aws

Guide du développeur

AWS SDK pour .NET (V4)

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

AWS SDK pour .NET (V4): Guide du développeur

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

Les marques commerciales et la présentation commerciale d'Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d'une manière susceptible d'entraîner une confusion chez les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

..... x
Qu'est-ce que le AWS SDK pour .NET 1
À propos de cette version 1
Maintenance et prise en charge des versions majeures du SDK 2
Cas d’utilisation courants 2
Sujets supplémentaires dans cette section 2
AWS Outils connexes 3
Outils pour Windows PowerShell et outils pour PowerShell Core 3
Toolkit for VS Code 3
Toolkit for Visual Studio 3
Boîte à outils pour Azure DevOps 4
Plateformes prises en charge 4
.NET Core 4
.NET Standard 2.0 5
.NET Framework 4.5 5
.NET Framework 3.5 5
Bibliothèque de classes portable et Xamarin 5
Support Unity 5
SDKs et référence sur les outils 5
Historique des révisions 6
Nouveautés 6
Ressources supplémentaires 8
Premiers pas 11
Installation et configuration de votre chaîne d'outils 11
Développement multiplateforme 11
Windows avec Visual Studio et .NET Core 12
Étape suivante 12
Authentification avec AWS 12
Connectez-vous en utilisant les informations d'identification de console 13
Activation et configuration d’IAM Identity Center 14
Configurez le SDK pour utiliser IAM Identity Center. 14
Démarrer une session sur le portail AWS d'accès 16
Informations supplémentaires 17
Création d'une application simple 17

Application multiplateforme simple 18
Application simple basée sur Windows 23
Étapes suivantes 29
Configuration 30
Lancement d'un nouveau projet 30
Installez AWSSDK des packages avec NuGet 32
Utilisation NuGet à partir de l'invite de commande ou du terminal 33
Utilisation NuGet depuis l'explorateur de solutions Visual Studio 33
Utilisation NuGet depuis la console Package Manager 34
Installez AWSSDK des assemblages sans NuGet 35
AWS Région 36
Création d'un client de service avec une région particulière 37
Spécifiez une région pour tous les clients du service 38
Résolution de la région 39
Informations spéciales sur la région de Chine (Pékin) 39
Informations spéciales sur les nouveaux AWS services 39
Résolution des informations d'identification et des profils 40
Résolution du profil 41
Utilisation des informations d'identification d'un compte utilisateur fédéré 41
Spécifier des rôles ou des informations d'identification temporaires 42
Utilisation des informations d'identification du proxy 43
Rétentatives et délais 43
Nouvelles tentatives 44
Délais 46
Observabilité 48
Ressources supplémentaires 48
Configurez un `TelemetryProvider` 49
Métriques 51
Fournisseurs de services de télémétrie 52
Utilisateurs et rôles 54
Utilisateurs et ensembles d'autorisations 54
Rôles de service 55
Configuration avancée 56
AWSSDK.Prolongations. NETCore.Configuration et IConfiguration 56
Configuration d'autres paramètres de l'application 61
Référence des fichiers de configuration pour AWS SDK pour .NET 69

Utilisation d’informations d'identification existantes 81
Avertissements et conseils importants concernant les informations d'identification 82
Utilisation du fichier AWS d'informations d'identification partagé 83
Utilisation du SDK Store (Windows uniquement) 87
Utilisation de l'SDK 92
Programmation asynchrone 92
Pagination 94
Où puis-je trouver des paginateurs ? 94
Que m'apportent les paginateurs ? 94
Pagination synchrone ou asynchrone 95
exemple 95
Considérations supplémentaires pour les paginateurs 99
Support pour HTTP 2 99
Considérations supplémentaires 102
Outils supplémentaires 102
AWS Outil de déploiement 102
AWS Framework de traitement des messages pour .NET 103
Intégrations avec .NET Aspire 103
Authentification avancée 104
Authentification unique 104
Prérequis 105
Configuration d'un profil SSO 105
Génération et utilisation de jetons SSO 107
Ressources supplémentaires 112
Didacticiels 112
Tutoriel : application .NET uniquement 113
Tutoriel : AWS CLI et application .NET 122
Déployer vers AWS 131
Déploiement à partir de la CLI .NET 131
Déployez à partir des boîtes à outils de l'IDE 131
Cas d’utilisation 132
Applications ASP.NET Core 132
Applications de console .NET 133
Applications Blazor WebAssembly 134
AWS Lambda projets 135
Prérequis 135

Commandes Lambda disponibles 136
Étapes de déploiement 136
AWS Services d'appels 138
Exemples de code guidés 138
CloudFormation 139
Amazon Cognito 143
DynamoDB 152
Amazon EC2 182
IAM 244
Amazon S3 264
Amazon SNS 274
Amazon SQS 278
AWS Lambda 312
APIs 312
Prérequis 312
Informations supplémentaires 312
Rubriques 312
Annotations Lambda 313
Bibliothèques et frameworks de haut niveau 315
Cadre de traitement des messages 315
Intégration AWS à .NET Aspire 336
Autres services et configuration 337
Exemples de code 338
Aurora 339
Mise en route 339
Principes de base 341
Actions 366
Auto Scaling 381
Mise en route 339
Principes de base 341
Actions 366
Amazon Bedrock 417
Mise en route 339
Actions 366
Exécution d’Amazon Bedrock 421
Anthropic Claude 421

Cohere Command 427
Meta Llama 431
Mistral AI 434
CloudFormation 438
Mise en route 339
CloudWatch 442
Mise en route 339
Principes de base 341
Actions 366
CloudWatch Journaux 500
Actions 366
Scénarios 504
Fournisseur d'identité Amazon Cognito 526
Actions 366
AWS Control Tower 527
Mise en route 339
Principes de base 341
Actions 366
DynamoDB 567
Mise en route 339
Principes de base 341
Actions 366
Amazon EC2 607
Mise en route 339
Amazon ECS 609
Mise en route 339
AWS IoT 612
Mise en route 339
Principes de base 341
Actions 366
AWS IoT data 671
Actions 366
Amazon Redshift 674
Mise en route 339
Principes de base 341
Actions 366

Amazon S3 705
Mise en route 339
Principes de base 341
Actions 366
Scénarios 504
Migration de votre projet 742
Migration vers la version 4 742
.NET Framework 742
Types de valeur 743
Collections 743
AWS Security Token Service (STS) 744
Changements liés à `ClientConfig` 744
Les AWSSDK .Extensions. NETCore.Package d'installation NuGet 745
`CookieSigner` et `UrlSigner` 746
DateTime contre UTC DateTime 746
DateTime analyse syntaxique 747
`ConvertFromUnixEpochSeconds` et `ConvertFromUnixEpochMilliseconds` 747
Logging 747
Support pour HTTP 2 748
Authentification unique 748
Modifications spécifiques à DynamoDB 748
Modifications spécifiques à EC2 752
Modifications spécifiques à S3 753
Éléments de programmation qui ont été supprimés 757
Migration à partir de .NET Standard 1.3 760
Sécurité 762
Protection des données 763
Gestion de l’identité et des accès 764
Public ciblé 764
Authentification par des identités 765
Gestion de l’accès à l’aide de politiques 766
Comment Services AWS travailler avec IAM 768
Résolution des problèmes AWS d'identité et d'accès 769
Validation de la conformité 771
Résilience 771
Sécurité de l’infrastructure 772

Application d’une version minimale de TLS 772
.NET Core 773
.NET Framework 774
Outils AWS pour PowerShell 775
Xamarin 776
Unity 776
Navigateur (pour Blazor WebAssembly) 776
Migration du client de chiffrement S3 (V1 vers V2) 777
Présentation de la migration 777
Mettez à jour les clients existants vers des clients de transition vers la version 1 pour lire les nouveaux formats 778
Migrer les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats 779
Mettre à jour les clients V2 pour ne plus lire les formats V1 782
Migration du client de chiffrement S3 (V2 vers V4) 783
Comprendre les concepts de la V4 783
Mettre à jour les clients existants pour lire les formats V4 785
Migrer vers des clients V4 786
Exemples de configuration supplémentaires 790
Considérations spéciales 793
Obtention d' AWSSDK assemblages 793
Téléchargez et extrayez des fichiers ZIP 793
Accès aux informations d'identification et aux profils dans une application 794
Exemples de cours CredentialProfileStoreChain 795
Exemples pour les classes SharedCredentialsFile et AWSCredentials Factory 796
Support Unity 797
Assistance Xamarin 798
Référence d’API 799
À propos des versions de référence de l'API 799
Historique du document 802

La version 4 (V4) du AWS SDK pour .NET est sortie !

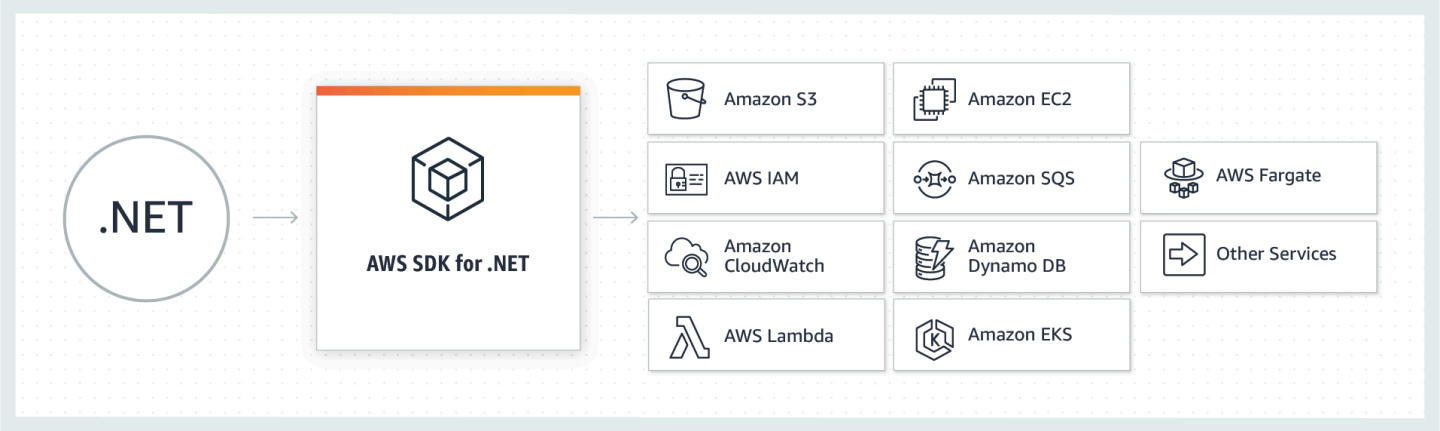
Pour plus d'informations sur les modifications majeures et la migration de vos applications, consultez la rubrique relative à la migration.

Click here for details

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.

Qu'est-ce que le AWS SDK pour .NET

AWS SDK pour .NET Cela facilite la création d'applications .NET qui exploitent des AWS services rentables, évolutifs et fiables tels qu'Amazon Simple Storage Service (Amazon S3) et Amazon Elastic Compute Cloud (Amazon EC2). Le SDK simplifie l'utilisation des AWS services en fournissant un ensemble de bibliothèques cohérentes et familières aux développeurs .NET.



(OK, j'ai compris ! Je suis prêt à m'installer et à faire une visite rapide.)

À propos de cette version

Documentation actuelle et passée

Cette documentation concerne les versions 4.0 et ultérieures (V4) du AWS SDK pour .NET. Il est principalement centré sur .NET Core et ASP.NET Core, mais contient également des informations sur .NET Framework et ASP.NET 4. x. Outre Windows et Visual Studio, il accorde une attention égale au développement multiplateforme.

Pour plus d'informations sur la migration à partir de versions antérieures du SDK, consultez. Migration de votre projet

Pour plus d'informations sur la version précédente du SDK, V3, qui sera obsolète à l'avenir, consultez le guide du développeur AWS SDK pour .NET (V3).

Pour trouver le contenu obsolète des versions antérieures du SDK, consultez les éléments suivants :

- AWS SDK pour .NET (version 2, obsolète) Guide du développeur

- Références d'API obsolètes pour AWS SDK pour .NET

Maintenance et prise en charge des versions majeures du SDK

Pour plus d'informations sur la maintenance et le support des versions majeures du SDK et de leurs dépendances sous-jacentes, consultez les informations suivantes dans le guide de référence AWS SDKs and Tools :

- AWS SDKs et politique de maintenance des outils
- AWS SDKs et matrice de support des versions d'outils

Cas d’utilisation courants

AWS SDK pour .NET Cela vous aide à réaliser plusieurs cas d'utilisation convaincants, notamment les suivants :

- Gérez les utilisateurs et les rôles avec Gestion des identités et des accès AWS (IAM).
- Accédez à Amazon Simple Storage Service (Amazon S3) pour créer des buckets et stocker des objets.
- Gérez les abonnements HTTP Amazon Simple Notification Service (Amazon SNS) aux rubriques.
- Utilisez l'utilitaire de transfert S3 pour transférer des fichiers vers Amazon S3 depuis vos applications Xamarin.
- Utilisez Amazon Simple Queue Service (Amazon SQS) pour traiter les messages et les flux de travail entre les composants d'un système.
- Effectuez des transferts Amazon S3 efficaces en envoyant des instructions SQL à Amazon S3 Select.
- Créez et lancez EC2 des instances Amazon, configurez et demandez des instances Amazon EC2 Spot.

Sujets supplémentaires dans cette section

- AWS outils liés à AWS SDK pour .NET

- Plateformes prises en charge par le AWS SDK pour .NET
- AWS SDKs Guide de référence et d'outils
- Historique des révisions
- Quoi de neuf dans le AWS SDK pour .NET
- Ressources supplémentaires

AWS outils liés à AWS SDK pour .NET

Outils pour Windows PowerShell et outils pour PowerShell Core

Les AWS Tools for Windows PowerShell et AWS Tools for PowerShell Core sont PowerShell des modules basés sur les fonctionnalités exposées par le AWS SDK pour .NET. Les AWS PowerShell outils vous permettent de créer des scripts pour des opérations sur vos AWS ressources à partir de l' PowerShell invite. Bien que les applets de commande soient implémentés à l'aide des clients de service et des méthodes du SDK, les applets de commande fournissent une PowerShell expérience idiomatique pour la spécification des paramètres et la gestion des résultats.

Consultez AWS Tools for Windows PowerShell pour démarrer.

Toolkit for VS Code

Le AWS Toolkit for Visual Studio Code est un plugin pour l'éditeur de code Visual Studio (VS Code). La boîte à outils facilite le développement, le débogage et le déploiement d'applications qui utilisent AWS.

Avec la boîte à outils, vous pouvez faire notamment les choses suivantes :

- Créez des applications sans serveur contenant des AWS Lambda fonctions, puis déployez les applications sur une AWS CloudFormation pile.
- Travaillez avec les EventBridge schémas Amazon.
- IntelliSense À utiliser lorsque vous travaillez avec des fichiers de définition de tâches Amazon ECS.
- Visualisez une AWS Cloud Development Kit (AWS CDK) application.

Toolkit for Visual Studio

Il AWS Toolkit for Visual Studio s'agit d'un plugin pour l'IDE Visual Studio qui facilite le développement, le débogage et le déploiement d'applications .NET utilisant Amazon Web Services.

Le Toolkit for Visual Studio fournit des modèles Visual Studio pour des services tels que Lambda et des assistants de déploiement pour les applications Web et les applications sans serveur. Vous pouvez utiliser l' AWS explorateur pour gérer les EC2 instances Amazon, utiliser les tables Amazon DynamoDB, publier des messages dans les files d'attente Amazon Simple Notification Service (Amazon SNS), etc., le tout dans Visual Studio.

Pour commencer, voir Configuration du AWS Toolkit for Visual Studio.

Boîte à outils pour Azure DevOps

AWS Toolkit pour Microsoft Azure DevOps Ajoute des tâches pour permettre facilement aux pipelines de création et de publication dans Azure DevOps et Azure DevOps Server de fonctionner avec AWS les services. Vous pouvez travailler avec Amazon S3,, AWS Elastic Beanstalk AWS CodeDeploy, Lambda AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) et Amazon SNS. Vous pouvez également exécuter des commandes à l'aide du PowerShell module Windows et du AWS Command Line Interface (AWS CLI).

Pour commencer à utiliser le AWS Toolkit pour Azure DevOps, consultez le guide de AWS Toolkit pour Microsoft Azure DevOps l'utilisateur.

Plateformes prises en charge par le AWS SDK pour .NET

AWS SDK pour .NET Il fournit des groupes distincts d'assemblages permettant aux développeurs de cibler différentes plateformes. Toutefois, toutes les fonctionnalités SDK ne sont pas identiques sur toutes ces plateformes. Cette rubrique aborde les différences de prise en charge pour chaque plateforme.

.NET Core

Il AWS SDK pour .NET prend en charge les applications écrites pour .NET Core (.NET Core 3.1, .NET 5, .NET 6, etc.). AWS les clients de service prennent uniquement en charge les modèles d'appel asynchrones dans .NET Core. Cela affecte également de nombreuses abstractions de haut niveau créées à partir de clients de service, comme Amazon S3`TransferUtility`, qui ne prend en charge que les appels asynchrones dans l'environnement .NET Core.

.NET Standard 2.0

Les variantes non liées au framework AWS SDK pour .NET sont conformes au .NET Standard 2.0. AWS SDK pour .NET fournit uniquement des méthodes asynchrones pour les applications écrites selon le standard .NET.

.NET Framework 4.5

Cette version AWS SDK pour .NET est compilée avec .NET Framework 4.7.2 et s'exécute dans le runtime .NET 4.0. AWS les clients de service prennent en charge les modèles d'appel synchrones et asynchrones et utilisent les mots clés async et await introduits dans C# 5.0.

.NET Framework 3.5

La version 4 AWS SDK pour .NET ne prend pas en charge .NET Framework 3.5.

Bibliothèque de classes portable et Xamarin

AWS SDK pour .NET Il contient également une implémentation de bibliothèque de classes portable. L'implémentation de la bibliothèque de classes portable peut cibler plusieurs plateformes, notamment Universal Windows Platform (UWP) et Xamarin sur iOS et Android. Consultez le SDK mobile pour .NET et Xamarin and Xamarin pour plus de détails. AWS les clients de service ne prennent en charge que les modèles d'appel asynchrones.

Support Unity

Pour plus d'informations sur le support Unity, consultezConsidérations spéciales relatives au support Unity.

AWS SDKs Guide de référence et d'outils

Le guide de référence sur les outils AWS SDKs et contient des informations pertinentes et importantes pour de nombreuses boîtes à outils AWS SDKs et les AWS CLI. Voici quelques exemples des informations contenues dans la référence :

- Informations sur les `credentials`fichiers partagés AWS`config` et leur emplacement.
- Configuration AWS des comptes, des utilisateurs et des rôles
- Référence des paramètres de configuration et d'authentification

- AWS bibliothèques CRT (Common Runtime)
- AWS SDKs et politique de maintenance des outils
- AWS SDKs et matrice de support des versions d'outils

Historique des révisions

Pour découvrir ce qui a changé dans les différentes versions, consultez ce qui suit :

- Journaux des modifications apportées au SDK
- Quoi de neuf dans le AWS SDK pour .NET
- Historique du document

Quoi de neuf dans le AWS SDK pour .NET

Pour obtenir des informations de haut niveau sur les nouveaux développements liés au AWS SDK pour .NET, consultez la page du produit https://aws.amazon.com/sdk-for-net/et les journaux des modifications du SDK.

Voici les nouveautés du AWS SDK pour .NET.

17 octobre 2025 : version 2.0 de l'outil de AWS déploiement

La version 2.0 de l'outil de AWS déploiement pour .NET CLI a été publiée. Pour plus d'informations sur l'outil de déploiement, consultezDéployez des applications pour AWS. Pour plus d'informations sur la version 2.0 de l'outil de déploiement, consultez le billet de blog Nouveautés de l'outil de AWS déploiement pour .NET.

21 août 2025 : Prochainement end-of-support pour la version 3 du AWS SDK pour .NET

La version 3 (V3) de end-of-support for AWS SDK pour .NET a été annoncée. Consultez le guide de migration pour migrer vos applications V3 et éviter les interruptions. Pour plus d'informations, consultez le billet de blog annonçant la sortie end-of-support de la AWS SDK pour .NET version 3.

2 juillet 2025 : Disponibilité générale du fournisseur de cache distribué

Le fournisseur de cache distribué AWS .NET pour Amazon DynamoDB est généralement disponible ! Ce fournisseur implémente l'interface ASP.NET Core IDistributedCache, qui vous permet d'intégrer

l'infrastructure durable et entièrement gérée de DynamoDB dans votre couche de mise en cache avec un minimum de modifications de code. Le fournisseur de cache distribué est disponible via AWS. AspNetCore. DistributedCacheProvideremballage sur NuGet. Pour plus d'informations, consultez le billet de blog Le fournisseur de cache distribuéAWS .NET pour Amazon DynamoDB est désormais généralement disponible.

> **Note**
>
> La version publiée du fournisseur de cache distribué n'est prise en charge que sur la version 4 du AWS SDK pour .NET. Si vous avez essayé la version préliminaire du fournisseur de cache distribué, vous devrez peut-être mettre à jour les dépendances des packages dans votre application.

5 mai 2025 : Disponibilité générale du cadre de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET est généralement disponible ! Le framework est un framework AWS natif qui simplifie le développement d'applications de traitement de messages .NET utilisant des AWS services tels qu'Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) et Amazon. EventBridge

28 avril 2025 : Version 4 du AWS SDK pour .NET

La version 4 du AWS SDK pour .NET est généralement disponible ! Pour plus d'informations sur la migration de vos applications vers la version V4, consultezMigration vers la version 4. Consultez également le billet de blog sur la disponibilité générale de la AWS SDK pour .NET version 4.0 qui annonce la disponibilité générale.

