currently
paused.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
    }

    return string.Empty;
}
```

- Per i dettagli sull'API, consulta la [SendEmail](#) sezione AWS SDK for .NET API Reference.

Scenari

Workflow della newsletter

Il seguente esempio di codice mostra come utilizzare il flusso di lavoro delle newsletter di Amazon SES API v2.

AWS SDK for .NET

Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui il flusso di lavoro.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace Sesev2Scenario;

public static class NewsletterWorkflow
{
    /*
        This workflow demonstrates how to use the Amazon Simple Email Service (SES) v2
        to send a coupon newsletter to a list of subscribers.
        The workflow performs the following tasks:

        1. Prepare the application:
            - Create a verified email identity for sending and replying to emails.
            - Create a contact list to store the subscribers' email addresses.
            - Create an email template for the coupon newsletter.

        2. Gather subscriber email addresses:
            - Prompt the user for a base email address.
```

- Create 3 variants of the email address using subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com).
- Add each variant as a contact to the contact list.
- Send a welcome email to each new contact.

3. Send the coupon newsletter:

- Retrieve the list of contacts from the contact list.
- Send the coupon newsletter using the email template to each contact.

4. Monitor and review:

- Provide instructions for the user to review the sending activity and metrics in the AWS console.

5. Clean up resources:

- Delete the contact list (which also deletes all contacts within it).
- Delete the email template.
- Optionally delete the verified email identity.

*/

```
public static SESv2Wrapper _sesv2Wrapper;
public static string? _baseEmailAddress = null;
public static string? _verifiedEmail = null;
private static string _contactListName = "weekly-coupons-newsletter";
private static string _templateName = "weekly-coupons";
private static string _subject = "Weekly Coupons Newsletter";
private static string _htmlContentFile = "coupon-newsletter.html";
private static string _textContentFile = "coupon-newsletter.txt";
private static string _htmlWelcomeFile = "welcome.html";
private static string _textWelcomeFile = "welcome.txt";
private static string _couponsDataFile = "sample_coupons.json";

// Relative location of the shared workflow resources folder.
private static string _resourcesFilePathLocation = "../..../..../..../..../
workflows/sesv2_weekly_mailer/resources/";

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))
        .Build();
}
```

```
        .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonSimpleEmailServiceV2>()
            .AddTransient<SESV2Wrapper>()
    )
    .Build();

ServicesSetup(host);

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Amazon SES v2 Coupon Newsletter
Workflow.");
    Console.WriteLine("This workflow demonstrates how to use the Amazon
Simple Email Service (SES) v2 " +
        "\r\n" + "to send a coupon newsletter to a list of
subscribers.");

    // Prepare the application.
    var emailIdentity = await PrepareApplication();

    // Gather subscriber email addresses.
    await GatherSubscriberEmailAddresses(emailIdentity);

    // Send the coupon newsletter.
    await SendCouponNewsletter(emailIdentity);

    // Monitor and review.
    MonitorAndReview(true);

    // Clean up resources.
    await Cleanup(emailIdentity, true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Amazon SES v2 Coupon Newsletter Workflow is
complete.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred: {ex.Message}");
}
```

```
    }  
  }  
  
  /// <summary>  
  /// Populate the services for use within the console application.  
  /// </summary>  
  /// <param name="host">The services host.</param>  
  private static void ServicesSetup(IHost host)  
  {  
    _sesv2Wrapper = host.Services.GetRequiredService<SESV2Wrapper>();  
  }  
  
  /// <summary>  
  /// Set up the resources for the workflow.  
  /// </summary>  
  /// <returns>The email address of the verified identity.</returns>  
  public static async Task<string?> PrepareApplication()  
  {  
    var htmlContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +  
_htmlContentFile);  
    var textContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +  
_textContentFile);  
  
    Console.WriteLine(new string('-', 80));  
    Console.WriteLine("1. In this step, we will prepare the application:" +  
      "\r\n - Create a verified email identity for sending and  
replying to emails." +  
      "\r\n - Create a contact list to store the subscribers'  
email addresses." +  
      "\r\n - Create an email template for the coupon  
newsletter.\r\n");  
  
    // Prompt the user for a verified email address.  
    while (!IsEmail(_verifiedEmail))  
    {  
      Console.Write("Enter a verified email address or an email to verify: ");  
      _verifiedEmail = Console.ReadLine();  
    }  
  
    try  
    {  
      // Create an email identity and start the verification process.  
      await _sesv2Wrapper.CreateEmailIdentityAsync(_verifiedEmail);  
      Console.WriteLine($"Identity {_verifiedEmail} created.");  
    }  
  }  
}
```



```
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Identity {_verifiedEmail} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating email identity: {ex.Message}");
    }

    // Create a contact list.
    try
    {
        await _sesv2Wrapper.CreateContactListAsync(_contactListName);
        Console.WriteLine($"Contact list {_contactListName} created.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Contact list {_contactListName} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating contact list: {ex.Message}");
    }

    // Create an email template.
    try
    {
        await _sesv2Wrapper.CreateEmailTemplateAsync(_templateName, _subject,
htmlContent, textContent);
        Console.WriteLine($"Email template {_templateName} created.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Email template {_templateName} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating email template: {ex.Message}");
    }

    return _verifiedEmail;
}
```

```

    /// <summary>
    /// Generate subscriber addresses and send welcome emails.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> GatherSubscriberEmailAddresses(string
fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("2. In Step 2, we will gather subscriber email addresses:"
+
            "\r\n - Prompt the user for a base email address." +
            "\r\n - Create 3 variants of the email address using
subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com)." +
            "\r\n - Add each variant as a contact to the contact
list." +
            "\r\n - Send a welcome email to each new contact.\r\n");

        // Prompt the user for a base email address.
        while (!IsEmail(_baseEmailAddress))
        {
            Console.Write("Enter a base email address (e.g., user@example.com): ");
            _baseEmailAddress = Console.ReadLine();
        }

        // Create 3 variants of the email address using +ses-weekly-newsletter-1,
+ses-weekly-newsletter-2, etc.
        var baseEmailAddressParts = _baseEmailAddress!.Split("@");
        for (int i = 1; i <= 3; i++)
        {
            string emailAddress = $"{baseEmailAddressParts[0]}+ses-weekly-
newsletter-{i}@{baseEmailAddressParts[1]}";

            try
            {
                // Create a contact with the email address in the contact list.
                await _sesv2Wrapper.CreateContactAsync(emailAddress,
_contactListName);
                Console.WriteLine($"Contact {emailAddress} added to the
{_contactListName} contact list.");
            }
            catch (AlreadyExistsException)
            {

```

```
        Console.WriteLine($"Contact {emailAddress} already exists in the
{_contactListName} contact list.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating contact {emailAddress}:
{ex.Message}");
        return false;
    }

    // Send a welcome email to the new contact.
    try
    {
        string subject = "Welcome to the Weekly Coupons Newsletter";
        string htmlContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _htmlWelcomeFile);
        string textContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _textWelcomeFile);

        await _sesv2Wrapper.SendEmailAsync(fromEmailAddress, new
List<string> { emailAddress }, subject, htmlContent, textContent);
        Console.WriteLine($"Welcome email sent to {emailAddress}.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending welcome email to {emailAddress}:
{ex.Message}");
        return false;
    }

    // Wait 2 seconds before sending the next email (if the account is in
the SES Sandbox).
    await Task.Delay(2000);
}

return true;
}

/// <summary>
/// Send the coupon newsletter to the subscribers in the contact list.
/// </summary>
/// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
/// <returns>True if successful.</returns>
```

```
public static async Task<bool> SendCouponNewsletter(string fromEmailAddress)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("3. In this step, we will send the coupon newsletter:" +
        "\r\n - Retrieve the list of contacts from the contact
list." +
        "\r\n - Send the coupon newsletter using the email
template to each contact.\r\n");

    // Retrieve the list of contacts from the contact list.
    var contacts = await _sesv2Wrapper.ListContactsAsync(_contactListName);
    if (!contacts.Any())
    {
        Console.WriteLine($"No contacts found in the {_contactListName} contact
list.");
        return false;
    }

    // Load the coupon data from the sample_coupons.json file.
    string couponsData = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _couponsDataFile);

    // Send the coupon newsletter to each contact using the email template.
    try
    {
        foreach (var contact in contacts)
        {
            // To use the Contact List for list management, send to only one
address at a time.
            await _sesv2Wrapper.SendEmailAsync(fromEmailAddress,
                new List<string> { contact.EmailAddress },
                null, null, null, _templateName, couponsData, _contactListName);
        }

        Console.WriteLine($"Coupon newsletter sent to contact list
{_contactListName}.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending coupon newsletter to contact list
{_contactListName}: {ex.Message}");
        return false;
    }
}
```

```
        return true;
    }

    /// <summary>
    /// Provide instructions for monitoring sending activity and metrics.
    /// </summary>
    /// <param name="interactive">True to run in interactive mode.</param>
    /// <returns>True if successful.</returns>
    public static bool MonitorAndReview(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. In step 4, we will monitor and review:" +
            "\r\n - Provide instructions for the user to review the
sending activity and metrics in the AWS console.\r\n");

        Console.WriteLine("Review your sending activity using the SES Homepage in
the AWS console.");
        Console.WriteLine("Press Enter to open the SES Homepage in your default
browser...");
        if (interactive)
        {
            Console.ReadLine();
            try
            {
                // Open the SES Homepage in the default browser.
                Process.Start(new ProcessStartInfo
                {
                    FileName = "https://console.aws.amazon.com/ses/home",
                    UseShellExecute = true
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error opening the SES Homepage: {ex.Message}");
                return false;
            }
        }

        Console.WriteLine("Review the sending activity and email metrics, then press
Enter to continue...");
        if (interactive)
            Console.ReadLine();
        return true;
    }
}
```

```

    }

    /// <summary>
    /// Clean up the resources used in the workflow.
    /// </summary>
    /// <param name="verifiedEmailAddress">The verified email address from
    PrepareApplication.</param>
    /// <param name="interactive">True if interactive.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Cleanup(string verifiedEmailAddress, bool
    interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Finally, we clean up resources:" +
            "\r\n - Delete the contact list (which also deletes all
    contacts within it)." +
            "\r\n - Delete the email template." +
            "\r\n - Optionally delete the verified email identity.\r
    \n");

        Console.WriteLine("Cleaning up resources...");

        // Delete the contact list (this also deletes all contacts in the list).
        try
        {
            await _sesv2Wrapper.DeleteContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Contact list {_contactListName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting contact list {_contactListName}:
    {ex.Message}");
            return false;
        }

        // Delete the email template.
        try
        {
            await _sesv2Wrapper.DeleteEmailTemplateAsync(_templateName);
            Console.WriteLine($"Email template {_templateName} deleted.");
        }
    }