15 février 2025 : Intégrations avec .NET Aspire

Des intégrations avec .NET Aspire visant à améliorer la boucle de développement interne ont été publiées. Pour plus d'informations, consultez Intégration AWS à .NET Aspire dans le AWS SDK pour .NET.

10 février 2025 : sortie de l'AG pour l'observabilité

L'observabilité est la mesure dans laquelle l'état actuel d'un système peut être déduit des données qu'il émet. L'observabilité a été ajoutée au AWS SDK pour .NET, y compris la mise en œuvre d'un fournisseur de télémétrie. Pour plus d'informations, consultez Observabilité ce guide et le billet de blog annonçant la disponibilité générale des OpenTelemetry bibliothèques AWS .NET.

15 janvier 2025 : Nouveau comportement par défaut pour la protection de l'intégrité

À partir de la version 3.7.412.0 du AWS SDK pour .NET, le SDK fournit des protections d'intégrité par défaut en calculant automatiquement une somme de contrôle pour les téléchargements. `CRC32` Pour plus d'informations, voir l' GitHub annonce surhttps://github.com/aws/aws-sdk-net/issues/3610. Le SDK fournit également des paramètres globaux pour les protections de l'intégrité des données que vous pouvez définir en externe. Vous pouvez en savoir plus sur les protections de l'intégrité des données dans le guide de référence sur les outils AWS SDKs et.

15 novembre 2024 : article de blog pour la version 4, sortie de la version 4

La version 4 de la AWS SDK pour .NET version 4 a été publiée le 15 novembre 2024. Pour plus d'informations sur cet aperçu, consultez le billet de blog Preview 4 of AWS SDK pour .NET V4.

16 août 2024 : article de blog pour la version préliminaire 1, sortie de la version 4

La AWS SDK pour .NET version 4 a été publiée en tant que premier aperçu le 16 août 2024. Pour plus d'informations sur cet aperçu, consultez le billet de blog Aperçu 1 de la version AWS SDK pour .NET 4.

Ressources supplémentaires

Services pris en charge

Il AWS SDK pour .NET prend en charge la plupart des produits d' AWS infrastructure, et de nouveaux services sont ajoutés fréquemment. Pour obtenir la liste des AWS services pris en charge par le SDK, consultez le fichier README du SDK.

Page d'accueil du AWS SDK pour .NET

Pour plus d'informations sur le AWS SDK pour .NET, consultez la page d'accueil du SDK à https://aws.amazon.com/sdk-for-net/l'adresse.

Documentation de référence du SDK

La documentation de référence du SDK vous permet de parcourir et de rechercher l'ensemble du code inclus dans le SDK. Il fournit une documentation complète et des exemples d'utilisation. Pour plus d’informations, consultez la page Référence de l’API AWS SDK pour .NET.

AWS Re:post (anciennement AWS forums)

Consultez AWS RE:Post, en particulier le sujet concerné AWS SDK pour .NET, pour poser des questions ou faire part de vos commentaires. AWS Chaque page de documentation contient un lien Try AWS Re:post en bas de page qui vous amène à la rubrique Re:Post associée. AWS les ingénieurs surveillent les sujets et répondent aux questions, aux commentaires et aux problèmes.

Si vous êtes connecté à Re:post, vous pouvez également suivre un sujet. Pour suivre le sujet du AWS SDK pour .NET, rendez-vous sur la page Tous les sujets, recherchez « .NET activé AWS », puis cliquez sur le bouton Suivre.

Trousses à outils

- AWS Toolkit for Visual Studio: Si vous utilisez l'IDE Microsoft Visual Studio, vous devez consulter le guide de l'AWS Toolkit for Visual Studio utilisateur.
- AWS Toolkit for Visual Studio Code: Si vous utilisez l'IDE Microsoft Visual Studio, vous devez consulter le guide de l'AWS Toolkit for Visual Studio Code utilisateur.

Bibliothèques, extensions et outils utiles

Consultez les aws-sdk-net référentiels aws/dotnet et aws/ du GitHub site Web pour obtenir des liens vers des bibliothèques, des outils et des ressources que vous pouvez utiliser pour créer des applications et des services .NET sur lesquels vous pouvez créer des applications et des services .NET. AWS

Voici quelques exemples :

- AWS Extension de configuration .NET pour Systems Manager
- AWS Configuration des extensions .NET Core
- AWS Journalisation .NET
- Amazon Cognito Authentication Extension Library
- Kit SDK AWS X-Ray pour .NET

Autres ressources

Voici d'autres ressources qui pourraient s'avérer utiles :

- Réseau des développeurs
- Environnement de développement .NET sur le AWS cloud - Déploiement de référence Quick Start

- Bonjour, Cloud ! blogue
- AWS Livre blanc : Développement et déploiement d'applications .NET sur AWS
- AWS Microservice Extractor for .NET
- Assistant de portage pour .NET
- AWS SDKs Guide de référence et d'outils

Commencer à utiliser le AWS SDK pour .NET

Découvrez comment installer, configurer et utiliser le AWS SDK pour .NET. Vous devez installer votre chaîne d'outils et configurer un certain nombre d'éléments essentiels dont votre application a besoin pour accéder aux AWS services. Il s'agit des licences suivantes :

- Un compte ou un rôle d'utilisateur approprié
- Informations d'authentification pour ce compte utilisateur ou pour assumer ce rôle

Pour plus d'informations sur la configuration de projets et d'autres éléments critiques, consultezConfiguration du AWS SDK pour .NET.

Rubriques

- Installation et configuration de votre chaîne d'outils pour AWS SDK pour .NET
- Authentifier le AWS SDK pour .NET avec AWS
- Création d'une application simple à l'aide du AWS SDK pour .NET

Installation et configuration de votre chaîne d'outils pour AWS SDK pour .NET

Pour utiliser le AWS SDK pour .NET, vous devez avoir installé certains outils de développement.

Développement multiplateforme

Les éléments suivants sont requis pour le développement .NET multiplateforme sous Windows, Linux ou macOS :

- Microsoft .NET Core SDK, version 2.1, 3.1 ou ultérieure, qui inclut l'interface de ligne de commande (CLI) NET (**`dotnet`**) et le moteur d'exécution .NET core.

- Un éditeur de code ou un environnement de développement intégré (IDE) adapté à votre système d'exploitation et à vos exigences. Il s'agit généralement d'une solution qui fournit un certain support pour .NET Core.

Les exemples incluent Microsoft Visual Studio Code (VS Code), JetBrains Rider et Microsoft Visual Studio.

- (Facultatif) Une AWS boîte à outils, si elle est disponible, pour l'éditeur que vous avez choisi et votre système d'exploitation.

  Les exemples incluent AWS Toolkit for Visual Studio CodeAWS Toolkit for JetBrains, et AWS Toolkit for Visual Studio.

Windows avec Visual Studio et .NET Core

Les éléments suivants sont requis pour le développement sous Windows avec Visual Studio et .NET Core :

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 ou version ultérieure

  Ceci est généralement inclus par défaut lors de l'installation d'une version récente de Visual Studio.

- (Facultatif) Le AWS Toolkit for Visual Studio, qui est un plugin qui fournit une interface utilisateur pour gérer vos AWS ressources et vos profils locaux à partir de Visual Studio. Pour installer le kit d'outils, voir Configuration du AWS Toolkit for Visual Studio.

  Pour plus d’informations, consultez le Guide de l’utilisateur AWS Toolkit for Visual Studio.

Étape suivante

Authentifier le AWS SDK pour .NET avec AWS

Authentifier le AWS SDK pour .NET avec AWS

Vous devez définir la manière dont votre code s'authentifie AWS lorsque vous développez avec Services AWS. Vous pouvez configurer l'accès programmatique aux AWS ressources de différentes manières, en fonction de l'environnement et de l' AWS accès dont vous disposez.

Pour découvrir les différentes méthodes d'authentification pour le SDK, consultez la section Authentification et accès dans le guide de référence des outils AWS SDKs et.

Cette rubrique part du principe qu'un nouvel utilisateur se développe localement, qu'il n'a pas reçu de méthode d'authentification de la part de son employeur et qu'il l'utilisera AWS IAM Identity Center pour obtenir des informations d'identification temporaires. Si votre environnement s'écarte de ces hypothèses, certaines des informations contenues dans cette rubrique ne s'appliquent peut-être pas à votre cas ou vous ont déjà été communiquées.

La configuration de cet environnement nécessite plusieurs étapes, qui sont résumées comme suit :

1. Connectez-vous en utilisant les informations d'identification de console
2. Activation et configuration d'IAM Identity Center
3. Configurez le SDK pour utiliser IAM Identity Center.
4. Démarrer une session sur le portail AWS d'accès

Connectez-vous en utilisant les informations d'identification de console

Vous pouvez utiliser vos identifiants de connexion existants à la console de AWS gestion pour accéder aux AWS services par programmation. Après un flux d'authentification basé sur un navigateur, AWS génère des informations d'identification temporaires qui fonctionnent avec les outils de développement locaux tels que la AWS CLI Outils AWS pour PowerShell et le AWS SDK pour .NET v4.

Grâce à ce processus, vous pouvez vous authentifier à l'aide des informations d'identification root créées lors de la configuration initiale du compte, d'un utilisateur IAM ou d'une identité fédérée auprès de votre fournisseur d'identité, et la AWS CLI gère Outils AWS pour PowerShell automatiquement les informations d'identification temporaires pour vous. Cette approche améliore la sécurité en éliminant le besoin de stocker des informations d'identification à long terme localement. Lorsque vous exécutez l'`Invoke-AWSLogin` applet de `aws login` commande depuis Outils AWS pour PowerShell la AWS CLI, vous pouvez effectuer une sélection parmi vos sessions de console actives ou vous connecter via le flux d'authentification basé sur le navigateur, ce qui générera automatiquement des informations d'identification temporaires. Le jeton d'identification temporaire expirera dans 15 minutes, mais le AWS SDK pour .NET v4 actualise automatiquement le jeton lorsque cela est nécessaire lors de vos demandes.

> **Important**
>
> Si vous utilisez les informations d'identification de la console pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution AWS de connexion puisse fonctionner :
>
> - AWSSDK3. Se connecter
>
> Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

- Outils AWS pour PowerShell Creds de connexion - Guide du développeur.
- Connectez-vous pour le développement AWS local à l'aide des informations d'identification de la console à l'aide de la AWS CLI.

Le `config` fichier ressemblera à ceci une fois que vous vous serez connecté en suivant les étapes indiquées ci-dessus pour Outils AWS pour PowerShell notre AWS CLI.

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

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

Activation et configuration d’IAM Identity Center

Pour utiliser IAM Identity Center, il doit d'abord être activé et configuré. Pour en savoir plus sur la procédure à suivre pour le SDK, consultez l'étape 1 de la rubrique consacrée à l'authentification IAM Identity Center dans le guide de référence AWS SDKs and Tools. Suivez particulièrement toutes les instructions nécessaires dans I do not have established access through IAM Identity Center.

Configurez le SDK pour utiliser IAM Identity Center.

Les informations relatives à la configuration du SDK pour utiliser IAM Identity Center se trouvent à l'étape 2 de la rubrique consacrée à l'authentification IAM Identity Center dans le guide de référence

AWS SDKs and Tools. Une fois cette configuration terminée, votre système doit contenir les éléments suivants :

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- Le AWS `config` fichier partagé qui contient un `[default]`profil avec un ensemble de valeurs de configuration pouvant être référencées à partir du SDK. Pour connaître l'emplacement de ce fichier, reportez-vous à la section Emplacement des fichiers partagés dans le Guide de référence des outils AWS SDKs et. AWS SDK pour .NET Utilise le fournisseur de jetons SSO du profil pour obtenir des informations d'identification avant d'envoyer des demandes à AWS. La valeur `sso_role_name`, qui est un rôle IAM connecté à un ensemble d'autorisations IAM Identity Center, doit autoriser l'accès aux Services AWS utilisés dans votre application.

  L'exemple de fichier `config` suivant illustre un profil par défaut configuré avec le fournisseur de jetons SSO. Le paramètre `sso_session` du profil fait référence à la section `sso-session` nommée. La `sso-session` section contient les paramètres permettant de lancer une session sur le portail AWS d'accès.

```
[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**
>
> Si vous l'utilisez AWS IAM Identity Center pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Démarrer une session sur le portail AWS d'accès

Avant d'exécuter une application qui y accède Services AWS, vous avez besoin d'une session de portail d' AWS accès active pour que le SDK utilise l'authentification IAM Identity Center pour résoudre les informations d'identification. En fonction de la durée de session que vous avez configurée, votre accès finira par expirer et le SDK rencontrera une erreur d'authentification. Pour vous connecter au portail AWS d'accès, exécutez la commande suivante dans le AWS CLI.

```
aws sso login
```

Étant donné que vous disposez d'une configuration de profil par défaut, il n'est pas nécessaire d'appeler la commande avec l'option `--profile`. Si la configuration de votre fournisseur de jetons SSO utilise un profil nommé, la commande est `aws sso login --profile named-profile`.

Pour vérifier si vous avez déjà une session active, exécutez la AWS CLI commande suivante.

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

La réponse à cette commande doit indiquer le compte IAM Identity Center et l'ensemble d'autorisations configurés dans le fichier partagé `config`.

> **Note**
>
> Si vous disposez déjà d'une session active sur le portail AWS d'accès et que vous l'exécutez`aws sso login`, il ne vous sera pas demandé de fournir d'informations d'identification.
> Le processus de connexion peut vous demander d'autoriser l' AWS CLI accès à vos données. Comme AWS CLI il repose sur le SDK pour Python, les messages d'autorisation peuvent contenir des variantes du `botocore` nom.

Informations supplémentaires

- Pour plus d'informations sur l'utilisation d'IAM Identity Center et du SSO dans un environnement de développement, consultez Authentification unique la Authentification avancée section. Ces informations incluent des méthodes alternatives et plus avancées, ainsi que des didacticiels qui vous montrent comment utiliser ces méthodes.
- Pour plus d'options d'authentification pour le SDK, telles que l'utilisation de profils et de variables d'environnement, consultez le chapitre de configuration du guide de référence des outils AWS SDKs et des outils.
- Pour en savoir plus sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.
- Pour créer des AWS informations d'identification à court terme, consultez la section Informations d'identification de sécurité temporaires dans le guide de l'utilisateur IAM.
- Pour en savoir plus sur les autres fournisseurs d'informations d'identification, consultez la section Fournisseurs d'informations d'identification standardisés dans le guide de référence sur les outils AWS SDKs et.

Création d'une application simple à l'aide du AWS SDK pour .NET

Cette section fournit des didacticiels de base pour les développeurs qui découvrent le AWS SDK pour .NET.

> **Note**
>
> Avant d'utiliser ces didacticiels, vous devez d'abord avoir installé votre chaîne d'outils et configuré l'authentification du SDK.

Pour plus d'informations sur le développement de logiciels pour des AWS services spécifiques ainsi que des exemples de code, consultez AWS Services d'appels. Pour des exemples de code supplémentaires, voirSDK pour .NET exemples de code (v4).

Rubriques

- Application multiplateforme simple utilisant AWS SDK pour .NET
- Application Windows simple utilisant le AWS SDK pour .NET

- Étapes suivantes

Application multiplateforme simple utilisant AWS SDK pour .NET

Ce didacticiel utilise .NET Core AWS SDK pour .NET et .NET pour le développement multiplateforme. Le didacticiel explique comment utiliser le SDK pour répertorier les compartiments Amazon S3 que vous possédez et, éventuellement, créer un compartiment.

Vous allez réaliser ce didacticiel à l'aide d'outils multiplateformes comme l'interface de ligne de commande (CLI) .NET. Pour d'autres méthodes de configuration de votre environnement de développement, consultezInstallation et configuration de votre chaîne d'outils pour AWS SDK pour .NET.

Nécessaire pour le développement .NET multiplateforme sous Windows, Linux ou macOS :

- Microsoft .NET Core SDK, version 2.1, 3.1 ou ultérieure, qui inclut l'interface de ligne de commande (CLI) NET (**`dotnet`**) et le moteur d'exécution .NET core.

- Un éditeur de code ou un environnement de développement intégré (IDE) adapté à votre système d'exploitation et à vos exigences. Il s'agit généralement d'une solution qui fournit un certain support pour .NET Core.

  Les exemples incluent Microsoft Visual Studio Code (VS Code), JetBrains Rider et Microsoft Visual Studio.

> **Note**
>
> Avant d'utiliser ces didacticiels, vous devez d'abord avoir installé votre chaîne d'outils et configuré l'authentification du SDK.

Étapes

- Création du projet
- Créer le code
- Exécutez l'application
- Nettoyage

Création du projet

1. Ouvrez l'invite de commande ou le terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

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

3. Accédez au `S3CreateAndList` dossier que vous venez de créer et exécutez les commandes suivantes :

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

Les commandes précédentes installent les NuGet packages depuis le gestionnaire de NuGet packages. Comme nous savons exactement de quels NuGet packages nous avons besoin pour ce didacticiel, nous pouvons effectuer cette étape dès maintenant. Il est également courant que les packages requis soient connus au cours du développement. Dans ce cas, une commande similaire peut être exécutée à ce moment-là.

Créer le code

1. Dans le dossier `S3CreateAndList`, recherchez et ouvrez `Program.cs` dans votre éditeur de code.
2. Remplacez le contenu par le code suivant et enregistrez le fichier.

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

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

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

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

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

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

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

```
                }
                catch (Exception e)
                {
                    Console.WriteLine("Caught exception when creating a bucket:");
                    Console.WriteLine(e.Message);
                }
            }

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

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

```
                    "\n\nUsage: S3CreateAndList [bucket_name]" +
                    "\n - bucket_name: A valid, globally unique bucket name." +
                    "\n - If bucket_name isn't supplied, this utility simply lists
 your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

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

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

Exécutez l'application

1. Exécutez la commande suivante.

```
dotnet run
```

2. Examinez le résultat pour connaître le nombre de compartiments Amazon S3 que vous possédez, le cas échéant, ainsi que leurs noms.

3. Choisissez un nom pour un nouveau compartiment Amazon S3. Utilisez « dotnet-quicktour-s 3-1-cross- » comme base et ajoutez-y quelque chose d'unique, comme un GUID ou votre nom. Assurez-vous de suivre les règles relatives aux noms de compartiments, telles que décrites dans la section Règles relatives à la dénomination des compartiments dans le guide de l'utilisateur Amazon S3.
4. Exécutez la commande suivante en la *amzn-s3-demo-bucket* remplaçant par le nom du bucket que vous avez choisi.

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

5. Examinez le résultat pour voir le nouveau compartiment créé.

Nettoyage

Au cours de ce didacticiel, vous avez créé des ressources que vous pouvez choisir de nettoyer pour le moment.

- Si vous ne souhaitez pas conserver le compartiment créé par l'application lors d'une étape précédente, supprimez-le à l'aide de la console Amazon S3 à l'adresse https://console.aws.amazon.com/s3/.
- Si vous ne souhaitez pas conserver votre projet .NET, supprimez le dossier `S3CreateAndList` de votre environnement de développement.

Étapes suivantes

Retournez au menu de visite rapide ou allez directement à la fin de cette visite rapide.

Application Windows simple utilisant le AWS SDK pour .NET

Ce didacticiel utilise AWS SDK pour .NET Windows avec Visual Studio et .NET Core. Le didacticiel explique comment utiliser le SDK pour répertorier les compartiments Amazon S3 que vous possédez et éventuellement créer un compartiment.

Vous exécuterez ce didacticiel sous Windows avec Visual Studio et .NET Core. Pour d'autres méthodes de configuration de votre environnement de développement, consultezInstallation et configuration de votre chaîne d'outils pour AWS SDK pour .NET.

Nécessaire pour le développement sous Windows avec Visual Studio et .NET Core :

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 ou version ultérieure

  Ceci est généralement inclus par défaut lors de l'installation d'une version récente de Visual Studio.

> **Note**
>
> Avant d'utiliser ces didacticiels, vous devez d'abord avoir installé votre chaîne d'outils et configuré l'authentification du SDK.

Étapes

- Création du projet
- Créer le code
- Exécutez l'application
- Nettoyage

Création du projet

1. Ouvrez Visual Studio et créez un nouveau projet qui utilise la version C# du modèle d'application console, c'est-à-dire avec la description suivante : «... pour créer une application de ligne de commande pouvant s'exécuter sur .NET... ». Appelez le projet `S3CreateAndList`.

   > **Note**
   >
   > Ne choisissez pas la version .NET Framework du modèle d'application de console ou, si c'est le cas, veillez à utiliser .NET Framework 4.7.2 ou version ultérieure.

2. Une fois le projet nouvellement créé chargé, choisissez Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Recherchez les NuGet packages suivants et installez-les dans le projet : `AWSSDK.S3AWSSDK.SecurityToken`,`AWSSDK.SSO`, et `AWSSDK.SSOOIDC`

Ce processus installe les NuGet packages à partir du gestionnaire de NuGet packages. Comme nous savons exactement de quels NuGet packages nous avons besoin pour ce didacticiel, nous pouvons effectuer cette étape dès maintenant. Il est également courant que les packages requis soient connus au cours du développement. Dans ce cas, appliquez un processus similaire pour les installer à ce moment-là.

4. Si vous avez l'intention d'exécuter l'application à partir de l'invite de commande, ouvrez une invite de commande maintenant et accédez au dossier qui contiendra le résultat de la compilation. C'est généralement quelque chose comme ça`S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, mais cela dépend de votre environnement.

Créer le code

1. Dans le projet `S3CreateAndList`, recherchez et ouvrez `Program.cs` dans l'IDE.
2. Remplacez le contenu par le code suivant et enregistrez le fichier.

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

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

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-
start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
 privileges for
  //   STS and S3 buckets.
        // - An active SSO Token.
```

```
        //    If an active SSO token isn't available, the SSO user should do the
following:
        //    In a terminal, the SSO user must call "aws sso login".

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

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

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

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

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

```
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

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

        //
        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
```

```
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

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

3. Générer l'application

> **Note**
>
> Si vous utilisez une ancienne version de Visual Studio, il est possible que vous obteniez une erreur de compilation similaire à la suivante :
> « La fonctionnalité « async main » n'est pas disponible dans C# 7.0. Veuillez utiliser la version linguistique 7.1 ou supérieure. »
> Si cette erreur s'affiche, configurez votre projet pour utiliser une version ultérieure de la langue. Cela se fait généralement dans les propriétés du projet, Build, Advanced.

Exécutez l'application

1. Exécutez l'application sans arguments de ligne de commande. Faites-le soit dans l'invite de commande (si vous en avez ouvert une plus tôt), soit depuis l'IDE.
2. Examinez le résultat pour connaître le nombre de compartiments Amazon S3 que vous possédez, le cas échéant, ainsi que leurs noms.

3. Choisissez un nom pour un nouveau compartiment Amazon S3. Utilisez « dotnet-quicktour-s 3-1-winvs- » comme base et ajoutez-y quelque chose d'unique, comme un GUID ou votre nom. Assurez-vous de suivre les règles relatives aux noms de compartiments, telles que décrites dans la section Règles de dénomination des compartiments du guide de l'utilisateur Amazon S3.
4. Exécutez à nouveau l'application, en fournissant cette fois le nom du compartiment.

   Dans la ligne de commande, remplacez *amzn-s3-demo-bucket* la commande suivante par le nom du bucket que vous avez choisi.

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

   Ou, si vous exécutez l'application dans l'IDE, choisissez Project, S3 CreateAndList Properties, Debug et entrez le nom du bucket à cet endroit.
5. Examinez le résultat pour voir le nouveau compartiment créé.

Nettoyage

Au cours de ce didacticiel, vous avez créé des ressources que vous pouvez choisir de nettoyer pour le moment.

- Si vous ne souhaitez pas conserver le compartiment créé par l'application lors d'une étape précédente, supprimez-le à l'aide de la console Amazon S3 à l'adresse https://console.aws.amazon.com/s3/.
- Si vous ne souhaitez pas conserver votre projet .NET, supprimez le dossier `S3CreateAndList` de votre environnement de développement.

Étapes suivantes

Retournez au menu de visite rapide ou allez directement à la fin de cette visite rapide.

Étapes suivantes

Veillez à nettoyer les ressources restantes que vous avez créées lors de l'exécution de ces didacticiels. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Maintenant que vous les avez visitées AWS SDK pour .NET, vous voudrez peut-être commencer votre projet.

Configuration du AWS SDK pour .NET

Pour utiliser le AWS SDK pour .NET, vous devez configurer votre environnement et également configurer un certain nombre d'éléments essentiels dont votre application a besoin pour accéder aux AWS services. Il s'agit des licences suivantes :

- Spécification de la AWS région
- AWSSDK packages ou assemblages

Certaines des rubriques de cette section fournissent des informations sur la façon de configurer ces éléments essentiels.

Les autres rubriques de cette section et d'autres sections fournissent des informations sur des méthodes plus avancées de configuration de votre projet.

Rubriques

- Lancement d'un nouveau projet
- Installez AWSSDK des packages avec NuGet
- Installez AWSSDK des assemblages sans NuGet
- Configuration de AWS la région pour AWS SDK pour .NET
- Résolution des informations d'identification et des profils
- Rétentatives et délais
- Observabilité
- Informations supplémentaires sur les utilisateurs et les rôles
- Configuration avancée pour votre AWS SDK pour .NET projet
- Utilisation d'informations d'identification existantes

Lancement d'un nouveau projet

Il existe plusieurs techniques que vous pouvez utiliser pour démarrer un nouveau projet afin d'accéder aux AWS services. Voici certaines de ces techniques :

- Si vous débutez dans le développement .NET, AWS ou du moins si vous êtes novice dans le AWS SDK pour .NET domaine, vous pouvez consulter des exemples complets dansCréation d'une application simple. Il vous présente le SDK.

- Vous pouvez démarrer un projet de base à l'aide de l'interface de ligne de commande .NET. Pour en voir un exemple, ouvrez une invite de commande ou un terminal, créez un dossier ou un répertoire, naviguez jusqu'à celui-ci, puis entrez ce qui suit.

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

  Un projet vide est créé dans lequel vous pouvez ajouter du code et des NuGet packages. Pour plus d'informations, consultez le guide .NET Core.

  Pour consulter la liste des modèles de projets, utilisez ce qui suit : `dotnet new --list`

- AWS Toolkit for Visual Studio Il inclut des modèles de projet C# pour une variété de AWS services. Après avoir installé le kit d'outils dans Visual Studio, vous pouvez accéder aux modèles lors de la création d'un nouveau projet.

  Pour le voir, consultez la section Utilisation des AWS services dans le guide de AWS Toolkit for Visual Studio l'utilisateur. Plusieurs des exemples présentés dans cette section créent de nouveaux projets.

- Si vous développez avec Visual Studio sous Windows mais sans le AWS Toolkit for Visual Studio, utilisez les techniques habituelles pour créer un nouveau projet.

  Pour voir un exemple, ouvrez Visual Studio et choisissez Fichier, Nouveau, Projet. Recherchez « .net core » et choisissez la version C# du modèle d'application console (.NET Core) ou d'application WPF (.NET Core). Un projet vide est créé dans lequel vous pouvez ajouter du code et des NuGet packages.

Vous trouverez quelques exemples de la manière d'utiliser les AWS services dansExemples de code guidés.

> **Important**
>
> Si vous l'utilisez AWS IAM Identity Center pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Installez AWSSDK des packages avec NuGet

NuGetest un système de gestion de paquets pour la plate-forme .NET. Avec NuGet, vous pouvez installer les AWSSDKpackages, ainsi que plusieurs autres extensions, sur votre projet. Pour plus d'informations, consultez le référentiel aws/dotnet sur le site Web. GitHub

NuGet possède toujours les versions les plus récentes des AWSSDK packages, ainsi que les versions précédentes. NuGetconnaît les dépendances entre les packages et installe automatiquement tous les packages requis.

> **Warning**
>
> La liste des NuGet packages peut inclure un package nommé simplement AWSSDK « » (sans identifiant ajouté). N'installez PAS ce NuGet package ; il s'agit d'un ancien package qui ne doit pas être utilisé pour de nouveaux projets.

Les packages installés avec NuGet sont stockés avec votre projet plutôt que dans un emplacement central. Cela vous permet d'installer des versions d'assemblage spécifiques à une application donnée sans créer de problèmes de compatibilité pour les autres applications. Pour plus d'informations NuGet, consultez la NuGet documentation.

> **Note**
>
> Si vous ne pouvez pas ou n'êtes pas autorisé à télécharger et à installer des NuGet packages par projet, vous pouvez obtenir les AWSSDK assemblages et les stocker localement (ou sur site).

Si cela s'applique à vous et que vous n'avez pas encore obtenu les AWSSDK assemblages, consultezObtention d' AWSSDK assemblages. Pour savoir comment utiliser les assemblages stockés localement, consultezInstallez AWSSDK des assemblages sans NuGet.

Utilisation NuGet à partir de l'invite de commande ou du terminal

1. Accédez aux AWSSDK packages concernés NuGet et déterminez les packages dont vous avez besoin pour votre projet, par exemple AWSSDK.S3.
2. Copiez la commande .NET CLI depuis la page Web de ce package, comme indiqué dans l'exemple suivant.

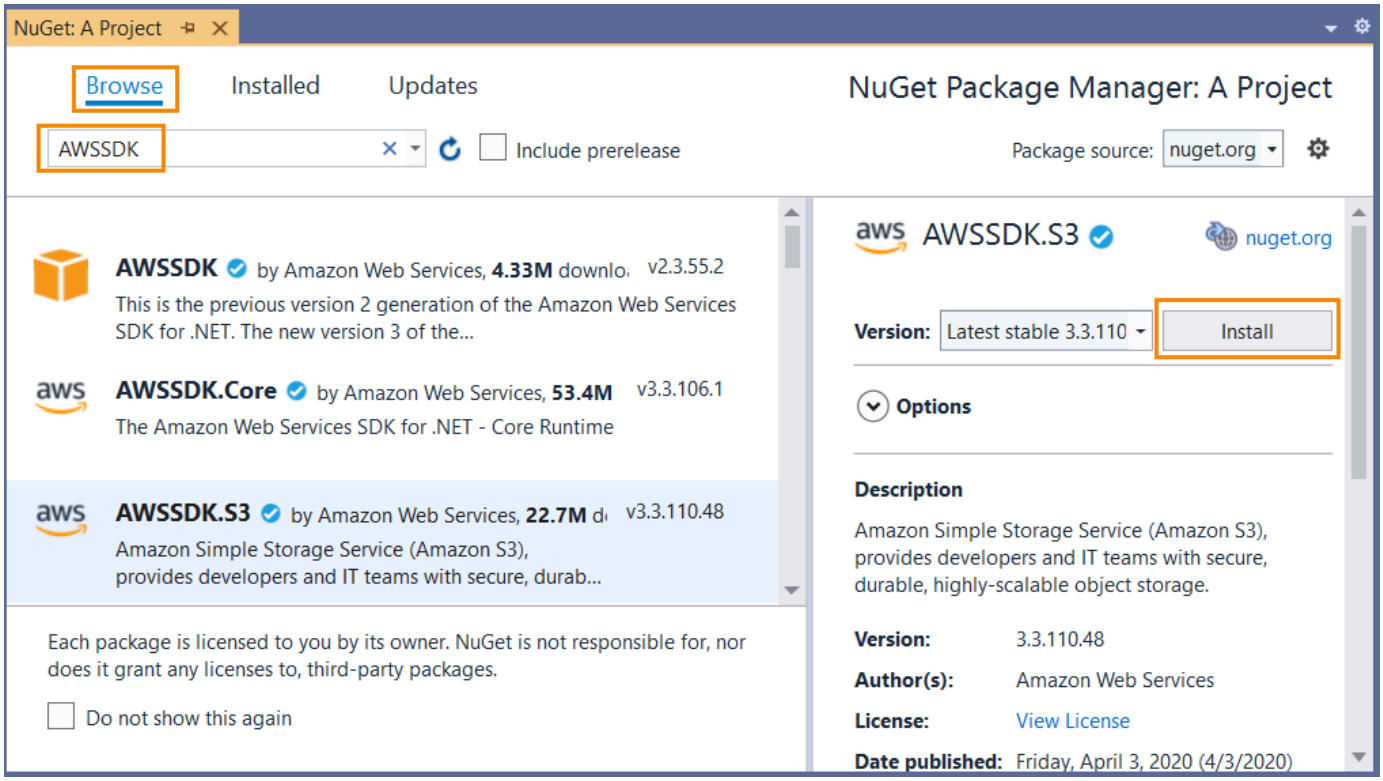
   ```
   dotnet add package AWSSDK.S3 --version 3.3.110.19
   ```
3. Dans le répertoire de votre projet, exécutez cette commande .NET CLI. NuGet installe également toutes les dépendances, telles que AWSSDK.Core.

> **Note**
>
> Si vous souhaitez uniquement disposer de la dernière version d'un NuGet package, vous pouvez exclure les informations de version de la commande, comme indiqué dans l'exemple suivant.
>
> ```
> dotnet add package AWSSDK.S3
> ```

Utilisation NuGet depuis l'explorateur de solutions Visual Studio

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur votre projet, puis sélectionnez Gérer les NuGet packages dans le menu contextuel.
2. Dans le volet gauche du Gestionnaire de NuGet packages, choisissez Browse. Vous pouvez ensuite utiliser le champ de recherche pour rechercher le package que vous souhaitez installer. NuGet installe également toutes les dépendances, telles que AWSSDK.Core.

   La figure suivante montre l'installation du package AWSSDK.S3.



Utilisation NuGet depuis la console Package Manager

Dans Visual Studio, choisissez Outils, Gestionnaire de NuGet packages, Console du gestionnaire de packages.

Vous pouvez installer les AWSSDK packages souhaités à partir de la console Package Manager à l'aide de la **`Install-Package`**commande. Par exemple, pour installer AWSSDK.S3, utilisez la commande suivante.

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

NuGet installe également toutes les dépendances, telles que AWSSDK.Core.

Si vous devez installer une version antérieure d'un package, utilisez l'`-Version`option et spécifiez la version de package souhaitée, comme indiqué dans l'exemple suivant.

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

Pour plus d'informations sur les commandes de la console Package Manager, consultez la PowerShellréférence dans la NuGetdocumentation de Microsoft.

Installez AWSSDK des assemblages sans NuGet

Cette rubrique décrit comment utiliser les AWSSDK assemblages que vous avez obtenus et stockés localement (ou sur site) comme décrit dansObtention d' AWSSDK assemblages. Cette méthode n'est pas recommandée pour gérer les références au SDK, mais elle est requise dans certains environnements.

> **Note**
>
> La méthode recommandée pour gérer les références aux SDK consiste à télécharger et à installer uniquement les NuGet packages dont chaque projet a besoin. Cette méthode est décrite dansInstallez AWSSDK des packages avec NuGet.

Pour installer des AWSSDK assemblages

1. Créez un dossier dans votre zone de projet pour les AWSSDK assemblages requis. Par exemple, vous pouvez appeler ce dossier`AwsAssemblies`.
2. Si ce n'est pas déjà fait, procurez-vous les AWSSDK assemblages, qui les placeront dans un dossier de téléchargement ou d'installation local. Copiez les fichiers DLL pour les assemblages requis depuis ce dossier de téléchargement dans votre projet (dans le `AwsAssemblies` dossier, dans notre exemple).

   Veillez également à copier toutes les dépendances. Vous trouverez des informations sur les dépendances sur le GitHubsite Web.
3. Faites référence aux assemblages requis comme suit.

   Cross-platform development

   1. Ouvrez le `.csproj` fichier de votre projet et ajoutez un `<ItemGroup>` élément.
   2. Dans l'`<ItemGroup>`élément, ajoutez un `<Reference>` élément avec un `Include` attribut pour chaque assemblage requis.

      Pour Amazon S3, par exemple, vous devez ajouter les lignes suivantes au `.csproj` fichier de votre projet.

      Sous Linux et macOS :

      ```
      <ItemGroup>
      ```

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

Sous Windows :

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

3. Enregistrez le `.csproj` fichier de votre projet.

Windows with Visual Studio and .NET Core

1. Dans Visual Studio, chargez votre projet et ouvrez Projet, Ajouter une référence.
2. Cliquez sur le bouton Parcourir en bas de la boîte de dialogue. Accédez au dossier de votre projet et au sous-dossier dans lequel vous avez copié les fichiers DLL requis (`AwsAssembliespar` exemple).
3. Sélectionnez tous les fichiers DLL, choisissez Ajouter, puis OK.
4. Enregistrez votre projet.

Configuration de AWS la région pour AWS SDK pour .NET

AWS Les régions vous permettent d'accéder à AWS des services qui résident physiquement dans une région géographique spécifique. Cela peut être utile pour la redondance et pour conserver vos données et vos applications en cours d'exécution près de l'endroit où vous et vos utilisateurs y accèderez.

Pour consulter la liste actuelle de toutes les régions et points de terminaison pris en charge pour chaque AWS service, consultez la section Points de terminaison et quotas du service dans le. Références générales AWS Pour consulter la liste des points de terminaison régionaux existants, consultez la section Points de terminaison AWS de service. Pour obtenir des informations détaillées sur les régions, voir Spécifier les AWS régions que votre compte peut utiliser.

Vous pouvez créer un client AWS de service qui se rend dans une région donnée. Vous pouvez également configurer votre application avec une région qui sera utilisée pour tous les clients AWS du service. Ces deux cas sont expliqués ci-après.

Création d'un client de service avec une région particulière

Vous pouvez spécifier la région pour tous les clients du AWS service dans votre application. La définition de la région de cette manière a priorité sur tout paramètre global pour ce client de service en particulier.

Région existante

Cet exemple vous montre comment instancier un EC2 client Amazon dans une région existante. Il utilise des RegionEndpointchamps définis.

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

Nouvelle région utilisant la RegionEndpoint classe

Cet exemple vous montre comment créer un nouveau point de terminaison de région en utilisant 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
}
```

Nouvelle région utilisant la classe de configuration du client de service

Cet exemple montre comment utiliser la `ServiceURL` propriété de la classe de configuration du client de service pour spécifier la région ; dans ce cas, en utilisant la classe Amazon EC2 Config.

Cette technique fonctionne même si le point de terminaison de région ne suit pas le modèle de point de terminaison de région normal.