```

```
    }
    catch (NotFoundException)
    {
        Console.WriteLine($"Email template {_templateName} not found.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error deleting email template {_templateName}:
{ex.Message}");
        return false;
    }

    // Ask the user if they want to delete the email identity.
    var deleteIdentity = !interactive ||
        GetYesNoResponse(
(y/n) "$Do you want to delete the email identity {verifiedEmailAddress}?");
    if (deleteIdentity)
    {
        try
        {
            await _sesv2Wrapper.DeleteEmailIdentityAsync(verifiedEmailAddress);
            Console.WriteLine($"Email identity {verifiedEmailAddress}
deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine(
                $"Email identity {verifiedEmailAddress} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"Error deleting email identity {verifiedEmailAddress}:
{ex.Message}");
            return false;
        }
    }
    else
    {
        Console.WriteLine(
            $"Skipping deletion of email identity {verifiedEmailAddress}.");
    }
}
```

```

        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>
    /// Simple check to verify a string is an email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email.</returns>
    private static bool IsEmail(string? email)
    {
        if (string.IsNullOrEmpty(email))
            return false;
        return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$",
RegexOptions.IgnoreCase);
    }
}

```

Wrapper per le operazioni di assistenza.

```

using System.Net;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;

namespace Sesv2Scenario;

/// <summary>
/// Wrapper class for Amazon Simple Email Service (SES) v2 operations.
/// </summary>

```



```
public class SESv2Wrapper
{
    private readonly IAmazonSimpleEmailServiceV2 _sesClient;

    /// <summary>
    /// Constructor for the SESv2Wrapper.
    /// </summary>
    /// <param name="sesClient">The injected SES v2 client.</param>
    public SESv2Wrapper(IAmazonSimpleEmailServiceV2 sesClient)
    {
        _sesClient = sesClient;
    }

    /// <summary>
    /// Creates a contact and adds it to the specified contact list.
    /// </summary>
    /// <param name="emailAddress">The email address of the contact.</param>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The response from the CreateContact operation.</returns>
    public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
    {
        var request = new CreateContactRequest
        {
            EmailAddress = emailAddress,
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (NotFoundException ex)
        {

```

```
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
    }
    return false;
}

/// <summary>
/// Creates a contact list with the specified name.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateContactListAsync(string contactListName)
{
    var request = new CreateContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Contact list with name {contactListName} already
exists.");
        Console.WriteLine(ex.Message);
        return true;
    }
    catch (LimitExceededException ex)
    {
```

```
        Console.WriteLine("The limit for contact lists has been exceeded.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
    }
    return false;
}

/// <summary>
/// Creates an email identity (email address or domain) and starts the
verification process.
/// </summary>
/// <param name="emailIdentity">The email address or domain to create and
verify.</param>
/// <returns>The response from the CreateEmailIdentity operation.</returns>
public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
{
    var request = new CreateEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.CreateEmailIdentityAsync(request);
        return response;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email identity {emailIdentity} already exists.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (ConcurrentModificationException ex)
```

```
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (LimitExceededException ex)
    {
        Console.WriteLine("The limit for email identities has been exceeded.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Creates an email template with the specified content.
/// </summary>
/// <param name="templateName">The name of the email template.</param>
/// <param name="subject">The subject of the email template.</param>
/// <param name="htmlContent">The HTML content of the email template.</param>
/// <param name="textContent">The text content of the email template.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
```

```
{
    var request = new CreateEmailTemplateRequest
    {
        TemplateName = templateName,
        TemplateContent = new EmailTemplateContent
        {
            Subject = subject,
            Html = htmlContent,
            Text = textContent
        }
    };

    try
    {
        var response = await _sesClient.CreateEmailTemplateAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email template with name {templateName} already
exists.");
        Console.WriteLine(ex.Message);
    }
    catch (LimitExceededException ex)
    {
        Console.WriteLine("The limit for email templates has been exceeded.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email template:
{ex.Message}");
    }

    return false;
}

///
```

```
/// Deletes a contact list and all contacts within it.
/// </summary>
/// <param name="contactListName">The name of the contact list to delete.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteContactListAsync(string contactListName)
{
    var request = new DeleteContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.DeleteContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
    }

    return false;
}
```

```
/// <summary>
/// Deletes an email identity (email address or domain).
/// </summary>
/// <param name="emailIdentity">The email address or domain to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
{
    var request = new DeleteEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.DeleteEmailIdentityAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
    }

    return false;
}
```

```
/// <summary>
/// Deletes an email template.
/// </summary>
/// <param name="templateName">The name of the email template to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailTemplateAsync(string templateName)
{
    var request = new DeleteEmailTemplateRequest
    {
        TemplateName = templateName
    };

    try
    {
        var response = await _sesClient.DeleteEmailTemplateAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email template {templateName} does not exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
    }

    return false;
}

/// <summary>
/// Lists the contacts in the specified contact list.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The list of contacts response from the ListContacts operation.</
returns>
public async Task<List<Contact>> ListContactsAsync(string contactListName)
```



```
{
    var request = new ListContactsRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.ListContactsAsync(request);
        return response.Contacts;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
    }

    return new List<Contact>();
}

/// <summary>
/// Sends an email with the specified content and options.
/// </summary>
/// <param name="fromEmailAddress">The email address to send the email from.</
param>
/// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
/// <param name="subject">The subject of the email.</param>
/// <param name="htmlContent">The HTML content of the email.</param>
/// <param name="textContent">The text content of the email.</param>
/// <param name="templateName">The name of the email template to use
(optional).</param>
```

```
/// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
/// <param name="contactListName">The name of the contact list for unsubscribe
functionality (optional).</param>
/// <returns>The MessageId response from the SendEmail operation.</returns>
public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,
    string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
{
    var request = new SendEmailRequest
    {
        FromEmailAddress = fromEmailAddress
    };

    if (toEmailAddresses.Any())
    {
        request.Destination = new Destination { ToAddresses =
toEmailAddresses };
    }

    if (!string.IsNullOrEmpty(templateName))
    {
        request.Content = new EmailContent()
        {
            Template = new Template
            {
                TemplateName = templateName,
                TemplateData = templateData
            }
        };
    }
    else
    {
        request.Content = new EmailContent
        {
            Simple = new Message
            {
                Subject = new Content { Data = subject },
                Body = new Body
                {
                    Html = new Content { Data = htmlContent },
                    Text = new Content { Data = textContent }
                }
            }
        }
    }
}
```

```
        }
    };
}

if (!string.IsNullOrEmpty(contactListName))
{
    request.ListManagementOptions = new ListManagementOptions
    {
        ContactListName = contactListName
    };
}

try
{
    var response = await _sesClient.SendEmailAsync(request);
    return response.MessageId;
}
catch (AccountSuspendedException ex)
{
    Console.WriteLine("The account's ability to send email has been
permanently restricted.");
    Console.WriteLine(ex.Message);
}
catch (MailFromDomainNotVerifiedException ex)
{
    Console.WriteLine("The sending domain is not verified.");
    Console.WriteLine(ex.Message);
}
catch (MessageRejectedException ex)
{
    Console.WriteLine("The message content is invalid.");
    Console.WriteLine(ex.Message);
}
catch (SendingPausedException ex)
{
    Console.WriteLine("The account's ability to send email is currently
paused.");
    Console.WriteLine(ex.Message);
}
catch (TooManyRequestsException ex)
{
    Console.WriteLine("Too many requests were made. Please try again
later.");
    Console.WriteLine(ex.Message);
}
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
    }

    return string.Empty;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateContact](#)
 - [CreateContactList](#)
 - [CreateEmailIdentity](#)
 - [CreateEmailTemplate](#)
 - [DeleteContactList](#)
 - [DeleteEmailIdentity](#)
 - [DeleteEmailTemplate](#)
 - [ListContacts](#)
 - [SendEmail.semplice](#)
 - [SendEmail.modello](#)

Esempi di utilizzo di Amazon SNS AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Amazon SNS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.


Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon SNS

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon SNS.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, consulta la [ListTopics](#) sezione AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Azioni

CheckIfPhoneNumberIsOptedOut

Il seguente esempio di codice mostra come utilizzare `CheckIfPhoneNumberIsOptedOut`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, consulta la [CheckIfPhoneNumberIsOptedOut](#) sezione AWS SDK for .NET API Reference.

CreateTopic

Il seguente esempio di codice mostra come utilizzare `CreateTopic`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea un argomento con un nome di specifico.

```
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;
    }
}

```

Crea un nuovo argomento con un nome e attributi FIFO e di deduplicazione specifici.

```

    /// <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;
}
```

- Per i dettagli sull'API, consulta la [CreateTopic](#) sezione AWS SDK for .NET API Reference.

DeleteTopic

Il seguente esempio di codice mostra come utilizzare `DeleteTopic`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina un argomento in base all'ARN dell'argomento.

```
/// <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;
}

```

- Per i dettagli sull'API, consulta la [DeleteTopic](#) sezione AWS SDK for .NET API Reference.

GetTopicAttributes

Il seguente esempio di codice mostra come utilizzare `GetTopicAttributes`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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");
        }
    }
}

```

- Per i dettagli sull'API, [GetTopicAttributes](#) consulta AWS SDK for .NET API Reference.