```
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
}
```

Spécifiez une région pour tous les clients du service

Il existe plusieurs manières de spécifier une région pour tous les clients de AWS service créés par votre application. Cette région est utilisée pour les clients de service qui ne sont pas créés avec une région en particulier.

Recherche AWS SDK pour .NET une valeur de région dans l'ordre suivant.

Profils

Définissez un profil que votre application ou le SDK a chargé. Pour de plus amples informations, veuillez consulter Résolution des informations d'identification et des profils.

Variables d'environnement

Défini dans la variable d'`AWS_REGION`environnement.

Sous Linux ou macOS :

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

Sous Windows :

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

> **Note**
>
> Si vous définissez cette variable d'environnement pour l'ensemble du système (à l'aide de `export` ou`setx`), elle affecte l'ensemble des boîtes à outils, SDKs et pas seulement le AWS SDK pour .NET.

AWSConfigs classe

Défini en tant que AWSConfigs. AWSRegionpropriété.

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

Résolution de la région

Si aucune des méthodes décrites ci-dessus n'est utilisée pour spécifier un Région AWS, les AWS SDK pour .NET tentatives de recherche d'une région dans laquelle le client du AWS service pourra opérer.

Ordre de résolution régional

1. Fichiers de configuration de l'application tels que `app.config` et`web.config`.
2. Variables d'environnement (`AWS_REGION`Net`AWS_DEFAULT_REGION`).
3. Un profil dont le nom est spécifié par une valeur dans`AWSConfigs.AWSProfileName`.
4. Un profil dont le nom est spécifié par la variable d'`AWS_PROFILE`environnement.
5. Le `[default]` profil.
6. Métadonnées de l' EC2 instance Amazon (si elle est exécutée sur une EC2 instance).

Si aucune région n'est trouvée, le SDK émet une exception indiquant qu'aucune région n'est configurée pour le client de AWS service.

Informations spéciales sur la région de Chine (Pékin)

Pour utiliser les services dans la région Chine (Beijing), vous devez posséder un compte et les informations d'identification qui sont spécifiques à la région Chine (Beijing). Les comptes et informations d'identification AWS des autres régions ne fonctionneront pas pour la région de Chine (Pékin). De même, les comptes et informations d'identification de la région de Chine (Pékin) ne fonctionneront pas pour AWS les autres régions. Pour plus d'informations sur les terminaux et les protocoles disponibles dans la région de Chine (Pékin), consultez la section Points de terminaison de la région de Pékin.

Informations spéciales sur les nouveaux AWS services

De nouveaux AWS services peuvent être lancés dans un premier temps dans quelques régions, puis pris en charge dans d'autres régions. Dans ces cas, il n'est pas nécessaire d'installer le dernier SDK

pour accéder aux nouvelles régions pour ce service. Vous pouvez spécifier les régions récemment ajoutées par client ou globalement, comme indiqué précédemment.

Résolution des informations d'identification et des profils

Il AWS SDK pour .NET recherche les informations d'identification dans un certain ordre et utilise le premier ensemble disponible pour l'application en cours.

Ordre de recherche des informations d'identification

1. Informations d'identification définies explicitement sur le client AWS de service, comme décrit dansAccès aux informations d'identification et aux profils dans une application.

   > **Note**
   >
   > Cette rubrique figure dans la Considérations spéciales section car il ne s'agit pas de la méthode préférée pour spécifier les informations d'identification.

2. Session AWSCredentials créée à partir des variables d'`AWS_SESSION_TOKEN`environnement `AWS_ACCESS_KEY_ID`AWS_SECRET_ACCESS_KEY, et, si les trois variables ont une valeur.
3. AWSCredentialsBasiques créés à partir des variables d'`AWS_SECRET_ACCESS_KEY`environnement `AWS_ACCESS_KEY_ID` et, si les deux variables ont une valeur.
4. AssumeRoleWithWebIdentityCredentialsqui sont créés à partir des variables d'`AWS_ROLE_ARN`environnement `AWS_WEB_IDENTITY_TOKEN_FILE` et, si les deux variables ont une valeur.
5. Un profil d'informations d'identification dont le nom est spécifié par une valeur dans AWSConfigs. AWSProfileNom.
6. Un profil d'informations d'identification dont le nom est spécifié par la variable d'`AWS_PROFILE`environnement.
7. Profil d'informations d'identification `[default]`.
8. Le fournisseur d'informations d'identification du conteneur.
9. Métadonnées de l' EC2 instance Amazon.

Si votre application s'exécute sur une EC2 instance Amazon, par exemple dans un environnement de production, utilisez un rôle IAM comme décrit dansOctroi d'accès à l'aide d'un rôle IAM. Dans le cas

contraire, comme lors des tests préliminaires, stockez vos informations d'identification dans un fichier utilisant le format de fichier AWS d'informations d'identification auquel votre application Web a accès sur le serveur.

Pour plus d'informations sur les variables d'environnement et les profils d'identification, consultez les rubriques suivantes du Guide de référence sur les outils AWS SDKs et les variables d'environnement : variables d'environnement, liste des variables d'environnement et fichiers de configuration et d'informations d'identification partagés.

Résolution du profil

Avec deux mécanismes de stockage différents pour les informations d'identification, il est important de comprendre comment les configurer AWS SDK pour .NET pour les utiliser. Le AWSConfigs. AWSProfilesLa propriété de localisation contrôle la manière dont AWS SDK pour .NET il trouve les profils d'identification.

AWSProfilesLocation	Comportement de résolution des profils
null (non défini) ou vide	Recherchez dans le SDK Store si la plateform e le prend en charge, puis recherchez le fichier AWS d'informations d'identification partagé à l'emplacement par défaut. Si le profil ne se trouve dans aucun de ces emplaceme nts, recherchez `~/.aws/config` (Linux ou macOS) ou `%USERPROFILE%\.aws \config` (Windows).
Le chemin d'accès à un fichier au format de fichier AWS d'informations d'identification	Recherche uniquement dans le fichier spécifié un profil portant le nom spécifié.

Utilisation des informations d'identification d'un compte utilisateur fédéré

Les applications qui utilisent le AWS SDK pour .NET (AWSSDK.Core version 3.1.6.0 et versions ultérieures) peuvent utiliser des comptes utilisateurs fédérés via Active Directory Federation Services (AD FS) pour accéder aux AWS services à l'aide du langage SAML (Security Assertion Markup Language).

La prise en charge de l'accès fédéré signifie que les utilisateurs peuvent s'authentifier en utilisant votre instance Active Directory. Des informations d'identification temporaires sont automatiquement accordées à l'utilisateur. Ces informations d'identification temporaires, valides pendant une heure, sont utilisées lorsque votre application invoque AWS des services. Le kit SDK se charge de gérer les informations d'identification temporaires. Pour les comptes utilisateur joints à un domaine, si votre application effectue un appel mais que les informations d'identification ont expiré, l'utilisateur est de nouveau authentifié automatiquement et de nouvelles informations d'identification lui sont octroyées. (Pour les comptes qui ne sont pas associés à un domaine, l'utilisateur est invité à saisir ses informations d'identification avant de se réauthentifier.)

Pour utiliser cette prise en charge dans votre application .NET, vous devez d'abord configurer le profil de rôle à l'aide d'une PowerShell applet de commande. Pour savoir comment procéder, consultez la AWS Tools for Windows PowerShell documentation.

Après avoir configuré le profil de rôle, référencez-le dans votre application. Il existe plusieurs façons de le faire, dont l'une consiste à utiliser le AWSConfigs. AWSProfileNommez la propriété de la même manière que vous le feriez avec d'autres profils d'identification.

L'AWS Security Token Serviceassemblage (AWSSDK. SecurityToken) fournit le support SAML pour obtenir des AWS informations d'identification. Pour utiliser les informations d'identification d'un compte utilisateur fédéré, assurez-vous que cet assemblage est disponible pour votre application.

Spécifier des rôles ou des informations d'identification temporaires

Pour les applications qui s'exécutent sur EC2 des instances Amazon, le moyen le plus sûr de gérer les informations d'identification consiste à utiliser les rôles IAM, comme décrit dansOctroi d'accès à l'aide d'un rôle IAM.

Pour les scénarios d'application dans lesquels le logiciel exécutable est accessible à des utilisateurs extérieurs à votre organisation, nous vous recommandons de concevoir le logiciel de manière à utiliser des informations d'identification de sécurité temporaires. En plus de fournir un accès restreint aux AWS ressources, ces informations d'identification ont l'avantage d'expirer après une période spécifiée. Pour plus d'informations sur les informations d'identification de sécurité temporaires, consultez les rubriques suivantes :

- Informations d'identification de sécurité temporaires
- Groupes d'identités Amazon Cognito

Utilisation des informations d'identification du proxy

Si votre logiciel communique avec AWS un proxy, vous pouvez spécifier les informations d'identification du proxy en utilisant la `ProxyCredentials` propriété de la `Config` classe d'un service. La `Config` classe d'un service fait généralement partie de l'espace de noms principal du service. Les exemples incluent les suivants : AmazonCloudDirectoryConfigsur Amazon. CloudDirectoryespace de noms et AmazonGameLiftConfigdans Amazon. GameLiftespace de noms.

Pour Amazon S3, par exemple, vous pouvez utiliser un code similaire au suivant, où `SecurelyStoredUserName` et `SecurelyStoredPassword` sont le nom d'utilisateur et le mot de passe du proxy spécifiés dans un NetworkCredentialobjet.

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

> **Note**
>
> Les versions antérieures du kit SDK utilisaient `ProxyUsername` et `ProxyPassword`, mais ces propriétés sont obsolètes.

Rétentatives et délais

Vous AWS SDK pour .NET permet de configurer le nombre de tentatives et les valeurs de délai d'expiration pour les requêtes HTTP adressées aux AWS services. Si les valeurs par défaut concernant les nouvelles tentatives et les délais d'expiration ne conviennent pas à votre application, vous pouvez les ajuster selon vos besoins spécifiques, mais il est important de comprendre en quoi le comportement de votre application en sera affecté.

Pour déterminer quelles valeurs utiliser pour les nouvelles tentatives et les délais d'expiration, prenez en compte les éléments suivants :

- Comment l'application AWS SDK pour .NET et votre application doivent-elles réagir lorsque la connectivité réseau se dégrade ou qu'un AWS service est inaccessible ? Souhaitez-vous que l'appel échoue rapidement ou est-il indiqué que l'appel continue de faire l'objet de nouvelles tentatives en votre nom ?

- Votre application est-elle une application ou un site web orienté vers l'utilisateur qui doit être réactive ou s'agit-il d'une tâche de traitement en arrière plan qui tolère davantage les latences accrues ?
- L'application est-elle déployée sur un réseau fiable à faible latence ou est-elle déployée sur un site distant avec une connectivité peu fiable ?

Nouvelles tentatives

Présentation de

Ils AWS SDK pour .NET peuvent réessayer les demandes qui échouent en raison d'un ralentissement côté serveur ou d'une interruption de connexion. Il existe deux propriétés des classes de configuration de service que vous pouvez utiliser pour spécifier le comportement d'un client de service en cas de nouvelle tentative. Les classes de configuration de service héritent de ces propriétés de l'abstrait Amazon.Runtime. ClientConfigclasse de l'AWS SDK pour .NET API Reference :

- `RetryMode`spécifie l'un des trois modes de nouvelle tentative définis dans Amazon.Runtime. RequestRetryModeénumération.

  La valeur par défaut de votre application peut être contrôlée à l'aide de la variable d'`AWS_RETRY_MODE`environnement ou du paramètre retry_mode dans le fichier de configuration partagé AWS .

- `MaxErrorRetry`indique le nombre de tentatives autorisées au niveau du client de service ; le SDK réessaie l'opération le nombre de fois spécifié avant d'échouer et de lancer une exception.

  La valeur par défaut de votre application peut être contrôlée à l'aide de la variable d'`AWS_MAX_ATTEMPTS`environnement ou du paramètre max_attempts du fichier de AWS configuration partagé.

Les descriptions détaillées de ces propriétés se trouvent dans le résumé Amazon.Runtime. ClientConfigclasse de l'AWS SDK pour .NET API Reference. Chaque valeur de `RetryMode` correspond par défaut à une valeur particulière de`MaxErrorRetry`, comme indiqué dans le tableau suivant.

RetryMode	MaxErrorRetry Correspondant (Amazon DynamoDB)	MaxErrorRetry Correspondant (tous les autres)
Héritée	10	4
Standard	10	2
Adaptatif (expérimental)	10	2

Comportement

Quand votre candidature démarre

Lorsque votre application démarre, les valeurs par défaut pour `RetryMode` et `MaxErrorRetry` sont configurées par le SDK. Ces valeurs par défaut sont utilisées lorsque vous créez un client de service, sauf si vous spécifiez d'autres valeurs.

- Si les propriétés ne sont pas définies dans votre environnement, la valeur par défaut `RetryMode` est configurée comme Legacy et la valeur par défaut pour `MaxErrorRetry` est configurée avec la valeur correspondante du tableau précédent.
- Si le mode de nouvelle tentative a été défini dans votre environnement, cette valeur est utilisée par défaut pour`RetryMode`. La valeur par défaut pour `MaxErrorRetry` est configurée avec la valeur correspondante du tableau précédent, sauf si la valeur pour les erreurs maximales a également été définie dans votre environnement (décrit ci-dessous).
- Si la valeur du maximum d'erreurs a été définie dans votre environnement, cette valeur est utilisée par défaut pour`MaxErrorRetry`. Amazon DynamoDB fait exception à cette règle ; la valeur DynamoDB par défaut `MaxErrorRetry` pour est toujours celle du tableau précédent.

Pendant l'exécution de votre application

Lorsque vous créez un client de service, vous pouvez utiliser les valeurs par défaut pour `RetryMode` et`MaxErrorRetry`, comme décrit précédemment, ou vous pouvez spécifier d'autres valeurs. Pour spécifier d'autres valeurs, créez et incluez un objet de configuration de service tel qu'AmazonDynamoDBConfigAmazon SQSConfig lorsque vous créez le client de service.

Ces valeurs ne peuvent pas être modifiées pour un client de service une fois celui-ci créé.

Considérations

Lorsqu'une nouvelle tentative a lieu, la latence de votre demande augmente. Vous devez configurer les nouvelles tentatives en fonction des limites de votre application en termes de latence totale des demandes et de taux d'erreur.

Délais

Vous AWS SDK pour .NET permet de configurer les délais d'expiration des demandes au niveau du client de service et par appel de méthode. Il existe deux mécanismes pour configurer les délais d'expiration, qui sont abordés dans les sections suivantes :

- Si vous utilisez des appels asynchrones, vous pouvez utiliser le `CancellationToken` paramètre de la méthode.
- Si vous utilisez des appels synchrones dans .NET Framework, vous pouvez utiliser les `ReadWriteTimeout` propriétés `Timeout` et du fichier abstrait Amazon.Runtime. ClientConfigclasse.

Utilisation du **`CancellationToken`** paramètre pour les délais

Vous AWS SDK pour .NET permet de configurer les délais d'expiration des demandes pour les appels asynchrones à l'aide du paramètre. `CancellationToken` L'extrait de code suivant montre un exemple. Le code est lancé `System.Threading.Tasks.TaskCanceledException` si la demande n'est pas terminée dans les 10 secondes.

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

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

Utilisation des **`ReadWriteTimeout`** propriétés **`Timeout`** et pour les délais

> **Note**
>
> La `Timeout` propriété n'a aucun impact sur les appels asynchrones. Si vous utilisez des appels asynchrones, consultez Utilisation du `CancellationToken` paramètre pour les délais plutôt.

Vous AWS SDK pour .NET permet de configurer les valeurs du délai d'expiration de la demande et du read/write délai d'expiration du socket au niveau du client de service. Ces valeurs sont spécifiées dans `Timeout` et dans les `ReadWriteTimeout` propriétés du résumé Amazon.Runtime. ClientConfigclasse. Ces valeurs sont transmises sous forme de `ReadWriteTimeout` propriétés `Timeout` et des HttpWebRequestobjets créés par l'objet client du AWS service. La valeur par défaut de `Timeout` est de 100 secondes et celle de `ReadWriteTimeout` est de 300 secondes.

Lorsque votre réseau présente une latence élevée ou qu'une opération fait l'objet d'une nouvelle tentative du fait de certaines conditions, un long délai d'expiration et un nombre élevé de nouvelles tentatives peuvent donner l'impression que certaines opérations du kit SDK ne répondent pas.

> **Note**
>
> La version AWS SDK pour .NET qui cible la bibliothèque de classes portable (PCL) utilise la HttpClientclasse au lieu de la `HttpWebRequest` classe et ne prend en charge que la propriété Timeout.

Vous trouverez ci-dessous les exceptions aux valeurs de délai d'expiration par défaut. Ces valeurs sont remplacées lorsque vous définissez explicitement les valeurs de délai d'expiration.

- `Timeout`et `ReadWriteTimeout` sont définis sur les valeurs maximales si la méthode appelée télécharge un flux, tel que AmazonS3Client. PutObjectAsync(), client Amazon S3. UploadPartAsync(), AmazonGlacierClient. UploadArchiveAsync(), et ainsi de suite.
- Les versions du .NET Framework cible AWS SDK pour .NET ont défini `Timeout` et `ReadWriteTimeout` atteint les valeurs maximales pour tous les clients et objets AmazonS3Client. AmazonGlacierClient

- Les versions AWS SDK pour .NET qui ciblent la bibliothèque de classes portable (PCL) et .NET Core sont définies `Timeout` sur la valeur maximale pour tous les clients et objets AmazonS3Client. AmazonGlacierClient

L'exemple suivant vous montre comment définir le mode de nouvelle tentative standard, un maximum de 3 tentatives, un délai d'expiration de 10 secondes et un read/write délai d'expiration de 10 secondes (le cas échéant). Le constructeur AmazonS3Client reçoit un objet AmazonS3Config.

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

Observabilité

L'observabilité est la mesure dans laquelle l'état actuel d'un système peut être déduit des données qu'il émet. Les données émises sont communément appelées télémétrie.

Ils AWS SDK pour .NET peuvent fournir deux signaux de télémétrie courants, les métriques et les traces, ainsi que la journalisation. Vous pouvez connecter un TelemetryProviderpour envoyer des données de télémétrie à un backend d'observabilité (tel que AWS X-RayAmazon CloudWatch), puis agir en conséquence.

Par défaut, les signaux de télémétrie sont désactivés dans le SDK. Cette rubrique explique comment activer et configurer la sortie de télémétrie.

Ressources supplémentaires

Pour plus d'informations sur l'activation et l'utilisation de l'observabilité, consultez les ressources suivantes :

- OpenTelemetry
- Le billet de blog Enhancing Observability in the AWS SDK pour .NET with OpenTelemetry.

- Le billet de blog annonçant la disponibilité générale des OpenTelemetry bibliothèques AWS .NET.
- Exportateurs pour OpenTelemetry
- Pour des exemples d'observabilité dans le Outils AWS pour PowerShell, voir Observabilité dans le guide PowerShell d'utilisation des outils.

Configurez un `TelemetryProvider`

Vous pouvez configurer un `TelemetryProvider` dans votre application globalement pour tous les clients de service ou pour des clients individuels, comme indiqué dans les exemples suivants. La the section called “Fournisseurs de services de télémétrie” section contient des informations sur les implémentations de télémétrie, notamment des informations sur les implémentations fournies avec le SDK.

Configuration du fournisseur de télémétrie global par défaut

Par défaut, chaque client de service tente d'utiliser le fournisseur de télémétrie disponible dans le monde entier. De cette façon, vous pouvez définir le fournisseur une seule fois, et tous les clients l'utiliseront. Cela ne doit être fait qu'une seule fois, avant de créer des clients de service.

L'extrait de code suivant explique comment définir le fournisseur de télémétrie global. Il crée ensuite un client de service Amazon S3 et tente d'effectuer une opération qui échoue. Le code ajoute à la fois le suivi et les métriques à l'application. Ce code utilise les NuGet packages suivants : `OpenTelemetry.Exporter.Console` et`OpenTelemetry.Instrumentation.AWS`.

> **Note**
>
> Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veillez également à ajouter `AWSSDK.SSO` et`AWSSDK.SSOOIDC`.

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

Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
```

```
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

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

var s3Client = new AmazonS3Client();

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

Console.Read();
```

Configuration d'un fournisseur de télémétrie pour un client de service spécifique

Vous pouvez configurer un client de service individuel auprès d'un fournisseur de télémétrie spécifique (autre que le fournisseur global). Pour ce faire, utilisez la `TelemetryProvider` classe de l'objet Config d'un constructeur de client de service. Par exemple, consultez AmazonS3Config et recherchez la propriété. `TelemetryProvider` Consultez the section called “Fournisseurs de services de télémétrie” pour plus d'informations sur les implémentations de télémétrie personnalisées.

Rubriques

- Métriques
- Fournisseurs de services de télémétrie

Métriques

Le tableau suivant répertorie les mesures de télémétrie émises par le SDK. Configurez un fournisseur de télémétrie pour rendre les métriques observables.

Quels indicateurs sont émis ?

Nom de la métrique	Unités	Type	Attributs	Description
durée de l'appel client	s	Histogramme	rpc.service, rpc.method	Durée totale de l'appel (y compris les nouvelles tentatives, le temps d'envoi ou de réception de la demande et le corps de réponse)
client.uptime	s	Histogramme	rpc. service	Le temps écoulé depuis la création d'un client
client.call.attempts	{tentative}	MonotonCounter	rpc.service, rpc.method	Le nombre de tentatives pour une opération individuelle
client.call.errors	{erreur}	MonotonCounter	rpc.service, rpc.method, exception.type	Le nombre d'erreurs associées à une opération
client.call.attempt duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour se connecter au service, envoyer la demande et récupérer le code d'état HTTP et les en-têtes (y compris le temps passé dans la file d'attente avant d'être envoyé)
client.call.resolve_endpoint_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour résoudre un point de terminaison (résolveur de point de terminaison, pas DNS) pour la demande
client.call.serialization_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour sérialiser le corps d'un message

Nom de la métrique	Unités	Type	Attributs	Description
client.call.deserialization_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour désérialiser le corps d'un message
client.call.auth.signing_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour signer une demande
client.call.auth.resolve_identity_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour acquérir une identité (telle qu'un AWS identifiant ou un jeton porteur) auprès d'un fournisseur d'identité
client.http.bytes_sent	Par	MonotonCounter	adresse du serveur	Le nombre total d'octets envoyés par le client HTTP
client.http.octets_received	Par	MonotonCounter	adresse du serveur	Le nombre total d'octets reçus par le client HTTP

Les descriptions des colonnes sont les suivantes :

- Nom de la métrique : nom de la métrique émise.
- Unités : unité de mesure de la métrique. Les unités sont données dans la notation UCUM sensible aux majuscules et minuscules (« c/s »).
- Type : type d'instrument utilisé pour capturer la métrique.
- Attributs : ensemble d'attributs (dimensions) émis avec la métrique.
- Description —Description de ce que mesure la métrique.

Fournisseurs de services de télémétrie

Le SDK fournit une implémentation en OpenTelemetrytant que fournisseur de télémétrie, décrite dans la section suivante.

Si vous avez des exigences spécifiques en matière de télémétrie, si vous avez déjà une solution de télémétrie en tête ou si vous avez besoin d'un contrôle précis sur la manière dont les données

de télémétrie sont capturées et traitées, vous pouvez également mettre en œuvre votre propre fournisseur de télémétrie.

Enregistrez votre propre implémentation auprès de la TelemetryProviderclasse. Voici un exemple simple de la façon d'enregistrer votre propre `TracerProvider` et`MeterProvider`.

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

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

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

Rubriques

- Configuration du fournisseur OpenTelemetry de télémétrie basé

Configuration du fournisseur OpenTelemetry de télémétrie basé

AWS SDK pour .NET Cela inclut la mise en œuvre d'un fournisseur de télémétrie OpenTelemetry basé. Pour plus de détails sur la façon de définir ce fournisseur en tant que fournisseur de télémétrie mondial, consultez. Configurez un `TelemetryProvider` Pour utiliser ce fournisseur de télémétrie, vous avez besoin des ressources suivantes dans votre projet :

- Le package OpenTelemetry.Instrumentation.aws. NuGet
- Un exportateur de télémétrie tel que OTLP ou Console. Pour plus d'informations, consultez la section Exportateurs dans la OpenTelemetry documentation.

L' OpenTelemetry implémentation incluse dans le SDK peut être configurée pour réduire le suivi des requêtes HTTPS, des informations d'identification et de la compression. Pour ce faire, définissez l'`SuppressDownstreamInstrumentation`option sur`true`, comme suit :

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

Pour plus d'informations sur ce fournisseur, consultez le billet de blog Enhancing Observability in the AWS SDK pour .NET with OpenTelemetry.

Informations supplémentaires sur les utilisateurs et les rôles

Pour développer AWS ou exécuter des applications .NET sur .NET AWS, vous devez disposer d'une combinaison d'utilisateurs, d'ensembles d'autorisations et de rôles de service adaptés à ces tâches.

Les utilisateurs, les ensembles d'autorisations et les rôles de service spécifiques que vous créez, ainsi que la manière dont vous les utilisez, dépendent des exigences de vos applications. Vous trouverez ci-dessous des informations complémentaires indiquant pourquoi ils peuvent être utilisés et comment les créer.

Utilisateurs et ensembles d'autorisations

Bien qu'il soit possible d'accéder aux services AWS en utilisant un compte utilisateur IAM avec des informations d'identification à long terme, il ne s'agit plus d'une bonne pratique et est à éviter. Même pendant le développement, il est recommandé de créer des utilisateurs et des ensembles d'autorisations et d'utiliser AWS IAM Identity Center des informations d'identification temporaires fournies par une source d'identité.

Pour le développement, vous pouvez employer l'utilisateur que vous avez créé ou qui vous a été attribué dans Authentification avec AWS. Si vous disposez des AWS Management Console autorisations appropriées, vous pouvez également créer différents ensembles d'autorisations avec le moins de privilèges pour cet utilisateur ou créer de nouveaux utilisateurs spécifiquement pour les projets de développement, en fournissant des ensembles d'autorisations dotés du moindre privilège. Le plan d'action que vous choisissez éventuellement dépend de votre situation.

Pour plus d'informations sur ces utilisateurs et ensembles d'autorisations et sur la façon de les créer, consultez Authentification et accès dans le guide de référence AWS SDKs et outils et Démarrage dans le guide de AWS IAM Identity Center l'utilisateur.

Rôles de service

Vous pouvez configurer un rôle AWS de service pour accéder aux AWS services pour le compte des utilisateurs. Ce type d'accès est approprié si plusieurs personnes doivent exécuter votre application à distance, par exemple sur une EC2 instance Amazon que vous avez créée à cette fin.

Le processus de création d'un rôle de service varie en fonction de la situation, mais il consiste principalement en ce qui suit.

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Cliquez sur Rôles, puis sur Créer un rôle.
3. Choisissez le AWS service, recherchez et sélectionnez EC2(par exemple), puis choisissez le cas EC2d'utilisation (par exemple).
4. Choisissez Suivant : Autorisations, puis sélectionnez les politiques appropriées pour les AWS services que votre application utilisera.

   > **Warning**
   >
   > NE choisissez PAS cette AdministratorAccesspolitique, car elle autorise les autorisations de lecture et d'écriture pour presque tous les éléments de votre compte.

5. Choisissez Next : Tags et entrez les tags de votre choix.

   Vous trouverez des informations sur les balises dans Contrôler l'accès à l'aide de balises de AWS ressources dans le guide de l'utilisateur IAM.
6. Choisissez Suivant : Réviser et fournir un nom et une description du rôle. Puis choisissez Create role (Créer un rôle).

Vous trouverez des informations de haut niveau sur les rôles IAM dans Identités (utilisateurs, groupes et rôles) dans le guide de l'utilisateur IAM. Vous trouverez des informations détaillées sur les rôles dans la rubrique Rôles IAM de ce guide.

Informations supplémentaires sur les rôles

- Utilisez des rôles IAM pour les tâches Amazon Elastic Container Service (Amazon ECS).
- Utilisez les rôles IAM pour les applications qui s'exécutent sur des EC2 instances Amazon.

Configuration avancée pour votre AWS SDK pour .NET projet

Les rubriques de cette section contiennent des informations sur les tâches et méthodes de configuration supplémentaires susceptibles de vous intéresser.

Rubriques

- À l'aide de AWSSDK .Extensions. NETCore.Setup et interface IConfiguration
- Configuration d'autres paramètres de l'application
- Référence des fichiers de configuration pour AWS SDK pour .NET

À l'aide de AWSSDK .Extensions. NETCore.Setup et interface IConfiguration

(Cette rubrique était auparavant intitulée « Configuration du AWS SDK pour .NET avec .NET Core »)

L'un des principaux changements apportés à .NET Core est la suppression des normes `ConfigurationManager` `app.config` et des `web.config` fichiers utilisés avec les applications .NET Framework et ASP.NET.

La configuration dans .NET Core est basée sur des paires clé-valeur établies par les fournisseurs de configuration. Les fournisseurs de configuration lisent les données de configuration en paires clé-valeur à partir de diverses sources de configuration, y compris des arguments de ligne de commande, des fichiers de répertoire, des variables d'environnement et des fichiers de paramètres.

> **Note**
>
> Pour plus d'informations, consultez Configuration in ASP.NET Core.

Pour faciliter l'utilisation AWS SDK pour .NET avec .NET Core, vous pouvez utiliser les AWSSDK.Extensions. NETCore NuGet .Package d'installation. Comme de nombreuses

bibliothèques .NET Core, elle ajoute des méthodes d'extension à l'`IConfiguration`interface pour faciliter la AWS configuration.

Le code source de ce package se trouve GitHub surhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

À l'aide de AWSSDK .Extensions. NETCore.Configuration

Supposons que vous créiez une application ASP.NET Core Model-View-Controller (MVC), ce qui peut être réalisé à l'aide du modèle d'application Web ASP.NET Core dans Visual Studio ou en l'exécutant `dotnet new mvc ...` dans la CLI .NET Core. Lorsque vous créez une telle application, le constructeur de `Startup.cs` gère la configuration en lisant diverses sources d'entrée provenant de fournisseurs de configuration tels que`appsettings.json`.

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

Pour utiliser l'`Configuration`objet afin d'obtenir les AWSoptions, ajoutez d'abord le `AWSSDK.Extensions.NETCore.Setup` NuGet package. Ajoutez ensuite vos options au fichier de configuration comme décrit ci-dessous.

Notez que l'un des fichiers ajoutés à votre projet est`appsettings.Development.json`. Cela correspond à un `EnvironmentName` ensemble de développement. Au cours du développement, vous insérez votre configuration dans ce fichier, qui n'est lu que lors des tests locaux. Lorsque vous déployez une instance Amazon EC2 `EnvironmentName` définie sur Production, ce fichier est ignoré et les informations d'identification IAM et la AWS SDK pour .NET région configurées pour l'instance Amazon EC2 sont renvoyées.

Les paramètres de configuration suivants présentent des exemples de valeurs que vous pouvez ajouter dans le `appsettings.Development.json` fichier de votre projet pour fournir des AWS paramètres.

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
  },
```

```
  "SupportEmail": "TechSupport@example.com"
}
```

Pour accéder à un paramètre dans un fichier CSHTML, utilisez la directive. `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>
```

Pour accéder aux AWS options définies dans le fichier à partir du code, appelez la méthode d'`GetAWSOptions`extension ajoutée à`IConfiguration`.

Pour construire un client de service à partir de ces options, appelez `CreateServiceClient`. L'exemple suivant montre comment créer un client de service Amazon S3. (Assurez-vous d'ajouter le NuGet package AWSSDK.S3 à votre projet.)

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

Vous pouvez également créer plusieurs clients de service avec des paramètres incompatibles en utilisant plusieurs entrées dans le `appsettings.Development.json` fichier, comme indiqué dans les exemples suivants où la configuration pour `service1` inclut la `us-west-2` région et la configuration pour `service2` inclut l'URL du point de terminaison spécial.

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

Vous pouvez ensuite obtenir les options pour un service spécifique en utilisant l'entrée dans le fichier JSON. Par exemple, pour obtenir les paramètres, `service1` utilisez ce qui suit.

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

Valeurs autorisées dans le fichier appsettings

Les valeurs de configuration d'application suivantes peuvent être définies dans le fichier `appsettings.Development.json`. Les noms des champs doivent utiliser le boîtier illustré. Pour plus de détails sur ces paramètres, consultez le `AWS.Runtime.ClientConfig` cours.

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

Injection de dépendances ASP.NET Core

Les AWSSDK.Extensions. NETCoreLe NuGet package .Setup s'intègre également à un nouveau système d'injection de dépendances dans ASP.NET Core. La `ConfigureServices` méthode de la

`Startup` classe de votre application est celle où les services MVC sont ajoutés. Si l'application utilise Entity Framework, c'est également là que l'initialisation se produit.

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

> **Note**
>
> Des informations générales sur l'injection de dépendances dans .NET Core sont disponibles sur le site de documentation .NET Core.

Le `AWSSDK.Extensions.NETCore.Setup` NuGet package ajoute de nouvelles méthodes d'extension `IServiceCollection` que vous pouvez utiliser pour ajouter AWS des services à l'injection de dépendances. Le code suivant explique comment ajouter les AWS options lues `IConfiguration` pour ajouter Amazon S3 et DynamoDB à la liste des services. (Assurez-vous d'ajouter les DBv2 NuGet packages AWSSDK.S3 et AWSSDK.Dynamo à votre projet.)

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

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

Désormais, si vos contrôleurs MVC utilisent `IAmazonS3` ou `IAmazonDynamoDB` en tant que paramètres dans leurs constructeurs, le système d'injection de dépendances transmet ces services.

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

    public HomeController(IAmazonS3 s3Client)
    {
```

```
        this.S3Client = s3Client;
    }

    ...

}
```

Configuration d'autres paramètres de l'application

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework. Les `Web.config` fichiers `App.config` et ne sont pas présents par défaut dans les projets basés sur .NET Core.

Ouvrir pour afficher le contenu du .NET Framework

Vous pouvez configurer un certain nombre de paramètres d'application :

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS Points de terminaison générés par le service

Ces paramètres peuvent être configurés dans le fichier `App.config` ou `Web.config` de l'application. Bien que vous puissiez également les configurer à l'aide de l' AWS SDK pour .NET API, nous vous recommandons d'utiliser le `.config` fichier de l'application. Les deux approches sont décrites ici.

Pour plus d'informations sur l'utilisation de l'`<aws>`élément, comme décrit plus loin dans cette rubrique, consultez la section Référence des fichiers de configuration pour AWS SDK pour .NET.

AWSLogging

Configure comment SDK doit consigner les événements, le cas échéant. Par exemple, l'approche recommandée consiste à utiliser l'élément `<logging>`, élément enfant de l'élément `<aws>` :

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

Autrement :

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

Les valeurs possibles sont :

`None`

Désactivez la journalisation des événements . Il s'agit de l'option par défaut.

`log4net`

Journaliser à l'aide de log4net.

`SystemDiagnostics`

Journaliser à l'aide de la classe `System.Diagnostics`.

Vous pouvez définir plusieurs valeurs pour l'attribut `logTo`, séparées par des virgules. L'exemple suivant définit la journalisation `log4net` et `System.Diagnostics` dans le fichier `.config` :

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

Autrement :

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

Vous pouvez également, à l'aide de l' AWS SDK pour .NET API, combiner les valeurs de l'LoggingOptionsénumération et définir la propriété AWSConfigs.Logging :

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

Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSLogMétriques

Spécifie si SDK doit ou non journaliser les métriques de performance. Pour définir la configuration de journalisation des métriques dans le fichier `.config`, définissez la valeur de l'attribut `logMetrics` dans l'élément `<logging>`, élément enfant de l'élément `<aws>` :

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

Sinon, définissez la clé `AWSLogMetrics` dans la section `<appSettings>` :

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

Sinon, pour définir la journalisation des métriques avec l' AWS SDK pour .NET API, définissez le AWSConfigs. LogMetricspropriété :

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

Ce paramètre configure la propriété `LogMetrics` par défaut pour tous les clients et toutes les configurations. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSRegion

Configure la AWS région par défaut pour les clients qui n'ont pas explicitement spécifié de région. Pour définir la région dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `region` dans l'élément `aws` :

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

Sinon, définissez la clé AWSRegion dans la section `<appSettings>` :

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

Sinon, pour définir la région à l'aide de l' AWS SDK pour .NET API, définissez le AWSConfigs. AWSRegionpropriété :

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

Pour plus d'informations sur la création d'un AWS client pour une région spécifique, consultez la section Sélection de AWS région. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSResponseJournalisation

Configure quand SDK doit journaliser les réponses du service. Les valeurs possibles sont :

`Never`

Les réponses du service ne sont jamais journalisées. Il s'agit de l'option par défaut.

`Always`

Les réponses du service sont toujours journalisées.

`OnError`

Journaliser les réponses du service uniquement lorsqu'une erreur se produit.

Pour définir la configuration de journalisation de service dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `logResponses` dans l'élément `<logging>`, élément enfant de l'élément `<aws>` :

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

Vous pouvez également définir la clé de AWSResponsejournalisation dans la `<appSettings>` section :

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

Sinon, pour configurer la journalisation des services avec l' AWS SDK pour .NET API, définissez le AWSConfigs. ResponseLoggingpropriété à l'une des valeurs de l'ResponseLoggingOptionénumération :

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

Les modifications apportées à ce paramètre prennent effet immédiatement.

`AWS.DynamoDBContext.TableNamePrefix`

Configure la valeur par défaut de `TableNamePrefix` ; `DynamoDBContext` est utilisé si aucune configuration manuelle n'est effectuée.

Pour définir le préfixe du nom de table dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `tableNamePrefix` dans l'élément `<dynamoDBContext>`, élément enfant de l'élément `<dynamoDB>`, lui-même élément enfant de l'élément `<aws>` :

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

Sinon, définissez la clé `AWS.DynamoDBContext.TableNamePrefix` dans la section `<appSettings>` :

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

Sinon, pour définir le préfixe du nom de table avec l' AWS SDK pour .NET API, définissez la propriété AWSConfigs.Dynamo DBContext TableNamePrefix :

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

Les modifications apportées à ce paramètre prennent effet uniquement pour les instances nouvellement construites de `DynamoDBContextConfig` et `DynamoDBContext`.

`AWS.S3.UseSignatureVersion4`

Configure si le client Amazon S3 doit ou non utiliser la signature version 4 pour signer avec les demandes.

Pour définir la signature de la version 4 pour Amazon S3 dans le `.config` fichier, l'approche recommandée consiste à définir l'`useSignatureVersion4`attribut de l'`<s3>`élément, qui est un élément enfant de l'`<aws>`élément :

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

Vous pouvez également définir la `AWS.S3.UseSignatureVersion4` clé sur `true` dans la `<appSettings>` section :

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

Sinon, pour définir la signature de la version 4 de la signature avec l' AWS SDK pour .NET API, définissez la propriété AWSConfigs.S3 UseSignatureVersion 4 sur `true` :

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

Par défaut, ce paramètre est `false`, mais Signature version 4 peut être utilisé par défaut dans certains cas ou avec certaines régions. Lorsque le paramètre est `true`, Signature version 4 est utilisé pour toutes les demandes. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances du client Amazon S3.

AWSEndpointDéfinition

Configure si SDK doit utiliser un fichier de configuration personnalisé qui définit les régions et les points de terminaison.

Pour définir le fichier de définition de point de terminaison dans le fichier `.config`, nous vous recommandons de définir la valeur de l'attribut `endpointDefinition` dans l'élément `<aws>`.

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

Vous pouvez également définir la clé de AWSEndpointdéfinition dans la `<appSettings>` section :

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

Sinon, pour définir le fichier de définition du point de terminaison avec l' AWS SDK pour .NET API, définissez le AWSConfigs. EndpointDefinitionpropriété :

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

Si vous n'indiquez pas de nom de fichier, aucun fichier de configuration personnalisé ne sera utilisé. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes. Le `endpoints.json` fichier est disponible à l'adressehttps://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS Points de terminaison générés par le service

Certains AWS services génèrent leurs propres points de terminaison au lieu de consommer un point de terminaison régional. Les clients de ces services utilisent une URL de service spécifique à chaque service et à vos ressources. Amazon CloudSearch et AWS IoT. Les exemples suivants illustrent comment obtenir les points de terminaison pour ces services.

Exemple de CloudSearch point de terminaison Amazon

Le CloudSearch client Amazon est utilisé pour accéder au service CloudSearch de configuration Amazon. Vous utilisez le service CloudSearch de configuration Amazon pour créer, configurer et gérer des domaines de recherche. Pour créer un domaine de recherche, créez un CreateDomainRequestobjet et indiquez la `DomainName` propriété. Créez un AmazonCloudSearchClientobjet à l'aide de l'objet de demande. Appelez la méthode CreateDomain. L'CreateDomainResponseobjet renvoyé par l'appel contient une `DomainStatus` propriété qui possède à la fois les `SearchService` points de terminaison `DocService` et. Créez un AmazonCloudSearchDomainConfigobjet et utilisez-le pour initialiser `DocService` les `SearchService` instances de la AmazonCloudSearchDomainClientclasse.

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

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

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

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

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

AWS IoT Exemple de points de terminaison

Pour obtenir le point de terminaison pour AWS IoT, créez un AmazonIoTClientobjet et appelez la DescribeEndPointméthode. L'DescribeEndPointResponseobjet renvoyé contient le`EndpointAddress`. Créez un AmazonIotDataConfigobjet, définissez la `ServiceURL` propriété et utilisez-le pour instancier la AmazonIotDataClientclasse.

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

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

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

Référence des fichiers de configuration pour AWS SDK pour .NET

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework. Les `Web.config` fichiers `App.config` et ne sont pas présents par défaut dans les projets basés sur .NET Core.

Ouvrir pour afficher le contenu du .NET Framework

Vous pouvez utiliser un projet `App.config` ou un `Web.config` fichier .NET pour spécifier des AWS paramètres, tels que les AWS informations d'identification, les options de journalisation, les points de terminaison de AWS service et AWS les régions, ainsi que certains paramètres pour AWS des services tels qu'Amazon DynamoDB EC2, Amazon et Amazon S3. Vous trouverez ci-dessous des informations expliquant comment mettre correctement en forme un fichier `App.config` ou `Web.config` pour spécifier ces types de paramètres.

> **Note**
>
> Bien que vous puissiez continuer à utiliser l'`<appSettings>`élément dans un `Web.config` fichier `App.config` OR pour spécifier des AWS paramètres, nous vous recommandons d'utiliser les `<aws>` éléments `<configSections>` et comme décrit plus loin dans cette

rubrique. Pour plus d'informations sur l'`<appSettings>`élément, consultez les exemples d'`<appSettings>`éléments dans Configuration de votre AWS SDK pour .NET application.

> **Note**
>
> Bien que vous puissiez continuer à utiliser les propriétés de `AWSConfigs`classe suivantes dans un fichier de code pour spécifier des AWS paramètres, les propriétés suivantes sont obsolètes et ne seront peut-être pas prises en charge dans les versions futures :
>
> - `DynamoDBContextTableNamePrefix`
> - `EC2UseSignatureVersion4`
> - `LoggingOptions`
> - `LogMetrics`
> - `ResponseLoggingOption`
> - `S3UseSignatureVersion4`
>
> En général, nous recommandons qu'au lieu d'utiliser les propriétés de `AWSConfigs` classe d'un fichier de code pour spécifier AWS les paramètres, vous utilisiez les `<aws>` éléments `<configSections>` et d'un `Web.config` fichier `App.config` or pour spécifier AWS les paramètres, comme décrit plus loin dans cette rubrique. Pour plus d'informations sur les propriétés précédentes, consultez les exemples de `AWSConfigs` code dans Configuration de votre AWS SDK pour .NET application.

Rubriques

- Déclarer une section AWS de paramètres
- Éléments autorisés
- Informations de référence sur les éléments

Déclarer une section AWS de paramètres

Vous définissez AWS les paramètres dans un `Web.config` fichier `App.config` or à partir de l'`<aws>`élément. Avant de commencer à utiliser l'élément `<aws>`, vous devez créer un élément `<section>` (qui est un élément enfant de l'élément `<configSections>`) et affecter à son attribut

name la valeur aws et à son attribut type la valeur Amazon.AWSSection, AWSSDK.Core, comme dans l'exemple suivant :

```
<?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>
```

L'éditeur Visual Studio ne fournit pas de saisie automatique du code (IntelliSense) pour l'<aws>élément ou ses éléments enfants.

Pour vous aider à créer une version correctement mise en forme de l'élément <aws>, appelez la méthode Amazon.AWSConfigs.GenerateConfigTemplate. Celle-ci génère une version canonique de l'élément <aws> sous la forme d'une chaîne bien formée, que vous pouvez adapter à vos besoins. Les sections suivantes décrivent les attributs et les éléments enfants de l'élément <aws>.

Éléments autorisés

Voici une liste des relations logiques entre les éléments autorisés dans une section de AWS paramètres. Vous pouvez générer la dernière version de cette liste en appelant la méthode Amazon.AWSConfigs.GenerateConfigTemplate, qui génère une version canonique de l'élément <aws> sous la forme d'une chaîne que vous pouvez adapter à vos besoins.