ListSubscriptions

Il seguente esempio di codice mostra come utilizzare `ListSubscriptions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListSubscriptions](#) consulta AWS SDK for .NET API Reference.

ListTopics

Il seguente esempio di codice mostra come utilizzare `ListTopics`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListTopics](#) consulta AWS SDK for .NET API Reference.

Publish

Il seguente esempio di codice mostra come utilizzare `Publish`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Publicare un messaggio in un argomento.

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

Pubblica un messaggio in un argomento con opzioni di gruppo, duplicazione e attributo.

```
/// <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);
}
}

```

Applica le selezioni dell'utente all'azione di pubblicazione.

```

/// <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;
}
```

- Per informazioni dettagliate sulle API, consulta [Pubblicazione](#) nella Documentazione di riferimento per le API AWS SDK for .NET .

Subscribe

Il seguente esempio di codice mostra come usare `Subscribe`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Sottoscrivi un indirizzo email a un argomento.

```
/// <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;
    }

```

Iscrivi una coda a un argomento con filtri opzionali.

```

    /// <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;
}
```

- Per informazioni dettagliate sulle API, consulta [Sottoscrizione](#) nella Documentazione di riferimento sulle API AWS SDK for .NET .

Unsubscribe

Il seguente esempio di codice mostra come utilizzare `Unsubscribe`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Annulla l'iscrizione a un argomento tramite un ARN di sottoscrizione.

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

- Per informazioni dettagliate sulle API, consulta [Annullamento della sottoscrizione](#) nella Documentazione di riferimento per le API AWS SDK for .NET .

Scenari

Publicazione di un SMS

Il seguente esempio di codice mostra come pubblicare messaggi SMS utilizzando Amazon SNS.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sulle API, consulta [Pubblicazione](#) nella Documentazione di riferimento per le API AWS SDK for .NET .

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

AWS SDK for .NET

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
/// <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 messageId = 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($"{t{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;
}
}
```

Crea una classe che avvolge le operazioni di Amazon SQS.

```
/// <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;
}
}

```

Crea una classe che avvolge le operazioni di Amazon SNS.

```

/// <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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
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;
        }
    }
}
}
```


Esempi di utilizzo di Amazon SQS AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET con Amazon SQS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve Amazon SQS

I seguenti esempi di codice mostrano come iniziare a usare Amazon SQS.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Azioni

CreateQueue

Il seguente esempio di codice mostra come utilizzare `CreateQueue`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda con un nome specifico.

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

Crea una coda Amazon SQS e inviale un messaggio.

```
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;
}
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [CreateQueueReference](#).

DeleteMessage

Il seguente esempio di codice mostra come utilizzare `DeleteMessage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ricevi un messaggio da una coda Amazon SQS e poi elimina il messaggio.

```
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;
}
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [DeleteMessageReference](#).

DeleteMessageBatch

Il seguente esempio di codice mostra come utilizzare `DeleteMessageBatch`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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();
}
```

- Per i dettagli sull'API, [DeleteMessageBatch](#) consulta AWS SDK for .NET API Reference.

DeleteQueue

Il seguente esempio di codice mostra come utilizzare `DeleteQueue`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina una coda utilizzando il relativo 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;
}
```

- Per i dettagli sull'API, consulta la sezione [DeleteQueue AWS SDK for .NET API Reference](#).

GetQueueAttributes

Il seguente esempio di codice mostra come utilizzare `GetQueueAttributes`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetQueueAttributes](#) consulta AWS SDK for .NET API Reference.

GetQueueUrl

Il seguente esempio di codice mostra come utilizzare `GetQueueUrl`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK for .NET API Reference.

ReceiveMessage

Il seguente esempio di codice mostra come utilizzare `ReceiveMessage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ricevi messaggi da una coda utilizzando il relativo 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;
}
```

Ricevi un messaggio da una coda Amazon SQS, quindi elimina il messaggio.

```
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;
}
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [ReceiveMessage](#) Reference.

SendMessage

Il seguente esempio di codice mostra come utilizzare `SendMessage`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda Amazon SQS e inviale un messaggio.

```
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;
}
}
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for .NET API [SendMessage](#) Reference.

SetQueueAttributes

Il seguente esempio di codice mostra come utilizzare `SetQueueAttributes`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta l'attributo policy di una coda per un argomento.

```
/// <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;
    }

```

- Per i dettagli sull'API, consulta la sezione [SetQueueAttributes AWS SDK for .NET API Reference](#).

Scenari

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).

- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
/// <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(
        $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
    for (int i = 0; i < _tones.Length; i++)
    {
        Console.WriteLine($"  {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:" +
                $"{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;
}
}
```

Crea una classe che avvolge le operazioni di Amazon SQS.

```
/// <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\"" +
                "}, " +
        "}, " +

```



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

Crea una classe che avvolge le operazioni di Amazon SNS.

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

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)

- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Pubblicare](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

AWS SDK for .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
        }
    }
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

AWS SDK for .NET

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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;
    }
}
```

Esempi di Step Functions utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with Step Functions.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Funzioni Hello Step

I seguenti esempi di codice mostrano come iniziare a usare Step Functions.

AWS SDK for .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}  
}
```

- Per i dettagli sull'API, [ListStateMachines](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

CreateActivity

Il seguente esempio di codice mostra come utilizzare `CreateActivity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateActivity](#) consulta AWS SDK for .NET API Reference.

CreateStateMachine

Il seguente esempio di codice mostra come utilizzare `CreateStateMachine`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [CreateStateMachine](#) consulta AWS SDK for .NET API Reference.

DeleteActivity

Il seguente esempio di codice mostra come utilizzare `DeleteActivity`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteActivity](#) consulta AWS SDK for .NET API Reference.

DeleteStateMachine

Il seguente esempio di codice mostra come utilizzare `DeleteStateMachine`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [DeleteStateMachine](#) consulta AWS SDK for .NET API Reference.

DescribeExecution

Il seguente esempio di codice mostra come utilizzare `DescribeExecution`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeExecution](#) consulta AWS SDK for .NET API Reference.

DescribeStateMachine

Il seguente esempio di codice mostra come utilizzare `DescribeStateMachine`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeStateMachine](#) consulta AWS SDK for .NET API Reference.

GetActivityTask

Il seguente esempio di codice mostra come utilizzare `GetActivityTask`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [GetActivityTask](#) consulta AWS SDK for .NET API Reference.

ListActivities

Il seguente esempio di codice mostra come utilizzare `ListActivities`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// List the Step Functions activities for the current account.
/// </summary>
/// <returns>A list of ActivityListItem.</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;
}
```

- Per i dettagli sull'API, [ListActivities](#) consulta AWS SDK for .NET API Reference.

ListExecutions

Il seguente esempio di codice mostra come utilizzare `ListExecutions`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListExecutions](#) consulta AWS SDK for .NET API Reference.

ListStateMachines

Il seguente esempio di codice mostra come utilizzare `ListStateMachines`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListStateMachines](#) consulta AWS SDK for .NET API Reference.

SendTaskSuccess

Il seguente esempio di codice mostra come utilizzare `SendTaskSuccess`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [SendTaskSuccess](#) consulta AWS SDK for .NET API Reference.

StartExecution

Il seguente esempio di codice mostra come utilizzare `StartExecution`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [StartExecution](#) consulta AWS SDK for .NET API Reference.

Scenari

Inizia a usare le macchine a stati

L'esempio di codice seguente mostra come:

- Crea un'attività.
- Crea una macchina a stati da una definizione di Amazon States Language che contiene l'attività creata in precedenza come passaggio.
- Esegui la macchina a stati e rispondi all'attività con l'input dell'utente.
- Ottieni lo stato e l'output finali al termine dell'esecuzione, quindi ripulisci le risorse.

AWS SDK for .NET

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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($"{describeStateMachineResponse.Name}\t{describeStateMachineResponse.StateMachineArn}");
    Console.WriteLine($"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.WriteLine("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);
    }
}

```

Definisci una classe che racchiuda le azioni della macchina a stati e delle attività.

```

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 ActivityListItem.</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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [CreateActivity](#)
 - [CreateStateMachine](#)
 - [DeleteActivity](#)
 - [DeleteStateMachine](#)
 - [DescribeExecution](#)
 - [DescribeStateMachine](#)
 - [GetActivityTask](#)
 - [ListActivities](#)
 - [ListStateMachines](#)
 - [SendTaskSuccess](#)
 - [StartExecution](#)

- [StopExecution](#)

AWS STS esempi utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with AWS STS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

AssumeRole

Il seguente esempio di codice mostra come utilizzare `AssumeRole`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AssumeRole](#) consulta AWS SDK for .NET API Reference.

AWS Support esempi che utilizzano AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK for .NET with AWS Support.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.


Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Salve AWS Support

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Support.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeServices](#) consulta AWS SDK for .NET API Reference.

Argomenti

- [Azioni](#)
- [Scenari](#)

Azioni

AddAttachmentsToSet

Il seguente esempio di codice mostra come utilizzare `AddAttachmentsToSet`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [AddAttachmentsToSet](#) consulta AWS SDK for .NET API Reference.

AddCommunicationToCase

Il seguente esempio di codice mostra come utilizzare `AddCommunicationToCase`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
    }

```

- Per i dettagli sull'API, [AddCommunicationToCase](#) consulta AWS SDK for .NET API Reference.

CreateCase

Il seguente esempio di codice mostra come utilizzare `CreateCase`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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;  
}
```

- Per i dettagli sull'API, [CreateCase](#) consulta AWS SDK for .NET API Reference.

DescribeAttachment

Il seguente esempio di codice mostra come utilizzare `DescribeAttachment`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [DescribeAttachment](#) consulta AWS SDK for .NET API Reference.