```
<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>
```

Informations de référence sur les éléments

Voici une liste des éléments autorisés dans une section de AWS paramètres. Pour chaque élément, ses attributs autorisés et éléments parents-enfants sont répertoriés.

Rubriques

- alias
- aws
- dynamoDB

- dynamo DBContext
- ec2
- journalisation
- map
- property
- proxy
- s3

alias

L'élément <alias> représente un élément unique dans une collection d'un ou plusieurs mappages entre la table de départ et la table d'arrivée qui spécifie une table différente de celle configurée pour un type. Cet élément est mappé à une instance de la classe Amazon.Util.TableAlias à partir de la propriété Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases du AWS SDK pour .NET. Le remappage intervient avant l'application d'un préfixe de nom de table.

Cet élément peut inclure les attributs suivants :

fromTable

Partie « table de départ » du mappage entre la table de départ et la table d'arrivée. Cet attribut est mappé à la propriété Amazon.Util.TableAlias.FromTable du AWS SDK pour .NET.

toTable

Partie « table d'arrivée » du mappage entre la table de départ et la table d'arrivée. Cet attribut est mappé à la propriété Amazon.Util.TableAlias.ToTable du AWS SDK pour .NET.

Le parent de l'élément <alias> est l'élément <tableAliases>.

L'élément <alias> ne contient aucun élément enfant.

Voici un exemple de l'élément <alias> utilisé :

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

aws

L'<aws>élément représente l'élément le plus haut d'une section de AWS paramètres. Cet élément peut inclure les attributs suivants :

endpointDefinition

Le chemin absolu vers un fichier de configuration personnalisé qui définit les AWS régions et les points de terminaison à utiliser. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.EndpointDefinition` du AWS SDK pour .NET.

profileName

Le nom du profil pour les AWS informations d'identification stockées qui seront utilisées pour effectuer des appels de service. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSProfileName` du AWS SDK pour .NET.

profilesLocation

Le chemin absolu vers l'emplacement du fichier d'informations d'identification partagé avec d'autres AWS SDKs. Par défaut, le fichier d'informations d'identification est stocké dans le répertoire `.aws` situé dans le répertoire de base de l'utilisateur actif. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSProfilesLocation` du AWS SDK pour .NET.

region

L'ID de AWS région par défaut pour les clients qui n'ont pas explicitement spécifié de région. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSRegion` du AWS SDK pour .NET.

L'élément <aws> n'a pas d'élément parent.

L'élément <aws> peut inclure les éléments enfants suivants :

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

Voici un exemple de l'élément <aws> utilisé :

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

dynamoDB

L'élément <dynamoDB> représente un ensemble de paramètres pour Amazon DynamoDB. Cet élément peut inclure l'attribut conversionSchema, qui représente la version à utiliser pour la conversion entre des objets .NET et des objets DynamoDB. Les valeurs autorisées sont notamment V1 et V2. Cet attribut est mappé à la classe `Amazon.DynamoDBv2.DynamoDBEntryConversion` du AWS SDK pour .NET. Pour plus d'informations, consultez DynamoDB Series – Conversion Schemas.

Le parent de l'élément <dynamoDB> est l'élément <aws>.

L'élément <dynamoDB> peut inclure l'élément enfant <dynamoDBContext>.

Voici un exemple de l'élément <dynamoDB> utilisé :

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

dynamo DBContext

L'élément <dynamoDBContext> représente un ensemble de paramètres propres au contexte Amazon DynamoDB. Cet élément peut inclure l'tableNamePrefixattribut, qui représente le préfixe de nom de table par défaut que le contexte DynamoDB utilisera s'il n'est pas configuré manuellement. Cet attribut est mappé à la propriété `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` à partir de la propriété `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` du AWS SDK pour .NET. Pour plus d'informations, consultez Enhancements to the DynamoDB SDK.

Le parent de l'élément <dynamoDBContext> est l'élément <dynamoDB>.

L'élément <dynamoDBContext> peut inclure les éléments enfants suivants :

- <alias> (une ou plusieurs instances)
- <map> (une ou plusieurs instances)

Voici un exemple de l'élément <dynamoDBContext> utilisé :

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

ec2

L'<ec2>élément représente un ensemble de EC2 paramètres Amazon. Cet élément peut inclure l'attribut useSignatureVersion4, qui indique si la signature version 4 sera utilisée pour toutes les demandes (vrai) ou si la signature version 4 ne sera pas utilisée pour toutes les demandes (faux, valeur par défaut). Cet attribut est mappé à la propriété Amazon.Util.EC2Config.UseSignatureVersion4 à partir de la propriété Amazon.AWSConfigs.EC2Config.UseSignatureVersion4 du AWS SDK pour .NET.

Le parent de l'<ec2>élément est l'élément.

L'élément <ec2> ne contient aucun élément enfant.

Voici un exemple de l'élément <ec2> utilisé :

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

journalisation

L'élément <logging> représente un ensemble de paramètres pour la journalisation des réponses et la journalisation des métriques de performance. Cet élément peut inclure les attributs suivants :

logMetrics

Indique si les métriques de performance sont journalisées pour tous les clients et toutes les configurations (true) ; sinon, false. Cet attribut est mappé à la

propriété `Amazon.Util.LoggingConfig.LogMetrics` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetrics` du AWS SDK pour .NET.

`logMetricsCustomFormatter`

Type de données et nom d'assembly d'un formateur personnalisé pour les métriques de performance. Cet attribut est mappé à la propriété `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` du AWS SDK pour .NET.

`logMetricsFormat`

Format sous lequel se présentent les métriques de journalisation (mappé à la propriété `Amazon.Util.LoggingConfig.LogMetricsFormat` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` du AWS SDK pour .NET).

Les valeurs autorisées sont les suivantes :

`JSON`

Utilise le format JSON.

`Standard`

Utilise le format par défaut.

`logResponses`

Indique à quel moment les réponses du service sont journalisées (mappé à la propriété `Amazon.Util.LoggingConfig.LogResponses` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogResponses` du AWS SDK pour .NET).

Les valeurs autorisées sont les suivantes :

`Always`

Les réponses du service sont toujours journalisées.

`Never`

Les réponses du service ne sont jamais journalisées.

`OnError`

Les réponse du service sont journalisées uniquement en cas d'erreur.

`logTo`

Où se connecter (cartes de la `LogTo` propriété depuis la `Amazon.AWSConfigs.LoggingConfig.LogTo` propriété dans le AWS SDK pour .NET).

Les valeurs autorisées comprennent une ou plusieurs des valeurs suivantes :

`Log4Net`

Journalisation dans log4net.

`None`

Désactivation de la journalisation.

`SystemDiagnostics`

Journalisation dans System.Diagnostics.

Le parent de l'élément `<logging>` est l'élément `<aws>`.

L'élément `<logging>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<logging>` utilisé :

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

map

L'`<map>`élément représente un élément unique d'une collection de type-to-table mappages entre les types .NET et les tables DynamoDB (correspond à une instance de `TypeMapping` la classe à partir de la propriété `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` du). AWS SDK pour .NET Pour plus d'informations, consultez Enhancements to the DynamoDB SDK.

Cet élément peut inclure les attributs suivants :

`targetTable`

Table DynamoDB à laquelle le mappage s'applique. Cet attribut est mappé à la propriété `Amazon.Util.TypeMapping.TargetTable` du AWS SDK pour .NET.

`type`

Type et nom d'assembly auxquels le mappage s'applique. Cet attribut est mappé à la propriété `Amazon.Util.TypeMapping.Type` du AWS SDK pour .NET.

Le parent de l'élément `<map>` est l'élément `<dynamoDBContext>`.

L'élément `<map>` peut inclure une ou plusieurs instances de l'élément enfant `<property>`.

Voici un exemple de l'élément `<map>` utilisé :

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

property

L'élément `<property>` représente une propriété DynamoDB. (Cet élément correspond à une instance d'Amazon.Util. PropertyConfig classe issue de la `AddProperty` méthode décrite dans le AWS SDK pour .NET) Pour plus d'informations, consultez Améliorations apportées au SDK DynamoDB et aux attributs DynamoDB.

Cet élément peut inclure les attributs suivants :

`attribute`

Nom d'un attribut de la propriété, par exemple le nom d'une clé de plage. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Attribute` du AWS SDK pour .NET.

`converter`

Type du convertisseur qui doit être utilisé pour cette propriété. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Converter` du AWS SDK pour .NET.

`ignore`

Indique si la propriété associée doit être ignorée (true) ; sinon, false. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Ignore` du AWS SDK pour .NET.

`name`

Le nom de la propriété. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Name` du AWS SDK pour .NET.

`version`

Indique si cette propriété doit stocker le numéro de version de l'élément (true) ; sinon, false. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Version` du AWS SDK pour .NET.

Le parent de l'élément `<property>` est l'élément `<map>`.

L'élément `<property>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<property>` utilisé :

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

proxy

L'élément `<proxy>` représente les paramètres de configuration d'un proxy pour le AWS SDK pour .NET à utiliser. Cet élément peut inclure les attributs suivants :

hôte

Nom d'hôte ou adresse IP du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Host` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Host` du AWS SDK pour .NET.

mot de passe

Mot de passe permettant de s'authentifier auprès du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Password` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Password` du AWS SDK pour .NET.

port

Numéro de port du proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Port` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Port` du AWS SDK pour .NET.

nom d’utilisateur

Nom d'utilisateur permettant de s'authentifier auprès du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Username` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Username` du AWS SDK pour .NET.

Le parent de l'élément `<proxy>` est l'élément `<aws>`.

L'élément `<proxy>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<proxy>` utilisé :

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

s3

L'élément `<s3>` représente un ensemble de paramètres Amazon S3. Cet élément peut inclure l'attribut useSignatureVersion4, qui indique si la signature version 4 sera utilisée pour toutes les demandes (vrai) ou si la signature version 4 ne sera pas utilisée pour toutes les demandes (faux, valeur par défaut). Cet attribut est mappé à la propriété `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` du AWS SDK pour .NET.

Le parent de l'élément `<s3>` est l'élément `<aws>`.

L'élément `<s3>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<s3>` utilisé :

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

Utilisation d’informations d'identification existantes

Les rubriques de cette section fournissent des informations sur l'utilisation d'informations d'identification à long terme ou à court terme sans utiliser AWS IAM Identity Center.

> **Warning**
>
> Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que AWS IAM Identity Center.

> **Note**
>
> Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section Autres méthodes d'authentification dans le guide de référence des outils AWS SDKs et.
>
> Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dansAuthentification avec AWS.

Avertissements et conseils importants concernant les informations d'identification

Avertissements concernant les informations d'identification

- N'utilisez PAS les informations d'identification root de votre compte pour accéder aux AWS ressources. Ces informations d'identification offrent un accès illimité au compte et sont difficiles à révoquer.
- N'incluez PAS de clés d'accès littérales ou d'informations d'identification dans vos fichiers d'application. Vous risqueriez en effet d'exposer accidentellement vos informations d'identification si, par exemple, vous chargiez le projet sur un référentiel public.
- N'incluez PAS de fichiers contenant des informations d'identification dans votre zone de projet.
- Sachez que toutes les informations d'identification stockées dans le AWS `credentials` fichier partagé sont stockées en texte brut.

Conseils supplémentaires pour gérer les informations d'identification en toute sécurité

Pour une discussion générale sur la manière de gérer en toute sécurité les AWS informations d'identification, consultez AWS les informations d'identification de sécurité dans le Références générales AWSguide de l'utilisateur IAM et les meilleures pratiques de sécurité et les cas d'utilisation. En plus de ces exposés, envisagez de prendre les mesures suivantes :

- Créez des utilisateurs supplémentaires, tels que des utilisateurs dans IAM Identity Center, et utilisez leurs informations d'identification au lieu d'utiliser vos informations d'identification d'utilisateur root AWS . Les informations d'identification des autres utilisateurs peuvent être révoquées si nécessaire ou sont temporaires par nature. En outre, vous pouvez appliquer une politique à chaque utilisateur pour qu'il n'accède qu'à certaines ressources et actions et ainsi adopter les autorisations du moindre privilège.

- Utilisez des rôles IAM pour les tâches Amazon Elastic Container Service (Amazon ECS).

- Utilisez des rôles IAM pour les applications qui s'exécutent sur des instances Amazon EC2.

- Utilisez des informations d'identification temporaires ou des variables d'environnement pour les applications accessibles aux utilisateurs extérieurs à votre organisation.

Rubriques

- Utilisation du fichier AWS d'informations d'identification partagé
- Utilisation du SDK Store (Windows uniquement)

Utilisation du fichier AWS d'informations d'identification partagé

(N'oubliez pas de consulter les avertissements et les instructions importants concernant les informations d'identification.)

L'un des moyens de fournir des informations d'identification pour vos applications consiste à créer des profils dans le fichier AWS d'informations d'identification partagé, puis à les stocker dans ces profils. Ce fichier peut être utilisé par l'autre AWS SDKs. Il peut également être utilisé par les AWS CLIAWS Tools for Windows PowerShell, et les AWS boîtes à outils pour Visual Studio et VS Code. JetBrains

> **Warning**
>
> Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que AWS IAM Identity Center.

> **Note**
>
> Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section Autres méthodes d'authentification dans le guide de référence des outils AWS SDKs et.
> Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dansAuthentification avec AWS.

Informations générales

Par défaut, le fichier AWS d'informations d'identification partagé se trouve dans le `.aws` répertoire de votre répertoire personnel et porte le nom `credentials` `~/.aws/credentials` (Linux ou macOS) ou `%USERPROFILE%\.aws\credentials` (Windows). Pour plus d'informations sur les emplacements alternatifs, consultez la section Emplacement des fichiers partagés dans le Guide de référence des outils AWS SDKs et des outils. Voir aussi Accès aux informations d'identification et aux profils dans une application.

Le fichier AWS d'informations d'identification partagé est un fichier en texte brut qui suit un certain format. Pour plus d'informations sur le format des fichiers AWS d'informations d'identification, consultez la section Format du fichier d'informations d'identification dans le guide de référence des outils AWS SDKs et.

Vous pouvez gérer les profils dans le fichier AWS d'informations d'identification partagé de plusieurs manières.

- Utilisez n'importe quel éditeur de texte pour créer et mettre à jour le fichier AWS d'informations d'identification partagé.

- Utilisez l'Amazon.Runtime. CredentialManagementespace de noms de l' AWS SDK pour .NET API, comme indiqué plus loin dans cette rubrique.
- Utilisez des commandes et des procédures pour Outils AWS pour PowerShellet les AWS boîtes à outils pour Visual Studio et VS Code. JetBrains
- Utilisez AWS CLIdes commandes, par exemple, `aws configure set aws_access_key_id` et`aws configure set aws_secret_access_key`.

Exemples de gestion de profils

Les sections suivantes présentent des exemples de profils dans le fichier AWS d'informations d'identification partagé. Certains exemples montrent le résultat, qui peut être obtenu par le biais de l'une des méthodes de gestion des informations d'identification décrites précédemment. D'autres exemples montrent comment utiliser une méthode particulière.

Le profil par défaut

Le fichier AWS d'informations d'identification partagé aura presque toujours un profil nommé default. C'est ici qu'ils AWS SDK pour .NET recherchent les informations d'identification si aucun autre profil n'est défini.

Le `[default]` profil ressemble généralement à ce qui suit.

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

Création d'un profil par programmation

Cet exemple montre comment créer un profil et l'enregistrer dans le fichier d' AWS informations d'identification partagé par programmation. Il utilise les classes suivantes de l'Amazon.Runtime. CredentialManagementnamespace : CredentialProfileOptionsCredentialProfile, et. SharedCredentialsFile

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

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

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

> **Warning**
>
> Un tel code ne devrait généralement pas figurer dans votre application. Si vous l'incluez dans votre application, prenez les précautions nécessaires pour vous assurer que les clés en texte brut ne soient pas visibles dans le code, sur le réseau ou même dans la mémoire de l'ordinateur.

Le profil créé par cet exemple est le suivant.

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

Mettre à jour un profil existant par programmation

Cet exemple montre comment mettre à jour par programmation le profil créé précédemment. Il utilise les classes suivantes de l'Amazon.Runtime. CredentialManagementespace de noms : CredentialProfileet. SharedCredentialsFile Il utilise également la RegionEndpointclasse de l'espace de noms 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);
    }
}
```

Le profil mis à jour est le suivant.

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

> **Note**
>
> Vous pouvez également définir la AWS région dans d'autres emplacements et en utilisant d'autres méthodes. Pour de plus amples informations, veuillez consulter Configuration de AWS la région pour AWS SDK pour .NET.

Utilisation du SDK Store (Windows uniquement)

(Assurez-vous de consulter les mises en garde et les directives importantes.)

Sous Windows, le SDK Store est un autre endroit où vous pouvez créer des profils et stocker des informations d'identification chiffrées pour votre AWS SDK pour .NET application. Il est situé dans`%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`. Vous pouvez utiliser le SDK Store pendant le développement comme alternative au fichier d' AWS informations d'identification partagé.

> **Warning**
>
> Afin d’éviter les risques de sécurité, n’employez pas les utilisateurs IAM pour l’authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des

données réelles. Préférez la fédération avec un fournisseur d'identité tel que AWS IAM Identity Center.

> **Note**
>
> Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section Autres méthodes d'authentification dans le guide de référence des outils AWS SDKs et.
> Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dansAuthentification avec AWS.

Informations générales

Le SDK Store offre les avantages suivants :

- Les informations d'identification du SDK Store sont cryptées et le SDK Store se trouve dans le répertoire personnel de l'utilisateur. Cela limite le risque d'exposition accidentelle de vos informations d'identification.
- Le SDK Store fournit également des informations d'identification aux AWS Tools for Windows PowerShellAWS Toolkit for Visual Studioet.

Les profils SDK Store sont spécifiques à un utilisateur spécifique sur un hôte donné. Vous ne pouvez pas les copier vers d'autres hôtes ou d'autres utilisateurs. Cela signifie que vous ne pouvez pas réutiliser les profils SDK Store présents sur votre machine de développement pour d'autres hôtes ou machines de développement. Cela signifie également que vous ne pouvez pas utiliser les profils SDK Store dans les applications de production.

Vous pouvez gérer les profils dans le SDK Store de la manière suivante :

- Utilisez l'interface utilisateur graphique (GUI) du AWS Toolkit for Visual Studio.
- Utilisez l'Amazon.Runtime. CredentialManagementespace de noms de l' AWS SDK pour .NET API, comme indiqué plus loin dans cette rubrique.

- Utilisez les commandes du AWS Tools for Windows PowerShell; par exemple, `Set-AWSCredential` et`Remove-AWSCredentialProfile`.

Exemples de gestion de profils

Les exemples suivants vous montrent comment créer et mettre à jour par programmation un profil dans le SDK Store.

Création d'un profil par programmation

Cet exemple montre comment créer un profil et l'enregistrer dans le SDK Store par programmation. Il utilise les classes suivantes de l'Amazon.Runtime. CredentialManagementnamespace : CredentialProfileOptionsCredentialProfile, et Net SDKCredentials File.

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

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

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

> **Warning**
>
> Un tel code ne devrait généralement pas figurer dans votre application. S'il est inclus dans votre application, prenez les précautions nécessaires pour vous assurer que les clés en texte brut ne soient pas visibles dans le code, sur le réseau ou même dans la mémoire de l'ordinateur.

Le profil créé par cet exemple est le suivant.

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

Mettre à jour un profil existant par programmation

Cet exemple montre comment mettre à jour par programmation le profil créé précédemment. Il utilise les classes suivantes de l'Amazon.Runtime. CredentialManagementnamespace : CredentialProfileet Net SDKCredentials File. Il utilise également la RegionEndpointclasse de l'espace de noms 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);
    }
}
```

Le profil mis à jour est le suivant.

```
"[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**
>
> Vous pouvez également définir la AWS région dans d'autres emplacements et en utilisant d'autres méthodes. Pour de plus amples informations, veuillez consulter Configuration de AWS la région pour AWS SDK pour .NET.

À l'aide du AWS SDK pour .NET

Cette section fournit des informations sur les fonctionnalités AWS SDK pour .NET que vous devrez peut-être prendre en compte lors de la création de vos applications.

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet.

Pour plus d'informations sur le développement de logiciels pour des AWS services spécifiques ainsi que des exemples de code, consultez AWS Services d'appels. Pour des exemples de code supplémentaires, voirSDK pour .NET exemples de code (v4).

Rubriques

- Programmation asynchrone à l'aide du AWS SDK pour .NET
- Utilisation de résultats paginés dans le AWS SDK pour .NET
- Support du protocole HTTP 2 dans AWS SDK pour .NET
- Outils supplémentaires

Programmation asynchrone à l'aide du AWS SDK pour .NET

AWS SDK pour .NET utilise le modèle asynchrone basé sur les tâches (TAP) pour son implémentation asynchrone. Pour en savoir plus sur le TAP, consultez la section Modèle asynchrone basé sur les tâches (TAP) sur docs.microsoft.com.

Cette rubrique vous donne un aperçu de la façon d'utiliser le TAP dans vos appels aux clients AWS du service.

Les méthodes asynchrones de l' AWS SDK pour .NET API sont des opérations basées sur la `Task` classe ou la `Task<TResult>` classe. Consultez le site docs.microsoft.com pour obtenir des informations sur ces classes : classe Task, Task< > class. TResult

Lorsque ces méthodes d'API sont appelées dans votre code, elles doivent être appelées dans une fonction déclarée avec le `async` mot clé, comme indiqué dans l'exemple suivant.

```
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;
}
```

Comme indiqué dans l'extrait de code précédent, la portée préférée de la `async` déclaration est la `Main` fonction. La définition de cette `async` étendue garantit que tous les appels aux clients du AWS service doivent être asynchrones. Si vous ne pouvez pas `Main` déclarer que vous êtes asynchrone pour une raison quelconque, vous pouvez utiliser le `async` mot clé sur des fonctions autres que `Main` puis appeler les méthodes de l'API à partir de là, comme indiqué dans l'exemple suivant.

```
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;
}
```

Notez la `Task<>` syntaxe spéciale requise `Main` lorsque vous utilisez ce modèle. En outre, vous devez utiliser le **`Result`**membre de la réponse pour obtenir les données.

Vous pouvez voir des exemples complets d'appels asynchrones à des clients AWS de service dans la Création d'une application simple section (Application multiplateforme simpleetApplication simple basée sur Windows) et dans. Exemples de code guidés

Utilisation de résultats paginés dans le AWS SDK pour .NET

Certains AWS services collectent et stockent une grande quantité de données, que vous pouvez récupérer à l'aide des appels d'API du AWS SDK pour .NET. Si la quantité de données que vous souhaitez récupérer devient trop importante pour un seul appel d'API, vous pouvez diviser les résultats en éléments plus faciles à gérer grâce à la pagination.

Pour vous permettre d'effectuer la pagination, les objets de demande et de réponse de nombreux clients de service du SDK fournissent un jeton de continuation (généralement nommé`NextToken`). Certains de ces clients de services fournissent également des paginateurs.

Les paginateurs vous permettent d'éviter la surcharge liée au jeton de continuation, qui peut impliquer des boucles, des variables d'état, plusieurs appels d'API, etc. Lorsque vous utilisez un paginateur, vous pouvez récupérer les données d'un AWS service par le biais d'une seule ligne de code, la déclaration d'une `foreach` boucle. Si plusieurs appels d'API sont nécessaires pour récupérer les données, le paginateur s'en charge pour vous.

Où puis-je trouver des paginateurs ?

Tous les services ne proposent pas de paginateurs. L'un des moyens de déterminer si un service fournit un paginateur pour une API particulière consiste à examiner la définition d'une classe de client de service dans la référence d'AWS SDK pour .NET API.

Par exemple, si vous examinez la définition de la AmazonCloudWatchLogsClientclasse, vous voyez une `Paginators` propriété. Il s'agit de la propriété qui fournit un paginateur pour Amazon CloudWatch Logs.

Que m'apportent les paginateurs ?

Les paginateurs contiennent des propriétés qui vous permettent de voir les réponses complètes. Ils contiennent également généralement une ou plusieurs propriétés qui vous permettent d'accéder aux parties les plus intéressantes des réponses, que nous appellerons les résultats clés.

Par exemple, dans ce qui `AmazonCloudWatchLogsClient` a été mentionné précédemment, l'`Paginator`objet contient une `Responses` propriété contenant l'DescribeLogGroupsResponseobjet complet issu de l'appel d'API. Cette `Responses` propriété contient, entre autres, un ensemble de groupes de journaux.

L'objet Paginator contient également un résultat clé nommé. `LogGroups` Cette propriété contient uniquement la partie de la réponse consacrée aux groupes de journaux. L'obtention de ce résultat clé vous permet de réduire et de simplifier votre code dans de nombreuses circonstances.

Pagination synchrone ou asynchrone

Les paginateurs fournissent des mécanismes de pagination synchrones et asynchrones. La pagination synchrone est disponible dans les projets .NET Framework 4.7.2 (ou version ultérieure). La pagination asynchrone est disponible dans les projets .NET Core (.NET Core 3.1, .NET 5, etc.).

Les opérations asynchrones et .NET Core étant recommandées, l'exemple suivant montre la pagination asynchrone. Des informations sur la façon d'effectuer les mêmes tâches à l'aide de la pagination synchrone et de .NET Framework 4.7.2 (ou version ultérieure) sont présentées après l'exemple dans. Considérations supplémentaires pour les paginateurs

exemple

L'exemple suivant montre comment utiliser le pour AWS SDK pour .NET afficher une liste de groupes de journaux. Pour le contraste, l'exemple montre comment procéder avec et sans paginateurs. Avant d'examiner le code complet, présenté plus loin, considérez les extraits suivants.

Obtenir des groupes de CloudWatch logs sans paginateurs

```
      // 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));
```

Obtenir des groupes de CloudWatch journaux à l'aide de paginateurs

```
      // 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);
}
```

Les résultats de ces deux extraits étant exactement les mêmes, l'avantage de l'utilisation de paginateurs est clairement visible.

> **Note**
>
> Avant d'essayer de créer et d'exécuter le code complet, assurez-vous d'avoir configuré votre environnement et votre projet.
> Vous pourriez également avoir besoin du Microsoft.Bcl. AsyncInterfaces NuGet package car les paginateurs asynchrones utilisent l'interface. `IAsyncEnumerable`

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.CloudWatch

Eléments de programmation :

- Espace de noms Amazon. CloudWatch

  Classe AmazonCloudWatchLogsClient
- Espace de noms Amazon. CloudWatchLogs.Modèle

  Classe DescribeLogGroupsRequest

  Classe DescribeLogGroupsResponse

  Classe LogGroup

Code complet

```
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}");
```

```
        }
      }
    }
  }
}
```

Considérations supplémentaires pour les paginateurs

- Les paginateurs ne peuvent pas être utilisés plus d'une fois

  Si vous avez besoin des résultats d'un AWS paginateur spécifique à plusieurs endroits de votre code, vous ne devez pas utiliser un objet de pagination plusieurs fois. Créez plutôt un nouveau paginateur chaque fois que vous en avez besoin. Ce concept est illustré dans l'exemple de code précédent de la `DisplayLogGroupsWithPaginators` méthode.

- Pagination synchrone

  La pagination synchrone est disponible pour les projets .NET Framework 4.7.2 (ou version ultérieure).

  Pour ce faire, créez un projet .NET Framework 4.7.2 (ou version ultérieure) et copiez-y le code précédent. Il suffit ensuite de supprimer le `await` mot clé des deux appels de `foreach` pagination, comme indiqué dans l'exemple suivant.

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

  Créez et exécutez le projet pour obtenir les mêmes résultats que ceux obtenus avec la pagination asynchrone.

Support du protocole HTTP 2 dans AWS SDK pour .NET

Certains AWS services et opérations nécessitent le protocole HTTP 2. Par exemple, le streaming bidirectionnel dans Amazon Transcribe Streaming n'est pas possible via HTTP 1.1 et nécessite donc le protocole HTTP 2 à la place. La version 4 de la prise en charge du protocole HTTP 2 AWS SDK pour .NET a été ajoutée afin que vous puissiez utiliser ces opérations dans vos applications. Pour

une opération HTTP 2 bidirectionnelle, le comportement du SDK lors de la réception de flux est similaire à celui du protocole HTTP 1.1. En d'autres termes, lorsque les applications qui utilisent le SDK envoient des événements au service, un éditeur est attribué par le développeur à la demande.

Pour voir ce comportement en action, considérez l'exemple suivant pour Amazon Transcribe Streaming. Il utilise l'Amazon. TranscribeStreaminget Amazon. TranscribeStreaming.Espaces de noms du modèle.

Dans cet exemple, le développeur définit la `StartStreamTranscriptionRequest.AudioStreamPublisher` propriété avec une fonction de rappel, qui est un .NET`Func`. Le SDK utilise le for `Func` défini `AudioStreamPublisher` pour extraire les événements du code de l'utilisateur afin de les diffuser à l'utilisateur. Le SDK appelle le `Func` jusqu'à ce qu'il renvoie null.

Le code montre comment le son d'un fichier peut être diffusé dans Amazon Transcribe Streaming pour y être traité.

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

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

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

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

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

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

    return audioEvent;
};

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

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

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

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

_ = response.TranscriptResultStream.StartProcessingAsync();

try
```

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

> **Warning**
>
> Certaines opérations HTTP 2 bidirectionnelles, telles que la `InvokeModelWithBidirectionalStreamAsync` méthode d'Amazon Bedrock, Amazon.BedrockRuntimeespace de noms, ne renvoyez pas de réponse depuis l'appel initial sur le client de service tant que certains événements n'ont pas été publiés. Ce comportement peut entraîner le blocage de votre application. Pour éviter cette situation, séparez le code de l'application qui fournit les événements à l'éditeur et exécutez-le sur un thread différent du thread qui appelle les opérations sur le client de service.

Considérations supplémentaires

- AWS SDK pour .NET la prise en charge de HTTP 2 n'est disponible que dans les versions qui ciblent .NET 8 et versions ultérieures. Il n'est pas disponible dans les versions qui ciblent .NET Framework.
- Pour des informations plus détaillées, voir PR 3730 dans le aws-sdk-net GitHubréférentiel.

Outils supplémentaires

Voici quelques outils supplémentaires que vous pouvez utiliser pour faciliter le développement, le déploiement et la maintenance de vos applications .NET.

AWS Outil de déploiement

Après avoir développé votre application .NET Core native pour le cloud sur une machine de développement, vous pouvez utiliser l'outil de AWS déploiement pour la CLI .NET afin de déployer plus facilement votre application sur AWS.

Pour de plus amples informations, veuillez consulter Déployez des applications pour AWS.

AWS Framework de traitement des messages pour .NET

Si vous utilisez des services tels qu'Amazon SQS, Amazon SNS ou EventBridge Amazon, vous pouvez peut-être tirer parti de l'infrastructure de traitement AWS des messages pour .NET. Pour de plus amples informations, veuillez consulter AWS Framework de traitement des messages pour .NET.

Intégrations avec .NET Aspire

Vous pouvez tirer parti des intégrations avec .NET Aspire pour améliorer la boucle de développement interne. Pour de plus amples informations, veuillez consulter Intégration AWS à .NET Aspire dans le AWS SDK pour .NET.

Authentification et autorisation avancées avec le AWS SDK pour .NET

Les rubriques de cette section fournissent des informations sur les techniques avancées d'authentification et d'autorisation dans votre AWS SDK pour .NET application.

Rubriques

- Authentification unique avec le AWS SDK pour .NET

Authentification unique avec le AWS SDK pour .NET

AWS IAM Identity Center est un service d'authentification unique (SSO) basé sur le cloud qui facilite la gestion centralisée de l'accès SSO à toutes vos applications Comptes AWS et à celles du cloud. Pour plus de détails, consultez le guide de l'utilisateur d'IAM Identity Center.

Si vous ne savez pas comment un SDK interagit avec IAM Identity Center, consultez les informations suivantes.

Schéma d'interaction de haut niveau

À un niveau élevé, SDKs interagissez avec IAM Identity Center de la même manière que le modèle suivant :

1. IAM Identity Center est configuré, généralement via la console IAM Identity Center, et un utilisateur SSO est invité à participer.
2. Le AWS `config` fichier partagé sur l'ordinateur de l'utilisateur est mis à jour avec les informations SSO.
3. L'utilisateur se connecte via IAM Identity Center et reçoit des informations d'identification à court terme pour les autorisations Gestion des identités et des accès AWS (IAM) qui ont été configurées pour lui. Cette connexion peut être initiée via un outil autre que le SDK tel que le AWS CLI, ou par programmation via une application .NET.
4. L'utilisateur poursuit son travail. Lorsqu'ils exécutent d'autres applications utilisant l'authentification unique, ils n'ont pas besoin de se reconnecter pour ouvrir les applications.

Le reste de cette rubrique fournit des informations de référence pour la configuration et l'utilisation AWS IAM Identity Center. Il fournit des informations supplémentaires et plus avancées que la

configuration SSO de base dans. Authentification avec AWS Si vous débutez dans l'utilisation de l'authentification AWS unique, vous devriez d'abord consulter cette rubrique pour obtenir des informations de base, puis suivre les didacticiels suivants pour voir l'authentification unique en action :

- Tutoriel : application .NET uniquement
- Tutoriel : AWS CLI et application .NET

Cette rubrique contient les sections suivantes :

- Prérequis
- Configuration d'un profil SSO
- Génération et utilisation de jetons SSO
- Ressources supplémentaires
- Didacticiels

Prérequis

Avant d'utiliser IAM Identity Center, vous devez effectuer certaines tâches, telles que le choix d'une source d'identité et la configuration des applications Comptes AWS et applications pertinentes. Pour plus d'informations, consultez les éléments suivants :

- Pour obtenir des informations générales sur ces tâches, consultez la section Getting Started dans le guide de l'utilisateur d'IAM Identity Center.
- Pour des exemples de tâches spécifiques, consultez la liste des didacticiels à la fin de cette rubrique. Assurez-vous toutefois de consulter les informations contenues dans cette rubrique avant d'essayer les didacticiels.

Configuration d'un profil SSO

Une fois le centre d'identité IAM configuré dans le cas correspondant Compte AWS, un profil nommé pour l'authentification unique doit être ajouté au fichier partagé AWS `config` de l'utilisateur. Ce profil est utilisé pour se connecter au portail AWS d'accès, qui renvoie des informations d'identification à court terme pour les autorisations IAM configurées pour l'utilisateur.

Le `config` fichier partagé est généralement nommé `%USERPROFILE%\.aws\config` sous Windows, Linux et macOS. `~/.aws/config` Vous pouvez utiliser votre éditeur de texte préféré pour ajouter un nouveau profil d'authentification unique. Vous pouvez également utiliser la `aws configure sso` commande. Pour plus d'informations sur cette commande, consultez la section Configuration de la AWS CLI pour utiliser IAM Identity Center dans le guide de l'AWS Command Line Interface utilisateur.

Le nouveau profil est similaire au suivant :

```
[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
```

Les paramètres du nouveau profil sont définis ci-dessous. Les deux premiers paramètres définissent le portail AWS d'accès. Les deux autres paramètres constituent une paire qui, ensemble, définit les autorisations configurées pour un utilisateur. Les quatre réglages sont obligatoires.

`sso_start_url`

URL pointant vers le portail d'AWS accès de l'organisation. Pour trouver cette valeur, ouvrez la console IAM Identity Center, sélectionnez Paramètres et recherchez l'URL du portail.

`sso_region`

Celui Région AWS qui contient l'hôte du portail d'accès. Il s'agit de la région qui a été sélectionnée lorsque vous avez activé IAM Identity Center. Elle peut être différente des régions que vous utilisez pour d'autres tâches.

Pour une liste complète des Régions AWS et de leurs codes, consultez la section Points de terminaison régionaux dans le Référence générale d'Amazon Web Services.

`sso_account_id`

L'ID d'un Compte AWS fichier ajouté via le AWS Organizations service. Pour consulter la liste des comptes disponibles, accédez à la console IAM Identity Center et ouvrez la Comptes AWSpage. L'identifiant de compte que vous choisissez pour ce paramètre correspond à la valeur que vous prévoyez d'attribuer au `sso_role_name` paramètre, qui est indiquée ci-dessous.

`sso_role_name`

Nom d'un ensemble d'autorisations IAM Identity Center. Cet ensemble d'autorisations définit les autorisations accordées à un utilisateur via IAM Identity Center.

La procédure suivante permet de trouver la valeur de ce paramètre.

1. Accédez à la console IAM Identity Center et ouvrez la Comptes AWSpage.
2. Choisissez un compte pour afficher ses informations. Le compte que vous choisirez sera celui qui contient l'utilisateur ou le groupe SSO auquel vous souhaitez accorder des autorisations SSO.
3. Consultez la liste des utilisateurs et des groupes assignés au compte et trouvez l'utilisateur ou le groupe qui vous intéresse. L'ensemble d'autorisations que vous spécifiez dans le `sso_role_name` paramètre est l'un des ensembles associés à cet utilisateur ou à ce groupe.

Lorsque vous attribuez une valeur à ce paramètre, utilisez le nom de l'ensemble d'autorisations, et non l'Amazon Resource Name (ARN).

Les ensembles d'autorisations sont associés à des politiques IAM et à des politiques d'autorisations personnalisées. Pour plus d'informations, consultez la section Ensembles d'autorisations dans le guide de l'utilisateur d'IAM Identity Center.

Génération et utilisation de jetons SSO

Pour utiliser l'authentification unique, un utilisateur doit d'abord générer un jeton temporaire, puis utiliser ce jeton pour accéder aux AWS applications et aux ressources appropriées. Pour les applications .NET, vous pouvez utiliser les méthodes suivantes pour générer et utiliser ces jetons temporaires :

- Créez des applications .NET qui génèrent d'abord un jeton, si nécessaire, puis utilisez-le.
- Générez un jeton avec le, AWS CLI puis utilisez-le dans les applications .NET.

Ces méthodes sont décrites dans les sections suivantes et démontrées dans les didacticiels.

> **Important**
>
> Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

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

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Application .NET uniquement

Cette section explique comment créer une application .NET qui génère un jeton SSO temporaire, si nécessaire, puis utilise ce jeton. Pour un didacticiel complet de ce processus, voirTutoriel pour l'authentification unique utilisant uniquement des applications .NET.

Générer et utiliser un jeton SSO par programmation

En plus d'utiliser le AWS CLI, vous pouvez également générer un jeton SSO par programmation.

Pour ce faire, votre application crée un `AWSCredentials`objet pour le profil SSO, qui charge les informations d'identification temporaires si elles sont disponibles. Votre application doit ensuite convertir l'`AWSCredentials`objet en `SSOAWSCredentials`objet et définir certaines propriétés d'options, notamment une méthode de rappel utilisée pour demander à l'utilisateur de fournir des informations de connexion, si nécessaire.

Cette méthode est illustrée dans l'extrait de code suivant.

Important

Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

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

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

```
static AWSCredentials LoadSsoCredentials()
{
```

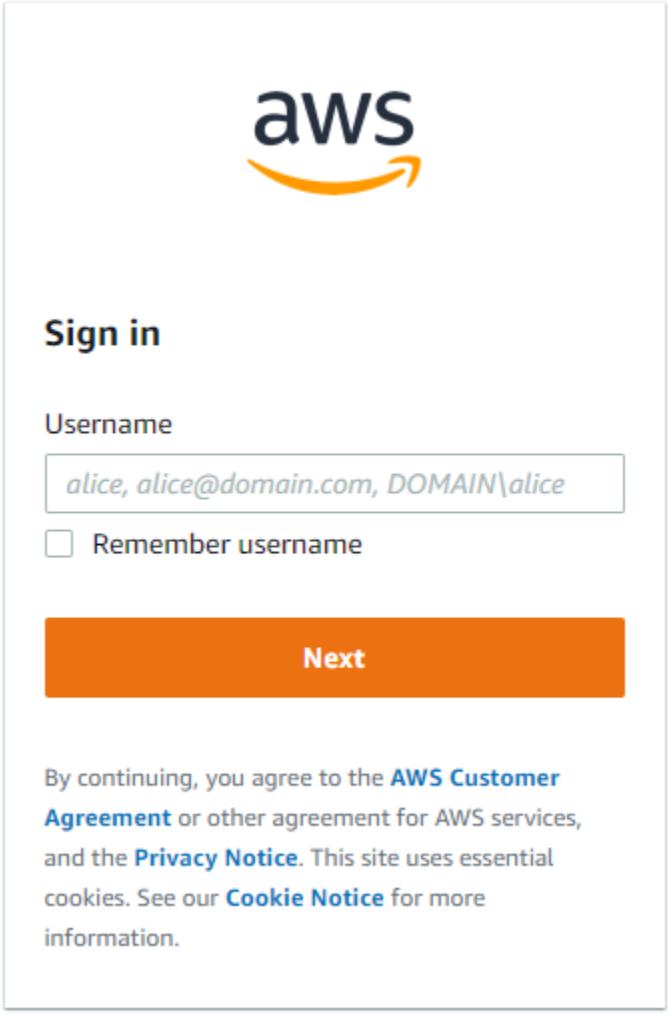
```
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    var ssoCredentials = credentials as SSOAWSCredentials;

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

    return ssoCredentials;
}
```

Si aucun jeton SSO approprié n'est disponible, la fenêtre du navigateur par défaut est ouverte et la page de connexion appropriée est ouverte. Par exemple, si vous utilisez IAM Identity Center comme source d'identité, l'utilisateur voit une page de connexion similaire à la suivante :



> **Note**
>
> La chaîne de texte que vous indiquez ne `SSOAWSCredentials.Options.ClientName` peut pas comporter d'espaces. Si la chaîne contient des espaces, vous obtiendrez une exception d'exécution.

[Tutoriel pour l'authentification unique utilisant uniquement des applications .NET]

AWS CLI et application .NET

Cette section explique comment générer un jeton SSO temporaire à l'aide du AWS CLI, et comment utiliser ce jeton dans une application. Pour un didacticiel complet de ce processus, voir[Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET].

Générez un jeton SSO à l'aide du AWS CLI

En plus de générer un jeton SSO temporaire par programmation, vous utilisez le AWS CLI pour générer le jeton. Les informations suivantes vous montrent comment procéder.

Une fois que l'utilisateur a créé un profil compatible SSO, comme indiqué dans la section précédente, il exécute la `aws sso login` commande à partir du. AWS CLI Ils doivent veiller à inclure le `--profile` paramètre dans le nom du profil compatible SSO. comme illustré dans l'exemple suivant :

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

Si l'utilisateur souhaite générer un nouveau jeton temporaire après l'expiration du jeton actuel, il peut exécuter à nouveau la même commande.

Utiliser le jeton SSO généré dans une application .NET

Les informations suivantes vous montrent comment utiliser un jeton temporaire déjà généré.

> **Important**
>
> Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Votre application crée un `AWSCredentials`objet pour le profil SSO, qui charge les informations d'identification temporaires générées précédemment par le AWS CLI. Cette méthode est similaire aux méthodes présentées dans Accès aux informations d'identification et aux profils dans une application et se présente sous la forme suivante :

```
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`objet est ensuite transmis au constructeur pour un client de service. Par exemple :

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

> **Note**
>
> L'utilisation `AWSCredentials` pour charger des informations d'identification temporaires n'est pas nécessaire si votre application a été conçue pour utiliser le `[default]` profil pour l'authentification unique. Dans ce cas, l'application peut créer des clients de AWS service sans paramètres, comme « `var client = new AmazonS3Client();` ».

Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET

Ressources supplémentaires

Pour obtenir de l'aide supplémentaire, consultez les ressources suivantes.

- Qu'est-ce que IAM Identity Center ?
- Configuration du AWS CLI pour utiliser IAM Identity Center
- Utilisation des informations d'identification IAM Identity Center dans le AWS Toolkit for Visual Studio

Didacticiels

Rubriques

- Tutoriel pour l'authentification unique utilisant uniquement des applications .NET
- Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET

Tutoriel pour l'authentification unique utilisant uniquement des applications .NET

Ce didacticiel explique comment activer l'authentification unique pour une application de base et pour un utilisateur d'authentification unique de test. Il configure l'application pour générer un jeton SSO temporaire par programmation au lieu d'utiliser le. AWS CLI

Ce didacticiel présente une petite partie des fonctionnalités SSO du AWS SDK pour .NET. Pour plus de détails sur l'utilisation d'IAM Identity Center avec le AWS SDK pour .NET, consultez la rubrique contenant des informations générales. Dans cette rubrique, consultez en particulier la description détaillée de ce scénario dans la sous-section intitulée Application .NET uniquement.

> **Note**
>
> Plusieurs étapes de ce didacticiel vous aident à configurer des services tels que AWS Organizations IAM Identity Center. Si vous avez déjà effectué cette configuration, ou si vous êtes uniquement intéressé par le code, vous pouvez passer à la section contenant l'exemple de code.