DescribeCases

Il seguente esempio di codice mostra come utilizzare `DescribeCases`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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;
    }
}
```

- Per i dettagli sull'API, [DescribeCases](#) consulta AWS SDK for .NET API Reference.

DescribeCommunications

Il seguente esempio di codice mostra come utilizzare `DescribeCommunications`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeCommunications](#) consulta AWS SDK for .NET API Reference.

DescribeServices

Il seguente esempio di codice mostra come utilizzare `DescribeServices`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Get the descriptions of AWS services.
/// </summary>
/// <param name="name">Optional language for services.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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;
}
```

- Per i dettagli sull'API, [DescribeServices](#) consulta AWS SDK for .NET API Reference.

DescribeSeverityLevels

Il seguente esempio di codice mostra come utilizzare `DescribeSeverityLevels`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Get the descriptions of support severity levels.
/// </summary>
/// <param name="name">Optional language for severity levels.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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;
}
```

- Per i dettagli sull'API, [DescribeSeverityLevels](#) consulta AWS SDK for .NET API Reference.

ResolveCase

Il seguente esempio di codice mostra come utilizzare `ResolveCase`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sull'API, [ResolveCase](#) consulta AWS SDK for .NET API Reference.

Scenari

Come iniziare con i casi

L'esempio di codice seguente mostra come:

- Ottieni e visualizza i servizi e i livelli di gravità disponibili per i casi.
- Crea una richiesta di supporto utilizzando un servizio, una categoria e un livello di gravità selezionato.
- Ottieni e visualizza un elenco di casi aperti per il giorno corrente.
- Aggiungi un set di collegamenti e una comunicazione al nuovo caso.
- Descrivi il nuovo collegamento e la nuova comunicazione per il caso.
- Risolvi il caso.
- Ottieni e visualizza un elenco di casi risolti per il giorno corrente.

AWS SDK for .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
/// <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.CreatedOn: {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));
}
}
```

Metodi wrapper utilizzati dallo scenario per AWS Support le azioni.

```
/// <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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
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 Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
    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;
    }
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for .NET .
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)
 - [DescribeCases](#)
 - [DescribeCommunications](#)
 - [DescribeServices](#)
 - [DescribeSeverityLevels](#)
 - [ResolveCase](#)

Esempi di Amazon Transcribe utilizzando AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Transcribe. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

CreateVocabulary

Il seguente esempio di codice mostra come utilizzare `CreateVocabulary`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [CreateVocabulary](#) consulta AWS SDK for .NET API Reference.

DeleteMedicalTranscriptionJob

Il seguente esempio di codice mostra come utilizzare `DeleteMedicalTranscriptionJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    /// <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;
    }
```

- Per i dettagli sull'API, [DeleteMedicalTranscriptionJob](#) consulta AWS SDK for .NET API Reference.

DeleteTranscriptionJob

Il seguente esempio di codice mostra come utilizzare `DeleteTranscriptionJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DeleteTranscriptionJob](#) consulta AWS SDK for .NET API Reference.

DeleteVocabulary

Il seguente esempio di codice mostra come utilizzare `DeleteVocabulary`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [DeleteVocabulary](#) consulta AWS SDK for .NET API Reference.

GetTranscriptionJob

Il seguente esempio di codice mostra come utilizzare `GetTranscriptionJob`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sull'API, [GetTranscriptionJob](#) consulta AWS SDK for .NET API Reference.

GetVocabulary

Il seguente esempio di codice mostra come utilizzare `GetVocabulary`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [GetVocabulary](#) consulta AWS SDK for .NET API Reference.

ListMedicalTranscriptionJobs

Il seguente esempio di codice mostra come utilizzare `ListMedicalTranscriptionJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListMedicalTranscriptionJobs](#) consulta AWS SDK for .NET API Reference.

ListTranscriptionJobs

Il seguente esempio di codice mostra come utilizzare `ListTranscriptionJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
    }

```

- Per i dettagli sull'API, [ListTranscriptionJobs](#) consulta AWS SDK for .NET API Reference.

ListVocabularies

Il seguente esempio di codice mostra come utilizzare `ListVocabularies`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
}

```

- Per i dettagli sull'API, [ListVocabularies](#) consulta AWS SDK for .NET API Reference.

StartMedicalTranscriptionJob

Il seguente esempio di codice mostra come utilizzare `StartMedicalTranscriptionJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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 MedicalTransactionJob 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;
}

```

- Per i dettagli sull'API, [StartMedicalTranscriptionJob](#) consulta AWS SDK for .NET API Reference.

StartTranscriptionJob

Il seguente esempio di codice mostra come utilizzare `StartTranscriptionJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// <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;
    }

```

- Per i dettagli sull'API, [StartTranscriptionJob](#) consulta AWS SDK for .NET API Reference.

UpdateVocabulary

Il seguente esempio di codice mostra come utilizzare `UpdateVocabulary`.

AWS SDK for .NET

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [UpdateVocabulary](#) consulta AWS SDK for .NET API Reference.

Esempi di Amazon Translate con AWS SDK for .NET

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando Amazon Translate. AWS SDK for .NET

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Ogni esempio include un collegamento a GitHub, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Azioni](#)

Azioni

DescribeTextTranslationJob

Il seguente esempio di codice mostra come utilizzare `DescribeTextTranslationJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [DescribeTextTranslationJob](#) consulta AWS SDK for .NET API Reference.

ListTextTranslationJobs

Il seguente esempio di codice mostra come utilizzare `ListTextTranslationJobs`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [ListTextTranslationJobs](#) consulta AWS SDK for .NET API Reference.

StartTextTranslationJob

Il seguente esempio di codice mostra come utilizzare `StartTextTranslationJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per i dettagli sull'API, [StartTextTranslationJob](#) consulta AWS SDK for .NET API Reference.

StopTextTranslationJob

Il seguente esempio di codice mostra come utilizzare `StopTextTranslationJob`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sull'API, [StopTextTranslationJob](#) consulta AWS SDK for .NET API Reference.

TranslateText

Il seguente esempio di codice mostra come utilizzare `TranslateText`.

AWS SDK for .NET

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
}
```

- Per i dettagli sull'API, [TranslateText](#) consulta AWS SDK for .NET API Reference.

Esempi trasversali di servizi che utilizzano AWS SDK for .NET

Le seguenti applicazioni di esempio utilizzano il AWS SDK for .NET per funzionare su più applicazioni Servizi AWS.

Gli esempi trasversali mirano a un livello avanzato di esperienza per aiutarti a iniziare a creare applicazioni.

Esempi

- [Costruzione di un'applicazione per la pubblicazione e la sottoscrizione che traduce i messaggi](#)
- [Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette](#)
- [Creazione di un'applicazione Web per tracciare i dati DynamoDB](#)
- [Creazione di un tracciatore di elementi di lavoro di Aurora Serverless](#)
- [Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio](#)
- [Rileva oggetti nelle immagini con Amazon Rekognition utilizzando un SDK AWS](#)
- [Trasforma i dati per la tua applicazione con S3 Object Lambda](#)
- [Utilizza il AWS Message Processing Framework for .NET per pubblicare e ricevere messaggi Amazon SQS](#)

Costruzione di un'applicazione per la pubblicazione e la sottoscrizione che traduce i messaggi

AWS SDK for .NET

Mostra come utilizzare l'API .NET di Amazon Simple Notification Service per creare un'applicazione Web con funzionalità di sottoscrizione e pubblicazione. Inoltre, questa applicazione di esempio traduce anche i messaggi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon SNS

- Amazon Translate

Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette

AWS SDK for .NET

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creazione di un'applicazione Web per tracciare i dati DynamoDB

AWS SDK for .NET

Mostra come utilizzare l'API Amazon DynamoDB per creare un'applicazione Web dinamica che traccia i dati di lavoro DynamoDB.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Creazione di un tracciatore di elementi di lavoro di Aurora Serverless

AWS SDK for .NET

Mostra come utilizzare per AWS SDK for .NET creare un'applicazione Web che tenga traccia degli elementi di lavoro in un database Amazon Aurora e invii report tramite e-mail utilizzando Amazon Simple Email Service (Amazon SES). Questo esempio utilizza un front-end creato con React.js per interagire con un backend .NET RESTful.

- Integra un'applicazione web React con AWS i servizi.
- Elenco, aggiunta e aggiornamento di elementi in una tabella Aurora.
- Invia un report per e-mail degli articoli di lavoro filtrati tramite Amazon SES.
- Distribuisci e gestisci risorse di esempio con lo AWS CloudFormation script incluso.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Aurora
- Amazon RDS
- Servizi di dati di Amazon RDS
- Amazon SES

Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio

AWS SDK for .NET

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Rileva oggetti nelle immagini con Amazon Rekognition utilizzando un SDK AWS

AWS SDK for .NET

Mostra come utilizzare l'API .NET di Amazon Rekognition per creare un'applicazione che utilizza Amazon Rekognition per identificare gli oggetti in base a una categoria nelle immagini situate in un bucket Amazon Simple Storage Service (Amazon S3). L'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#)

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Amazon S3
- Amazon SES

Trasforma i dati per la tua applicazione con S3 Object Lambda

AWS SDK for .NET

Mostra come aggiungere codice personalizzato alle richieste S3 GET standard per modificare l'oggetto richiesto recuperato da S3 in modo che l'oggetto soddisfi le esigenze del client o dell'applicazione richiedente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- Lambda
- Amazon S3

Utilizza il AWS Message Processing Framework for .NET per pubblicare e ricevere messaggi Amazon SQS

AWS SDK for .NET

Fornisce un tutorial per il AWS Message Processing Framework per .NET. Il tutorial crea un'applicazione Web che consente all'utente di pubblicare un messaggio Amazon SQS e un'applicazione da riga di comando che riceve il messaggio.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta il [tutorial completo](#) nella AWS SDK for .NET Developer Guide e l'esempio seguente. [GitHub](#)

Servizi utilizzati in questo esempio

- Amazon SQS

Sicurezza per questo AWS prodotto o servizio

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In quanto cliente AWS, è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza. La sicurezza è una responsabilità condivisa tra AWS e te. Il [modello di responsabilità condivisa](#) descrive questo come sicurezza del cloud e sicurezza nel cloud.

Security of the Cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce tutti i servizi offerti nel AWS Cloud e della fornitura di servizi che è possibile utilizzare in modo sicuro. La nostra responsabilità in AWS materia di sicurezza è la massima priorità e l'efficacia della nostra sicurezza viene regolarmente testata e verificata da revisori di terze parti nell'ambito dei Programmi di [AWS conformità](#).

Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato e da altri fattori, tra cui la sensibilità dei dati, i requisiti dell'organizzazione e le leggi e i regolamenti applicabili.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Argomenti

- [Protezione dei dati in questo AWS prodotto o servizio](#)
- [Identity and Access Management](#)
- [Convalida della conformità per questo AWS prodotto o servizio](#)
- [Resilienza per questo AWS prodotto o servizio](#)
- [Sicurezza dell'infrastruttura per questo AWS prodotto o servizio](#)
- [Imposizione di una versione TLS minima nel AWS SDK for .NET](#)
- [Migrazione del client di crittografia Amazon S3](#)

Protezione dei dati in questo AWS prodotto o servizio

Il [modello di responsabilità AWS condivisa](#) si applica alla protezione dei dati in questo AWS prodotto o servizio. Come descritto in questo modello, AWS è responsabile della protezione

dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi le [Domande frequenti sulla privacy dei dati](#). Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al [Modello di responsabilità condivisa AWS e GDPR](#) nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-2 per l'accesso AWS tramite un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-2](#).

Ti consigliamo vivamente di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con questo AWS prodotto o servizio o altro Servizi AWS utilizzando la console, l'API o gli SDK. AWS CLI AWS I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Identity and Access Management

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle

autorizzazioni) a utilizzare le risorse. AWS IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Argomenti

- [Destinatari](#)
- [Autenticazione con identità](#)
- [Gestione dell'accesso con policy](#)
- [Come Servizi AWS lavorare con IAM](#)
- [Risoluzione dei problemi di AWS identità e accesso](#)

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS svolgi.

Utente del servizio: se lo utilizzi Servizi AWS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità di AWS, consulta [Risoluzione dei problemi di AWS identità e accesso](#) o consulta la guida per l'utente della funzionalità Servizio AWS che stai utilizzando.

Amministratore del servizio: se sei responsabile delle AWS risorse della tua azienda, probabilmente hai pieno accesso a AWS. È tuo compito determinare a quali AWS funzionalità e risorse devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con AWS, consulta la guida per l'utente del Servizio AWS software che stai utilizzando.

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a AWS. Per visualizzare esempi di policy AWS basate sull'identità che puoi utilizzare in IAM, consulta la guida per l'utente di quella Servizio AWS che stai utilizzando.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l'accesso dell'utente root dell'account AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center (precedentemente AWS Single Sign-On), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS. Per ulteriori informazioni sull'accesso a AWS, vedi [Come accedere al tuo Account AWS nella Guida per l'utente di AWS](#).

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sull'utilizzo del metodo consigliato per firmare autonomamente le richieste, consulta [Signing AWS API request](#) nella IAM User Guide.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta [Autenticazione a più fattori](#) nella Guida per l'utente di AWS IAM Identity Center e [Utilizzo dell'autenticazione a più fattori \(MFA\) in AWS](#) nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un account AWS, si inizia con un'identità di accesso che ha accesso completo a tutti i servizi AWS e le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conservare le credenziali dell'utente root e utilizzarle per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'utente di IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, un provider di identità Web AWS Directory Service, la directory Identity Center o qualsiasi utente che accede Servizi AWS utilizzando credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni sul Centro identità IAM, consulta [Cos'è Centro identità IAM?](#) nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un [utente IAM](#) è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, per casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina [Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine](#) nella Guida per l'utente di IAM.

Un [gruppo IAM](#) è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato IAMAdmins e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta [Quando creare un utente IAM \(invece di un ruolo\)](#) nella Guida per l'utente di IAM.

Ruoli IAM

Un [ruolo IAM](#) è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Puoi assumere temporaneamente un ruolo IAM in AWS Management Console [cambiando ruolo](#). Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta [Utilizzo di ruoli IAM](#) nella Guida per l'utente di IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- **Accesso utente federato:** per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta [Creazione di un ruolo per un provider di identità di terza parte](#) nella Guida per l'utente di IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per ulteriori informazioni sui set di autorizzazioni, consulta [Set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .
- **Autorizzazioni utente IAM temporanee:** un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- **Accesso multi-account:** è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Differenza tra i ruoli IAM e le policy basate su risorse](#) nella Guida per l'utente di IAM.
- **Accesso a più servizi:** alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
 - **Sessioni di accesso diretto (FAS):** quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama un Servizio AWS, combinate con la richiesta Servizio AWS per

effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina [Forward access sessions](#).

- Ruolo di servizio: un ruolo di servizio è un [ruolo IAM](#) che un servizio assume per eseguire azioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione [Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS](#) nella Guida per l'utente di IAM.
- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.
- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 e che AWS CLI effettuano richieste API. AWS CLI è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un AWS ruolo a un'istanza EC2 e renderlo disponibile per tutte le sue applicazioni, crei un profilo di istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta [Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2](#) nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta [Quando creare un ruolo IAM \(invece di un utente\)](#) nella Guida per l'utente di IAM.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta [Panoramica delle policy JSON](#) nella Guida per l'utente di IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. Successivamente l'amministratore può aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'azione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'azione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall' o dall' AWS API.

Policy basate su identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruoli IAM). Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta [Creazione di policy IAM](#) nella Guida per l'utente di IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta [Scelta fra policy gestite e policy inline](#) nella Guida per l'utente di IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarle per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario [specificare un principale](#) in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano gli ACL. AWS WAF Per maggiori informazioni sulle ACL, consulta [Panoramica delle liste di controllo degli accessi \(ACL\)](#) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- **Limiti delle autorizzazioni:** un limite delle autorizzazioni è una funzione avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta [Limiti delle autorizzazioni per le entità IAM](#) nella Guida per l'utente di IAM.
- **Politiche di controllo dei servizi (SCP):** le SCP sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in AWS Organizations. AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più Account AWS di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità negli account dei membri, inclusa ciascuna. Utente root dell'account AWS Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle [Policy di controllo dei servizi](#) nella Guida per l'utente di AWS Organizations .
- **Policy di sessione:** le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta [Policy di sessione](#) nella Guida per l'utente di IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per sapere come si AWS determina se consentire una richiesta quando sono coinvolti più tipi di policy, consulta [Logica di valutazione delle policy](#) nella IAM User Guide.

Come Servizi AWS lavorare con IAM

Per avere una visione di alto livello di come Servizi AWS funziona la maggior parte delle funzionalità IAM, consulta [AWS i servizi che funzionano con IAM nella IAM](#) User Guide.

Per scoprire come utilizzare uno specifico Servizio AWS con IAM, consulta la sezione sulla sicurezza della Guida per l'utente del servizio pertinente.

Risoluzione dei problemi di AWS identità e accesso

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con un AWS IAM.

Argomenti

- [Non sono autorizzato a eseguire alcuna azione in AWS](#)
- [Non sono autorizzato a eseguire iam: PassRole](#)
- [Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse](#)

Non sono autorizzato a eseguire alcuna azione in AWS

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM mateojackson prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *my-example-widget* fittizia ma non dispone di autorizzazioni *aws:GetWidget* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente mateojackson deve essere aggiornata per consentire l'accesso alla risorsa *my-example-widget* utilizzando l'azione *aws:GetWidget*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in AWS. Tuttavia, l'operazione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie AWS risorse

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per servizi che supportano policy basate su risorse o liste di controllo accessi (ACL), utilizza tali policy per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se AWS supporta queste funzionalità, consulta [Come Servizi AWS lavorare con IAM](#).
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta [Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà](#) nella IAM User Guide.

- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta [Fornire l'accesso a soggetti Account AWS di proprietà di terze parti](#) nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta [Fornire l'accesso a utenti autenticati esternamente \(Federazione delle identità\)](#) nella Guida per l'utente di IAM.
- Per informazioni sulle differenze tra l'utilizzo di ruoli e policy basate su risorse per l'accesso multi-account, consulta [Differenza tra i ruoli IAM e le policy basate su risorse](#) nella Guida per l'utente IAM.

Convalida della conformità per questo AWS prodotto o servizio

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione [Scope by Compliance Program Servizi AWS](#) e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Scaricamento dei report in AWS Artifact](#) .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- [Guide introduttive su sicurezza e conformità](#): queste guide all'implementazione illustrano considerazioni sull'architettura e forniscono passaggi per implementare ambienti di base incentrati sulla AWS sicurezza e la conformità.
- [Progettazione per la sicurezza e la conformità HIPAA su Amazon Web Services](#): questo white paper descrive in che modo le aziende possono utilizzare AWS per creare applicazioni idonee all'HIPAA.

Note

Non Servizi AWS tutte sono idonee all'HIPAA. Per ulteriori informazioni, consulta la sezione [Riferimenti sui servizi conformi ai requisiti HIPAA](#).

- [AWS Risorse per](#) la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.

- [AWS Guide alla conformità dei clienti](#): comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).
- [Valutazione delle risorse con regole](#) nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- [AWS Security Hub](#)— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina [Documentazione di riferimento sui controlli della Centrale di sicurezza](#).
- [Amazon GuardDuty](#): Servizio AWS rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, come lo standard PCI DSS, soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- [AWS Audit Manager](#)— Ciò Servizio AWS consente di verificare continuamente l' AWS utilizzo per semplificare la gestione del rischio e la conformità alle normative e agli standard di settore.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Resilienza per questo AWS prodotto o servizio

L'infrastruttura AWS globale è costruita attorno a zone Regioni AWS di disponibilità.

Regioni AWS forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti.

Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

[Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedere Global Infrastructure.AWS](#)

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Sicurezza dell'infrastruttura per questo AWS prodotto o servizio

Questo AWS prodotto o servizio utilizza servizi gestiti ed è pertanto protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta [AWS Cloud Security](#). Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. O puoi utilizzare [AWS Security Token Service](#) (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Imposizione di una versione TLS minima nel AWS SDK for .NET

Per aumentare la sicurezza durante la comunicazione con AWS i servizi, è necessario configurarli AWS SDK for .NET per utilizzare TLS 1.2 o versioni successive.

AWS SDK for .NET utilizza il runtime di .NET sottostante per determinare il protocollo di sicurezza da utilizzare. Per impostazione predefinita, le versioni correnti di .NET utilizzano il protocollo configurato più recente supportato dal sistema operativo. L'applicazione può ignorare questo comportamento SDK, ma non è consigliabile farlo.

.NET Core

Per impostazione predefinita, .NET Core utilizza il protocollo configurato più recente supportato dal sistema operativo. AWS SDK for .NET non fornisce un meccanismo per sovrascrivere questo.

Se si utilizza una versione di .NET Core precedente alla 2.1, si consiglia vivamente di aggiornare la versione di .NET Core.

Vedere quanto segue per informazioni specifiche di ciascun sistema operativo.

Windows

Le distribuzioni moderne di Windows hanno il supporto TLS 1.2 [abilitato per impostazione predefinita](#). [Se utilizzi Windows 7 SP1 o Windows Server 2008 R2 SP1, devi assicurarti che il supporto TLS 1.2 sia abilitato nel registro, come descritto all'indirizzo <https://learn.microsoft.com/en-us/windows-server/security/tls/#tls-12-tls-registry-settings>](#) Se si esegue una distribuzione precedente, è necessario aggiornare il sistema operativo. Per informazioni sul supporto TLS 1.3 in Windows, consulta la documentazione Microsoft più recente per le versioni client o server minime richieste.

macOS

Se si esegue .NET Core 2.1 o versioni successive, TLS 1.2 è abilitato per impostazione predefinita. TLS 1.2 è supportato da [OS X Mavericks](#) 10.9 o versioni successive. [.NET Core versione 2.1 e successive richiedono versioni più recenti di macOS, come descritto all'indirizzo <https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos>](#).

Se si utilizza .NET Core 1.0, .NET Core [utilizza OpenSSL su macOS](#), una dipendenza che deve essere installata separatamente. OpenSSL ha aggiunto il supporto per TLS 1.2 nella versione 1.0.1 e ha aggiunto il supporto per TLS 1.3 nella versione 1.1.1.

Linux

.NET Core su Linux richiede OpenSSL, che viene fornito in bundle con molte distribuzioni Linux, ma può anche essere installato separatamente. OpenSSL ha aggiunto il supporto per TLS 1.2 nella versione 1.0.1 e ha aggiunto il supporto per TLS 1.3 nella versione 1.1.1. Se stai usando una

versione moderna di .NET Core (2.1 o successiva) e hai installato un gestore di pacchetti, è probabile che sia stata installata una versione più moderna di OpenSSL.

Per essere sicuro, è possibile eseguire `openssl version` in un terminale e verificare che la versione sia successiva alla 1.0.1.

.NET Framework

Se si esegue una versione moderna di .NET Framework (4.7 o versione successiva) e una versione moderna di Windows (almeno Windows 8 per i client, Windows Server 2012 o versione successiva per i server), TLS 1.2 è abilitato e utilizzato per impostazione predefinita.

Se utilizzi un runtime di .NET Framework che non utilizza le impostazioni del sistema operativo (.NET Framework da 3.5 a 4.5.2), AWS SDK for .NET tenterà di [aggiungere il supporto per TLS 1.1 e TLS 1.2](#) ai protocolli supportati. Se si utilizza .NET Framework 3.5, l'operazione avrà esito positivo solo se è installata la patch appropriata, come segue:

- [Windows 10 versione 1511 e Windows Server 2016 — KB3156421](#)
- [Windows 8.1 e Windows Server 2012 R2 — KB3154520](#)
- [Windows Server 2012 — KB3154519](#)
- [Windows 7 SP1 e Server 2008 R2 SP1 — KB3154518](#)

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e la versione minima di .NET Framework passerà alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

[Se l'applicazione è in esecuzione su un nuovo .NET Framework su Windows 7 SP1 o Windows Server 2008 R2 SP1, è necessario assicurarsi che il supporto TLS 1.2 sia abilitato nel registro, come descritto all'indirizzo <https://learn.microsoft.com/en-us/windows-server/security/tls/#tls-12-tls-registry-settings>](#) Le versioni più recenti di Windows lo hanno [abilitato per impostazione predefinita](#).

Per informazioni dettagliate sulle procedure consigliate per l'utilizzo di TLS con .NET Framework, consulta l'articolo di Microsoft all'[indirizzo <https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls>](https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls).

AWS Tools for PowerShell

[AWS Tools for PowerShell](#) usa AWS SDK for .NET per tutte le chiamate ai AWS servizi. Il comportamento dell'ambiente dipende dalla versione di Windows PowerShell in esecuzione, come segue.

Da Windows PowerShell 2.0 a 5.x

Da Windows PowerShell 2.0 a 5.x vengono eseguiti su .NET Framework. È possibile verificare quale runtime .NET (2.0 o 4.0) viene utilizzato PowerShell utilizzando il comando seguente.

```
$PSVersionTable.CLRVersion
```

- Quando si utilizza .NET Runtime 2.0, seguire le istruzioni fornite in precedenza per quanto riguarda AWS SDK for .NET e .NET Framework 3.5.

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e la versione minima di .NET Framework passerà alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

- Quando si utilizza .NET Runtime 4.0, seguire le istruzioni fornite in precedenza per quanto riguarda AWS SDK for .NET e .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 e versioni successive funzionano su .NET Core. È possibile verificare quale versione di .NET Core viene utilizzata eseguendo il comando seguente.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.FrameworkName], $true).FrameworkName
```

Segui le istruzioni fornite in precedenza relative alla AWS SDK for .NET versione pertinente di .NET Core.

Xamarin

Per Xamarin, consulta le istruzioni all'indirizzo <https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security>. In sintesi:

Per Android

- Richiede Android 5.0 o versioni successive.
- Proprietà del progetto, Opzioni Android: HttpClient l'implementazione deve essere impostata su Android e l'implementazione SSL/TLS impostata su Native TLS 1.2+.

Per iOS

- Richiede iOS 7 o versioni successive.
- Project Properties, iOS Build: HttpClient l'implementazione deve essere impostata su NS UrlSession.

Per macOS

- Richiede macOS 10.9 o versioni successive.
- Project Options, Build, Mac Build: HttpClient l'implementazione deve essere impostata su NS UrlSession.

Unità

È necessario utilizzare Unity 2018.2 o versioni successive e utilizzare il runtime di scripting equivalente .NET 4.x. È possibile impostarlo in Project Settings, Configuration, Player, come descritto all'indirizzo <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html>. Il runtime di scripting equivalente .NET 4.x consente il supporto TLS 1.2 a tutte le piattaforme Unity che eseguono Mono o IL2CPP. Per ulteriori informazioni, vedere <https://blog.unity.com/technology/scripting-runtime-improvements-in-unity-2018-2>.

Browser (per Blazor) WebAssembly

WebAssembly viene eseguito nel browser anziché sul server e utilizza il browser per gestire il traffico HTTP. Pertanto, il supporto TLS è determinato dal supporto del browser.

[Blazor WebAssembly, in anteprima per ASP.NET Core 3.1, è supportato solo nei browser che supportano WebAssembly, come descritto all'indirizzo <https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms>](https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms). Tutti i browser principali supportavano TLS 1.2 prima del supporto. WebAssembly Se questo è il caso del browser, se l'app viene eseguita, può comunicare tramite TLS 1.2.

Consulta la documentazione del tuo browser per ulteriori informazioni e verifiche.

Migrazione del client di crittografia Amazon S3

Questo argomento mostra come migrare le applicazioni dalla versione 1 (V1) del client di crittografia Amazon Simple Storage Service (Amazon S3) alla versione 2 (V2) e garantire la disponibilità delle applicazioni durante tutto il processo di migrazione.

Gli oggetti crittografati con il client V2 non possono essere decrittografati con il client V1. Per facilitare la migrazione al nuovo client senza dover ricrittografare tutti gli oggetti contemporaneamente, è stato fornito un client «V1-transitional». Questo client può decrittografare gli oggetti crittografati V1 e V2, ma crittografa gli oggetti solo in formato compatibile con V1. Il client V2 può decrittografare gli oggetti crittografati V1 e V2 (se abilitato per gli oggetti V1), ma crittografa gli oggetti solo in formato compatibile con V2.

Panoramica sulla migrazione

Questa migrazione avviene in tre fasi. Queste fasi vengono introdotte qui e descritte in dettaglio più avanti. Ogni fase deve essere completata per tutti i client che utilizzano oggetti condivisi prima di iniziare la fase successiva.

1. Aggiorna i client esistenti ai client di transizione V1 per leggere nuovi formati. Innanzitutto, aggiorna le tue applicazioni in modo che dipendano dal client di transizione V1 anziché dal client V1. Il client di transizione V1 consente al codice esistente di decrittografare gli oggetti scritti dai nuovi client V2 e gli oggetti scritti in formato compatibile con V1.

Note

Il client v1-Transitional viene fornito solo a scopo di migrazione. Procedi all'aggiornamento al client V2 dopo il passaggio al client v1-Transitional.

2. Migra i client di transizione V1 ai client V2 per scrivere nuovi formati. Quindi, sostituisci tutti i client di transizione V1 nelle tue applicazioni con client V2 e imposta il profilo di sicurezza su.

V2AndLegacy L'impostazione di questo profilo di sicurezza sui client V2 consente a tali client di decrittografare gli oggetti crittografati in un formato compatibile con V1.

3. Aggiorna i client V2 in modo che non leggano più i formati V1. Infine, dopo che tutti i client sono stati migrati alla V2 e tutti gli oggetti sono stati crittografati o ricrittografati in un formato compatibile con V2, imposta il profilo di sicurezza V2 su invece di V2AndLegacy. Ciò impedisce la decrittografia di oggetti in formato compatibile con V1.

Aggiorna i client esistenti ai client di transizione V1 per leggere nuovi formati

Il client di crittografia V2 utilizza algoritmi di crittografia che le versioni precedenti del client non supportano. Il primo passo della migrazione consiste nell'aggiornare i client di decrittografia V1 in modo che possano leggere il nuovo formato.

Il client di transizione V1 consente alle applicazioni di decrittografare oggetti crittografati V1 e V2. [Questo client fa parte del pacchetto Amazon.Extensions.S3.Encryption](#). NuGet Esegui i seguenti passaggi su ciascuna delle tue applicazioni per utilizzare il client di transizione v1.

1. [Assumi una nuova dipendenza dal pacchetto Amazon.Extensions.S3.Encryption](#).
Se il tuo progetto dipende direttamente dal file .S3 o. AWSSDK AWSSDK KeyManagementServicepacchetti, è necessario aggiornare tali dipendenze o rimuoverle in modo che le loro versioni aggiornate vengano inserite in questo nuovo pacchetto.
2. Modificate l'usingistruzione appropriata da `Amazon.S3.Encryption` a `Amazon.Extensions.S3.Encryption`, come segue:

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. Ricostruisci e ridistribuisce l'applicazione.

Il client di transizione V1 è completamente compatibile con l'API con il client V1, quindi non sono necessarie altre modifiche al codice.

Esegui la migrazione dei client di transizione V1 ai client V2 per scrivere nuovi formati

[Il client V2 fa parte del pacchetto Amazon.Extensions.S3.Encryption](#). NuGet Consente alle applicazioni di decrittografare gli oggetti crittografati V1 e V2 (se configurato per tale operazione), ma crittografa gli oggetti solo in formato compatibile con V2.

Dopo aver aggiornato i client esistenti per leggere il nuovo formato di crittografia, è possibile procedere all'aggiornamento sicuro delle applicazioni ai client di crittografia e decrittografia V2. Esegui i seguenti passaggi su ciascuna delle tue applicazioni per utilizzare il client V2:

1. Passare da `EncryptionMaterials` a `EncryptionMaterialsV2`.
 - a. Quando si utilizza KMS:
 - i. Fornisci un ID chiave KMS.
 - ii. Dichiarare il metodo di crittografia che stai utilizzando, ovvero `KmsType.KmsContext`
 - iii. Fornisci a KMS un contesto di crittografia da associare a questa chiave dati. Puoi inviare un dizionario vuoto (il contesto di crittografia Amazon verrà comunque inserito), ma è consigliabile fornire un contesto aggiuntivo.
 - b. Quando si utilizzano metodi di key wrap forniti dall'utente (crittografia simmetrica o asimmetrica):
 - i. Fornisci un'istanza AES o un'RSAistanza che contenga i materiali di crittografia.
 - ii. Dichiarare quale algoritmo di crittografia utilizzare; ovvero, `SymmetricAlgorithmType.AesGcm` o `AsymmetricAlgorithmType.RsaOaepSha1`.
2. Passa `AmazonS3CryptoConfiguration` a `AmazonS3CryptoConfigurationV2` con la `SecurityProfile` proprietà impostata su `SecurityProfile.V2AndLegacy`
3. Passare da `AmazonS3EncryptionClient` a `AmazonS3EncryptionClientV2`. Questo client utilizza gli `EncryptionMaterialsV2` oggetti appena convertiti `AmazonS3CryptoConfigurationV2` e quelli dei passaggi precedenti.

Esempio: da KMS a KMS+Context

Pre-migrazione

```
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);
```

Post-migrazione

```
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);
```

Esempio: algoritmo simmetrico (portachiavi da AES-CBC a AES-GCM)

StorageMode può essere ObjectMetadata o InstructionFile.

Pre-migrazione

```
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);
```

Post-migrazione

```
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

Quando decifrate con AES-GCM, leggete l'intero oggetto fino alla fine prima di iniziare a utilizzare i dati decrittografati. Questo serve a verificare che l'oggetto non sia stato modificato da quando è stato crittografato.

Esempio: algoritmo asimmetrico (Key Wrap da RSA a RSA-OAEP-SHA1)

StorageMode può essere ObjectMetadata o InstructionFile.

Pre-migrazione

```
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);
```

Post-migrazione

```
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);
```

Aggiorna i client V2 in modo che non leggano più i formati V1

Alla fine, tutti gli oggetti saranno stati crittografati o ricrittografati utilizzando un client V2. Al termine di questa conversione, è possibile disabilitare la compatibilità V1 nei client V2 impostando la `SecurityProfile` proprietà su `SecurityProfile.V2`, come mostrato nel frammento seguente.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Considerazioni speciali per ilAWS SDK for .NET

Questa sezione contiene considerazioni per casi speciali in cui le normali configurazioni o procedure non sono appropriate o sufficienti.

Argomenti

- [Ottenerne assieme per AWS SDK for .NET](#)
- [Accesso a credenziali e profili in un'applicazione](#)
- [Considerazioni speciali per il supporto di Unity](#)
- [Considerazioni speciali per il supporto Xamarin](#)

Ottenerne assieme per AWS SDK for .NET

Questo argomento descrive come ottenere gli AWSSDK assembly e archivarli localmente (o in locale) per utilizzarli nei progetti. Questo non è il metodo consigliato per gestire i riferimenti SDK, ma è obbligatorio in alcuni ambienti.

Note

Il metodo consigliato per gestire i riferimenti SDK consiste nel scaricare e installare solo i NuGet pacchetti necessari a ciascun progetto. Questo metodo è descritto in [Installa AWSSDK pacchetti con NuGet](#).

Se non puoi o non sei autorizzato a scaricare e installare NuGet pacchetti per progetto, sono disponibili le seguenti opzioni.

Scarica ed estrai i file ZIP

(Ricorda che questo non è il [metodo consigliato](#) per gestire i riferimenti aAWS SDK for .NET.)

1. Scaricate uno dei seguenti file ZIP:

- [aws-sdk-net8.0.zip](#): assieme che supportano .NET 8 e versioni successive.
- [aws-sdk-netcoreapp3.1.zip](#) - Assieme che supportano .NET Core 3.1 e versioni successive.
- [aws-sdk-netstandard2.0.zip](#) - Assieme che supportano .NET Standard 2.0 e 2.1.

- [aws-sdk-net45.zip](#) - Assembly che supportano .NET Framework 4.5 e versioni successive.
- [aws-sdk-net35.zip](#) - Assiemi che supportano .NET Framework 3.5.

Warning

A partire dal 15 agosto 2024, AWS SDK for .NET terminerà il supporto per .NET Framework 3.5 e cambierà la versione minima .NET Framework alla 4.6.2. Per ulteriori informazioni, consulta il post di blog [Importanti modifiche in arrivo per gli obiettivi .NET Framework 3.5 e 4.5](#) di AWS SDK for .NET

2. Estrai gli assembly in una cartella «download» del tuo file system; non importa dove. Prendete nota di questa cartella.
3. Quando impostate il progetto, ottenete gli assiemi necessari da questa cartella, come descritto in [Installa gli assembly AWSSDK senza NuGet](#)

Accesso a credenziali e profili in un'applicazione

Il metodo preferito per utilizzare le credenziali consiste nel consentire loro AWS SDK for .NET di trovarle e recuperarle automaticamente, come descritto in [Risoluzione di credenziali e profili](#)

Tuttavia, è anche possibile configurare l'applicazione per recuperare attivamente profili e credenziali e quindi utilizzare esplicitamente tali credenziali durante la creazione di un client di servizio. AWS

[Per recuperare attivamente profili e credenziali, utilizza le classi di Amazon.Runtime.CredentialManagement](#) spazio dei nomi.

- Per trovare un profilo in un file che utilizza il formato di file delle AWS credenziali (il file delle [AWS credenziali condivise nella posizione predefinita o un file](#) di credenziali personalizzato), utilizzate la classe. [SharedCredentialsFile](#) I file in questo formato vengono talvolta chiamati semplicemente file di credenziali in questo testo per brevità.
- [Per trovare un profilo nell'SDK Store, utilizzate la classe NetSDK.CredentialsFile](#)
- Per cercare sia in un file di credenziali che nell'SDK Store, a seconda della configurazione della proprietà di una classe, utilizzate la classe. [CredentialProfileStoreChain](#)

Puoi usare questa classe per trovare i profili. È inoltre possibile utilizzare questa classe per richiedere direttamente AWS le credenziali anziché utilizzare la `AWSCredentialsFactory` classe (descritta di seguito).

- Per recuperare o creare vari tipi di credenziali da un profilo, utilizzate la classe.

[AWSCredentialsFactory](#)

Le sezioni seguenti forniscono esempi per queste classi.

Esempi di classe `CredentialProfileStoreChain`

È possibile ottenere credenziali o profili dalla [CredentialProfileStoreChain](#) classe utilizzando i [TryGetProfile](#) metodi [TryGetAWSCredentials](#) o `r`. La `ProfilesLocation` proprietà della classe determina il comportamento dei metodi, come segue:

- Se `ProfilesLocation` è nullo o vuoto, cerca nell'SDK Store se la piattaforma lo supporta, quindi cerca nel file delle AWS credenziali condivise nella posizione predefinita.
- Se la `ProfilesLocation` proprietà contiene un valore, cerca nel file delle credenziali specificato nella proprietà.

Ottieni le credenziali dall'SDK Store o dal file delle credenziali condivise AWS

[Questo esempio mostra come ottenere le credenziali utilizzando la CredentialProfileStoreChain classe e quindi utilizzare le credenziali per creare un oggetto AmazonS3Client.](#) Le credenziali possono provenire dall'SDK Store o dal file delle credenziali condivise nella posizione predefinita. AWS

[Questo esempio utilizza anche Amazon.Runtime.AWSCredentials classe.](#)

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

```
}
```

Ottieni un profilo dall'SDK Store o dal file di AWS credenziali condivise

Questo esempio mostra come ottenere un profilo utilizzando la `CredentialProfileStoreChain` classe. Le credenziali possono provenire dall'SDK Store o dal file delle AWS credenziali condivise nella posizione predefinita.

Anche questo esempio utilizza la classe. [CredentialProfile](#)

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Ottieni credenziali da un file di credenziali personalizzato

Questo esempio mostra come ottenere le credenziali utilizzando la classe. `CredentialProfileStoreChain` Le credenziali provengono da un file che utilizza il formato di file delle AWS credenziali ma si trova in una posizione alternativa.

[Questo esempio utilizza anche Amazon.Runtime.AWSCredentials](#) classe.

```
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
}
```

Esempi di classi SharedCredentialsFile e AWSCredentialsFactory

Crea un client AmazonS3 utilizzando la classe SharedCredentialsFile

[Questo esempio mostra come trovare un profilo nel file delle AWS credenziali condivise, creare credenziali dal profilo e quindi utilizzare AWS le credenziali per creare un oggetto AmazonS3Client.](#) [SharedCredentialsFile](#) L'esempio utilizza la classe.

Questo esempio utilizza anche la [CredentialProfile](#) classe e [Amazon.Runtime.AWSCredentials](#) classe.

```
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

La [CredentialsFile](#) classe [NetSDK](#) può essere utilizzata esattamente allo stesso modo, tranne per il fatto che è necessario istanziare un nuovo oggetto [NetSDK CredentialsFile](#) anziché un oggetto [SharedCredentialsFile](#)

Considerazioni speciali per il supporto di Unity

Quando si utilizza [e.NET Standard 2.0](#) per l'applicazione Unity, l'applicazione deve fare riferimento direttamente AWS SDK for .NET agli assembly (file DLL) anziché utilizzarli. AWS SDK for .NET NuGet In base a questo requisito, è necessario eseguire le seguenti azioni importanti.

- È necessario procurarsi gli AWS SDK for .NET assieme e applicarli al progetto. Per informazioni su come eseguire questa operazione, consultate l'[Scarica ed estrai i file ZIP](#) argomento. [Ottenere assieme AWSSDK](#)
- Devi includere le seguenti DLL nel tuo progetto Unity insieme alle DLL [AWSSDKper.Core](#) e agli altri AWS servizi che stai utilizzando. A partire dalla versione 3.5.109 di AWS SDK for .NET, il file ZIP standard .NET contiene queste DLL aggiuntive.
 - [Microsoft.Bcl.AsyncInterfaces.dll](#)

- [System.Runtime.CompilerServices.Unsafe.dll](#)
 - [System.Threading.Tasks.Extensions.dll](#)
- Se utilizzi [IL2CPP per creare il tuo progetto Unity, devi aggiungere un file](#) alla cartella Asset per evitare il code stripping. Il `link.xml` file deve elencare tutti gli AWSSDK assembly che stai utilizzando e ognuno deve includere l'attributo `preserve="all"` Il frammento seguente mostra un esempio di questo file.

```
<linker>
  <assembly fullname="AWSSDK.Core" preserve="all"/>
  <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
  <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

Per leggere interessanti informazioni di base relative a questo requisito, consultate l'articolo all'[indirizzo https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-uwp/](https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-uwp/). [from-unity-xamarin-or](#)

Oltre a queste considerazioni speciali, consulta [Cosa è cambiato per la versione 3.5](#) per informazioni sulla migrazione dell'applicazione Unity alla versione 3.5 di AWS SDK for .NET

Considerazioni speciali per il supporto Xamarin

I progetti Xamarin (nuovi ed esistenti) devono essere indirizzati a .NET Standard 2.0. Vedere il [supporto .NET Standard 2.0 in Xamarin.Forms](#) e il [supporto di implementazione .NET](#).

Vedi anche le informazioni su [Libreria di classi portabile e Xamarin](#).

Informazioni di riferimento sull'API diAWS SDK for .NET

LaAWS SDK for .NETfornisce un'API da utilizzare per l'accessoAWS Servizi . Per vedere quali classi e metodi sono disponibili nell'API, vedere il[AWS SDK for .NETDocumentazione di riferimento API](#).

Oltre al riferimento generale sopra indicato, ciascuno degli esempi sotto[Esempi di codice con indicazioni](#)la sezione contiene riferimenti ai metodi e alle classi specifici utilizzati in quell'esempio.

Cronologia dei documenti

La tabella seguente descrive le modifiche importanti rispetto all'ultima versione della AWS SDK for .NET Developer Guide. Per ricevere notifiche sugli aggiornamenti di questa documentazione, puoi sottoscrivere un [feed RSS](#).

Modifica	Descrizione	Data
Cosa c'è di nuovo	Sono state aggiunte informazioni sulla versione di anteprima del AWS Message Processing Framework per .NET	28 marzo 2024
Cosa c'è di nuovo	Informazioni incluse sul supporto per .NET 8.	23 febbraio 2024
Cosa c'è di nuovo	Sono incluse informazioni sulle modifiche imminenti al supporto di .NET Framework.	18 febbraio 2024
Ottenere AWSSDK assemblee	Sono incluse informazioni sugli assembly che supportano .NET 8 e versioni successive.	8 gennaio 2024
AWS Message Processing Framework per .NET	Informazioni incluse sulla versione beta del Message Processing Framework.	10 dicembre 2023
AWS OpsWorks	È stata aggiunta una nota su End of Life for AWS OpsWorks.	8 dicembre 2023
Utilizzo di database Amazon DynamoDB NoSQL	Informazioni aggiornate sui modelli di programmazione per la persistenza di documenti e oggetti. Ora è possibile prevenire determina	15 novembre 2023

	te condizioni di latenza o deadlock dovute a comportamenti di avvio a freddo e di threadpool.	
Sono stati inclusi ulteriori aggiornamenti delle best practice IAM	Guida aggiornata per l'allineamento alle best practice IAM. Per ulteriori informazioni, consulta Best practice per la sicurezza in IAM .	5 ottobre 2023
Ottenere assieme AWSSDK	Sono state rimosse le informazioni sull'installazione di AWS SDK for .NET utilizzando il programma di Strumenti AWS per Windows installazione (ovvero MSI), che è diventato obsoleto.	25 settembre 2023
Aggiornamenti delle best practice di IAM	Guida aggiornata per l'allineamento alle best practice IAM. Per ulteriori informazioni, consulta Best practice per la sicurezza in IAM .	18 luglio 2023
Annotazioni Lambda	Il framework AWS Lambda Annotations è stato rilasciato per la disponibilità generale.	17 luglio 2023
Cosa c'è di nuovo	Sono state aggiunte informazioni sulla versione di anteprima del Distributed Cache Provider for DynamoDB.	15 luglio 2023
Sommaro	Sommario aggiornato per rendere gli esempi di codice più facilmente individuabili.	8 giugno 2023

Risoluzione della regione	Sono state aggiunte informazioni su come l'SDK risolve una specifica regionale mancante.	14 marzo 2023
Support per MSI	È stata aggiunta una nota sulla fine del supporto per il Strumenti AWS per Windows programma di installazione.	6 marzo 2023
Annotazioni Lambda (anteprima)	Anteprima del framework AWS Lambda Annotations.	22 settembre 2022
Distribuisce le applicazioni su AWS	Contenuto principale spostato in un sito GitHub Pages: https://aws.github.io//aws-dotnet-deploy	28 giugno 2022
Ritiro di EC2-Classic	Aggiunte note relative al ritiro di EC2-Classic.	13 aprile 2022
Single Sign-On con AWS SDK for .NET	Sono state aggiunte informazioni sul Single Sign-On (SSO) durante l'utilizzo di AWS SDK for .NET	17 marzo 2022
Applicazione di una versione TLS minima	Sono state aggiunte informazioni su TLS 1.3.	16 marzo 2022
Lavora con i servizi AWS	Elenchi inclusi degli esempi di codice disponibili su GitHub.	28 febbraio 2022
Abilitazione delle metriche SDK	Sono state rimosse le informazioni sull'abilitazione delle metriche SDK, che sono diventate obsolete.	20 gennaio 2022

Distribuisce le applicazioni su AWS	È stato aggiunto un riferimento al AWS Toolkit for Visual Studio, che fornisce funzionalità di distribuzione simili a AWS Deploy Tool.	26 ottobre 2021
AWS SDK for .NET versione 3: guida al consolidamento.	Le due guide per sviluppatori della AWS SDK for .NET versione 3, «V3» e «più recente», sono state consolidate in un'unica guida sotto l'URL «v3».	18 agosto 2021
Migrazione da .NET Standard 1.3	Il supporto per .NET Standard 1.3 su the AWS SDK for .NET è giunto al termine.	25 marzo 2021
Distribuisce le applicazioni su AWS (anteprima)	Sono state aggiunte informazioni di anteprima sullo strumento AWS Deploy Tool, che è possibile utilizzare per distribuire un'applicazione da .NET CLI.	15 marzo 2021
La versione 3.5 di AWS SDK for .NET	AWS SDK for .NET È stata rilasciata la versione 3.5 di.	25 agosto 2020
Impaginatori	Sono stati aggiunti degli impaginatori a molti client di servizio, che semplificano l'impaginazione dei risultati delle API.	24 agosto 2020
Tentativi e timeout	Sono state aggiunte informazioni sulle modalità di riprova.	20 agosto 2020

Migrazione del client di crittografia S3	Sono state aggiunte informazioni su come migrare i client di crittografia Amazon S3 da V1 a V2.	7 agosto 2020
Utilizzo delle chiavi KMS per la crittografia S3	Esempio aggiornato per utilizzare la versione 2 del client di crittografia S3.	6 agosto 2020
Migrazione da .NET Standard 1.3	Sono state aggiunte informazioni su come terminare il supporto per .NET Standard 1.3 alla fine del 2020.	18 maggio 2020
Avvio rapido	Aggiunta di una sezione di avvio rapido con tutorial e configurazione di base per introdurre il lettore a AWS SDK for .NET.	27 marzo 2020
Applicazione del TLS 1.2	Aggiunte informazioni su come applicare TLS 1.2 nell'SDK.	10 marzo 2020

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.