Conditions préalables

- Configurez votre environnement de développement si ce n'est pas déjà fait. Ceci est décrit dans les sections telles que Installation et configuration de votre chaîne d'outils pour AWS SDK pour .NET et Premiers pas.
- Identifiez ou créez-en au moins un Compte AWS que vous pouvez utiliser pour tester l'authentification unique. Dans le cadre de ce didacticiel, cela s'appelle le compte de test Compte AWS ou simplement le compte de test.
- Identifiez un utilisateur SSO qui peut tester le SSO pour vous. Il s'agit d'une personne qui utilisera le SSO et les applications de base que vous créez. Pour ce didacticiel, cette personne peut être vous (le développeur) ou quelqu'un d'autre. Nous recommandons également une configuration dans laquelle l'utilisateur SSO travaille sur un ordinateur qui ne se trouve pas dans votre environnement de développement. Cependant, cela n'est pas strictement nécessaire.
- L'ordinateur de l'utilisateur SSO doit disposer d'un framework .NET compatible avec celui que vous avez utilisé pour configurer votre environnement de développement.

Configurez AWS

Cette section explique comment configurer différents AWS services pour ce didacticiel.

Pour effectuer cette configuration, connectez-vous d'abord au test en Compte AWS tant qu'administrateur. Ensuite, procédez comme suit :

Amazon S3

Accédez à la console Amazon S3 et ajoutez des compartiments inoffensifs. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces buckets.

AWS IAM

Accédez à la console IAM et ajoutez quelques utilisateurs IAM. Si vous accordez des autorisations aux utilisateurs IAM, limitez les autorisations à quelques autorisations inoffensives en lecture seule. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces utilisateurs IAM.

AWS Organizations

Accédez à la AWS Organizations console et activez Organizations. Pour plus d'informations, consultez Création d'une organisation dans le AWS Organizations Guide de l'utilisateur.

Cette action ajoute le test Compte AWS à l'organisation en tant que compte de gestion. Si vous avez des comptes de test supplémentaires, vous pouvez les inviter à rejoindre l'organisation, mais cela n'est pas nécessaire pour ce didacticiel.

IAM Identity Center

Accédez à la console IAM Identity Center et activez l'authentification unique. Effectuez la vérification des e-mails si nécessaire. Pour plus d'informations, consultez la section Activer le centre d'identité IAM dans le guide de l'utilisateur d'IAM Identity Center.

Effectuez ensuite la configuration suivante.

Configuration du centre d'identité IAM

1. Accédez à la page des paramètres. Recherchez « l'URL du portail d'accès » et enregistrez la valeur pour une utilisation ultérieure dans le `sso_start_url` paramètre.
2. Dans le bandeau du AWS Management Console, recherchez Région AWS celui qui a été défini lorsque vous avez activé l'authentification unique. Il s'agit du menu déroulant situé à gauche de

l' Compte AWS identifiant. Enregistrez le code de région pour une utilisation ultérieure dans le `sso_region` réglage. Ce code sera similaire à`us-east-1`.

3. Créez un utilisateur SSO comme suit :

   a. Accédez à la page Utilisateurs.

   b. Choisissez Ajouter un utilisateur et entrez le nom d'utilisateur, l'adresse e-mail, le prénom et le nom de famille de l'utilisateur. Ensuite, choisissez Suivant.

   c. Choisissez Suivant sur la page des groupes, puis passez en revue les informations et choisissez Ajouter un utilisateur.

4. Créez un groupe comme suit :

   a. Accédez à la page Groupes.

   b. Choisissez Créer un groupe et entrez le nom et la description du groupe.

   c. Dans la section Ajouter des utilisateurs au groupe, sélectionnez l'utilisateur SSO de test que vous avez créé précédemment. Sélectionnez ensuite Créer un groupe.

5. Créez un ensemble d'autorisations comme suit :

   a. Accédez à la page Ensembles d'autorisations et choisissez Créer un ensemble d'autorisations.

   b. Sous Type d'ensemble d'autorisations, sélectionnez Ensemble d'autorisations personnalisé, puis cliquez sur Suivant.

   c. Ouvrez Inline policy et entrez la politique suivante :

      JSON

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

```
    ]
}
```

d. Pour ce didacticiel, entrez le `SSOReadOnlyRole` nom du jeu d'autorisations. Ajoutez une description si vous le souhaitez, puis choisissez Next.

e. Passez en revue les informations, puis choisissez Créer.

f. Enregistrez le nom de l'ensemble d'autorisations pour une utilisation ultérieure dans le `sso_role_name` paramètre.

6. Accédez à la page AWS des comptes et choisissez le AWS compte que vous avez ajouté à l'organisation plus tôt.
7. Dans la section Aperçu de cette page, recherchez l'identifiant du compte et enregistrez-le pour une utilisation ultérieure dans le `sso_account_id` paramètre.
8. Choisissez l'onglet Utilisateurs et groupes, puis choisissez Attribuer des utilisateurs ou des groupes.
9. Sur la page Attribuer des utilisateurs et des groupes, choisissez l'onglet Groupes, sélectionnez le groupe que vous avez créé précédemment, puis cliquez sur Suivant.
10. Sélectionnez l'ensemble d'autorisations que vous avez créé précédemment et choisissez Suivant, puis Soumettre. La configuration prend quelques instants.

Création d'exemples d'applications

Créez les applications suivantes. Ils seront exécutés sur l'ordinateur de l'utilisateur SSO.

Répertorier les compartiments Amazon S3

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSOOIDC` en plus de `AWSSDK.S3` et`AWSSDK.SecurityToken`.

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


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

```
using Amazon.SecurityToken.Model;

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

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

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

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

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

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

            var ssoCredentials = credentials as SSOAWSCredentials;
```

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

            return ssoCredentials;
        }

    }

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

Répertorier les utilisateurs IAM

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSOOIDC` en plus de `AWSSDK.IdentityManagement` et`AWSSDK.SecurityToken`.

```
using System;
```

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


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

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

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

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

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

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

```
        }

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

            var ssoCredentials = credentials as SSOAWSCredentials;

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

            return ssoCredentials;
        }

    }

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

```
        }
    }
}
```

En plus d'afficher des listes de buckets Amazon S3 et d'utilisateurs IAM, ces applications affichent l'ARN d'identité utilisateur pour le profil compatible SSO, décrit dans ce didacticiel. `my-sso-profile`

Ces applications exécutent des tâches de connexion SSO en fournissant une méthode de rappel dans la propriété Options d'un objet. SSOAWSCredentials

Instruire l'utilisateur SSO

Demandez à l'utilisateur SSO de vérifier ses e-mails et d'accepter l'invitation SSO. Ils sont invités à définir un mot de passe. Le message peut prendre quelques minutes pour arriver dans la boîte de réception de l'utilisateur SSO.

Fournissez à l'utilisateur SSO les applications que vous avez créées précédemment.

Demandez ensuite à l'utilisateur SSO d'effectuer les opérations suivantes :

1. Si le dossier contenant le AWS `config` fichier partagé n'existe pas, créez-le. Si le dossier existe et qu'un sous-dossier est appelé `.sso`, supprimez-le.

   L'emplacement de ce dossier est généralement `%USERPROFILE%\.aws` sous Windows, Linux et macOS. `~/.aws`
2. Créez un AWS `config` fichier partagé dans ce dossier, si nécessaire, et ajoutez-y un profil comme suit :

   ```
   [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. Exécutez l'application Amazon S3.
4. Sur la page de connexion Web qui s'affiche, connectez-vous. Utilisez le nom d'utilisateur indiqué dans le message d'invitation et le mot de passe créé en réponse au message.
5. Lorsque la connexion est terminée, l'application affiche la liste des compartiments S3.

6. Exécutez l'application IAM. L'application affiche la liste des utilisateurs IAM. Cela est vrai même si aucune deuxième connexion n'a été effectuée. L'application IAM utilise le jeton temporaire créé précédemment.

Nettoyage

Si vous ne souhaitez pas conserver les ressources que vous avez créées au cours de ce didacticiel, nettoyez-les. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET

Ce didacticiel explique comment activer l'authentification unique pour une application .NET de base et pour un utilisateur d'authentification unique de test. Il utilise le AWS CLI pour générer un jeton SSO temporaire au lieu de le générer par programmation.

Ce didacticiel présente une petite partie des fonctionnalités SSO du AWS SDK pour .NET. Pour plus de détails sur l'utilisation d'IAM Identity Center avec le AWS SDK pour .NET, consultez la rubrique contenant des informations générales. Dans cette rubrique, consultez en particulier la description détaillée de ce scénario dans la sous-section intituléeAWS CLI et application .NET.

> **Note**
>
> Plusieurs étapes de ce didacticiel vous aident à configurer des services tels que AWS Organizations IAM Identity Center. Si vous avez déjà effectué ces configurations, ou si vous êtes uniquement intéressé par le code, vous pouvez passer à la section contenant l'exemple de code.

Conditions préalables

- Configurez votre environnement de développement si ce n'est pas déjà fait. Ceci est décrit dans les sections telles que Installation et configuration de votre chaîne d'outils pour AWS SDK pour .NET etPremiers pas.
- Identifiez ou créez-en au moins un Compte AWS que vous pouvez utiliser pour tester l'authentification unique. Dans le cadre de ce didacticiel, cela s'appelle le compte de test Compte AWS ou simplement le compte de test.

- Identifiez un utilisateur SSO qui peut tester le SSO pour vous. Il s'agit d'une personne qui utilisera le SSO et les applications de base que vous créez. Pour ce didacticiel, cette personne peut être vous (le développeur) ou quelqu'un d'autre. Nous recommandons également une configuration dans laquelle l'utilisateur SSO travaille sur un ordinateur qui ne se trouve pas dans votre environnement de développement. Cependant, cela n'est pas strictement nécessaire.
- L'ordinateur de l'utilisateur SSO doit disposer d'un framework .NET compatible avec celui que vous avez utilisé pour configurer votre environnement de développement.

- Assurez-vous que la AWS CLI version 2 est installée sur l'ordinateur de l'utilisateur SSO. Vous pouvez vérifier cela `aws --version` en exécutant une invite de commande ou un terminal.

Configurez AWS

Cette section explique comment configurer différents AWS services pour ce didacticiel.

Pour effectuer cette configuration, connectez-vous d'abord au test en Compte AWS tant qu'administrateur. Ensuite, procédez comme suit :

Amazon S3

Accédez à la console Amazon S3 et ajoutez des compartiments inoffensifs. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces buckets.

AWS IAM

Accédez à la console IAM et ajoutez quelques utilisateurs IAM. Si vous accordez des autorisations aux utilisateurs IAM, limitez les autorisations à quelques autorisations inoffensives en lecture seule. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces utilisateurs IAM.

AWS Organizations

Accédez à la AWS Organizations console et activez Organizations. Pour plus d’informations, consultez Création d’une organisation dans le AWS Organizations Guide de l’utilisateur.

Cette action ajoute le test Compte AWS à l'organisation en tant que compte de gestion. Si vous avez des comptes de test supplémentaires, vous pouvez les inviter à rejoindre l'organisation, mais cela n'est pas nécessaire pour ce didacticiel.

IAM Identity Center

Accédez à la console IAM Identity Center et activez l'authentification unique. Effectuez la vérification des e-mails si nécessaire. Pour plus d'informations, consultez la section Activer le centre d'identité IAM dans le guide de l'utilisateur d'IAM Identity Center.

Effectuez ensuite la configuration suivante.

Configuration du centre d'identité IAM

1. Accédez à la page des paramètres. Recherchez « l'URL du portail d'accès » et enregistrez la valeur pour une utilisation ultérieure dans le `sso_start_url` paramètre.
2. Dans le bandeau du AWS Management Console, recherchez Région AWS celui qui a été défini lorsque vous avez activé l'authentification unique. Il s'agit du menu déroulant situé à gauche de l' Compte AWS identifiant. Enregistrez le code de région pour une utilisation ultérieure dans le `sso_region` réglage. Ce code sera similaire à`us-east-1`.
3. Créez un utilisateur SSO comme suit :
   a. Accédez à la page Utilisateurs.
   b. Choisissez Ajouter un utilisateur et entrez le nom d'utilisateur, l'adresse e-mail, le prénom et le nom de famille de l'utilisateur. Ensuite, choisissez Suivant.
   c. Choisissez Suivant sur la page des groupes, puis passez en revue les informations et choisissez Ajouter un utilisateur.
4. Créez un groupe comme suit :
   a. Accédez à la page Groupes.
   b. Choisissez Créer un groupe et entrez le nom et la description du groupe.
   c. Dans la section Ajouter des utilisateurs au groupe, sélectionnez l'utilisateur SSO de test que vous avez créé précédemment. Sélectionnez ensuite Créer un groupe.
5. Créez un ensemble d'autorisations comme suit :
   a. Accédez à la page Ensembles d'autorisations et choisissez Créer un ensemble d'autorisations.
   b. Sous Type d'ensemble d'autorisations, sélectionnez Ensemble d'autorisations personnalisé, puis cliquez sur Suivant.
   c. Ouvrez Inline policy et entrez la politique suivante :

JSON

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

d. Pour ce didacticiel, entrez le `SSOReadOnlyRole` nom du jeu d'autorisations. Ajoutez une description si vous le souhaitez, puis choisissez Next.

e. Passez en revue les informations, puis choisissez Créer.

f. Enregistrez le nom de l'ensemble d'autorisations pour une utilisation ultérieure dans le `sso_role_name` paramètre.

6. Accédez à la page AWS des comptes et choisissez le AWS compte que vous avez ajouté à l'organisation précédemment.
7. Dans la section Aperçu de cette page, recherchez l'identifiant du compte et enregistrez-le pour une utilisation ultérieure dans le `sso_account_id` paramètre.
8. Choisissez l'onglet Utilisateurs et groupes, puis choisissez Attribuer des utilisateurs ou des groupes.
9. Sur la page Attribuer des utilisateurs et des groupes, choisissez l'onglet Groupes, sélectionnez le groupe que vous avez créé précédemment, puis cliquez sur Suivant.
10. Sélectionnez l'ensemble d'autorisations que vous avez créé précédemment et choisissez Suivant, puis Soumettre. La configuration prend quelques instants.

Création d'exemples d'applications

Créez les applications suivantes. Ils seront exécutés sur l'ordinateur de l'utilisateur SSO.

Répertorier les compartiments Amazon S3

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSOOIDC` en plus de `AWSSDK.S3` et`AWSSDK.SecurityToken`.

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

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

namespace SSOExample.S3.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

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

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

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
```

```
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

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

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

Répertorier les utilisateurs IAM

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSOOIDC` en plus de `AWSSDK.IdentityManagement` et`AWSSDK.SecurityToken`.

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

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

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

namespace SSOExample.IAM.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

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

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

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

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

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

En plus d'afficher des listes de buckets Amazon S3 et d'utilisateurs IAM, ces applications affichent l'ARN d'identité utilisateur pour le profil compatible SSO, décrit dans ce didacticiel. `my-sso-profile`

Instruire l'utilisateur SSO

Demandez à l'utilisateur SSO de vérifier ses e-mails et d'accepter l'invitation SSO. Ils sont invités à définir un mot de passe. Le message peut prendre quelques minutes pour arriver dans la boîte de réception de l'utilisateur SSO.

Fournissez à l'utilisateur SSO les applications que vous avez créées précédemment.

Demandez ensuite à l'utilisateur SSO d'effectuer les opérations suivantes :

1. Si le dossier contenant le AWS `config` fichier partagé n'existe pas, créez-le. Si le dossier existe et qu'un sous-dossier est appelé `.sso`, supprimez-le.

   L'emplacement de ce dossier est généralement `%USERPROFILE%\.aws` sous Windows, Linux et macOS. `~/.aws`

2. Créez un AWS `config` fichier partagé dans ce dossier, si nécessaire, et ajoutez-y un profil comme suit :

```
[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. Exécutez l'application Amazon S3. Une exception d'exécution apparaît.
4. Exécutez la AWS CLI commande suivante :

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

5. Sur la page de connexion Web qui s'affiche, connectez-vous. Utilisez le nom d'utilisateur indiqué dans le message d'invitation et le mot de passe créé en réponse au message.
6. Exécutez à nouveau l'application Amazon S3. L'application affiche désormais la liste des compartiments S3.
7. Exécutez l'application IAM. L'application affiche la liste des utilisateurs IAM. Cela est vrai même si aucune deuxième connexion n'a été effectuée. L'application IAM utilise le jeton temporaire créé précédemment.

Nettoyage

Si vous ne souhaitez pas conserver les ressources que vous avez créées au cours de ce didacticiel, nettoyez-les. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Déployez des applications pour AWS

Après avoir développé votre application ou service .NET Core natif pour le cloud sur une machine de développement, vous devez le déployer AWS sur. Vous pouvez le faire en utilisant le AWS Management Console ou certains services tels que CloudFormation ou AWS Cloud Development Kit (AWS CDK). Vous pouvez également utiliser AWS des outils créés à des fins de déploiement. À l'aide de ces outils, vous pouvez effectuer les opérations suivantes.

Déploiement à partir de la CLI .NET

Vous pouvez utiliser les AWS outils suivants pour la CLI .NET afin de déployer vos applications pour AWS :

- AWS Outil de déploiement pour .NET CLI : prend en charge les déploiements vers AWS App RunnerAmazon Elastic Container Service (Amazon ECS) et. AWS Elastic Beanstalk
- AWS Lambda Outils pour .NET CLI : prend en charge les déploiements de AWS Lambda projets.

Déployez à partir des boîtes à outils de l'IDE

Vous pouvez utiliser des AWS boîtes à outils pour déployer vos applications directement depuis l'IDE de votre choix :

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > La fonctionnalité « Publier sur AWS » de la boîte à outils expose les mêmes fonctionnalités que l'outil de AWS déploiement pour .NET CLI. Pour en savoir plus, consultez la section Publier sur AWS dans le guide de AWS Toolkit for Visual Studio l'utilisateur.

- AWS Toolkit for JetBrains

  Consultez les sections Utilisation d'applications AWS sans serveur et Utilisation avec AWS App Runner.

- AWS Boîte à outils pour VS Code

  Consultez les sections Utilisation d'applications sans serveur et Utilisation AWS App Runner.

- AWS Toolkit pour Azure DevOps

Cas d'utilisation

Les sections suivantes contiennent des scénarios d'utilisation pour certains types d'applications, notamment des informations sur la manière dont vous utiliseriez la CLI .NET pour déployer ces applications.

- Applications ASP.NET Core
- Applications de console .NET
- Applications Blazor WebAssembly
- AWS Lambda projets

Applications ASP.NET Core

L'outil de AWS déploiement pour la CLI .NET vous aide à déployer vos applications ASP.NET et vous guide tout au long du processus de déploiement. Il s'agit d'un outil interactif pour la CLI .NET qui permet de déployer des applications .NET avec un minimum de AWS connaissances.

L'outil de déploiement possède les fonctionnalités suivantes :

- Recommandations de calcul pour votre application : obtenez les recommandations de calcul et découvrez quel AWS calcul convient le mieux à votre application.
- Génération de Dockerfile - L'outil génère un Dockerfile si nécessaire ou utilise un Dockerfile existant.
- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.
- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

L'AWS outil de déploiement prend en charge le déploiement d'applications ASP.NET Core sur les AWS services suivants :

- Utilisation d'Amazon ECS Service AWS Fargate: prend en charge les déploiements d'applications Web sur Amazon Elastic Container Service (Amazon ECS) avec une puissance de calcul gérée par AWS Fargate un moteur de calcul sans serveur.
- AWS App Runner- Prend en charge les déploiements vers un service entièrement géré qui permet aux développeurs de déployer facilement des applications Web conteneurisées à grande échelle. APIs Aucune expérience préalable en infrastructure n'est requise.
- AWS Elastic Beanstalk- Prend en charge les déploiements vers un service qui permet aux développeurs de déployer facilement des applications Web et APIs vers un environnement entièrement géré à grande échelle. Aucune expérience préalable en infrastructure n'est requise.

Pour en savoir plus, consultez la présentation de l'outil. Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez Comment installer pour obtenir les instructions d'installation.

Applications de console .NET

L'outil de AWS déploiement pour la CLI .NET vous aide à déployer vos applications de console .NET sous forme de service ou de tâche planifiée sous forme d'image de conteneur sous Linux et vous guide tout au long du processus de déploiement. Si votre application ne possède pas de Dockerfile, l'outil le génère automatiquement. Dans le cas contraire, un Dockerfile existant est utilisé.

L'outil de déploiement possède les fonctionnalités suivantes :

- Recommandations de calcul pour votre application : obtenez les recommandations de calcul et découvrez quel AWS calcul convient le mieux à votre application.
- Génération de Dockerfile - L'outil génère un Dockerfile si nécessaire ou utilise un Dockerfile existant.
- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également

contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.

- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

L'AWS outil de déploiement prend en charge le déploiement d'applications .NET Console vers les AWS services suivants :

- Utilisation du service Amazon ECS AWS Fargate: prend en charge les déploiements d'applications .NET en tant que service (par exemple, un processeur d'arrière-plan) sur Amazon Elastic Container Service (Amazon ECS) avec une puissance de calcul gérée AWS Fargate par un moteur de calcul sans serveur.
- Utilisation de tâches planifiées Amazon ECS AWS Fargate: prend en charge les déploiements d'applications .NET en tant que tâche planifiée (par exemple, end-of-day processus) sur Amazon ECS avec une puissance de calcul gérée par un moteur de calcul AWS Fargate sans serveur.

Pour en savoir plus, consultez la présentation de l'outil. Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez Comment installer pour obtenir les instructions d'installation.

Applications Blazor WebAssembly

L'outil de AWS déploiement pour la CLI .NET vous aide à héberger votre WebAssembly application Blazor dans Amazon S3, en utilisant Amazon CloudFront pour la diffusion de contenu sur le réseau. Votre application est déployée dans un compartiment S3 pour l'hébergement Web. L'outil crée et configure un compartiment S3, puis télécharge votre application Blazor dans le compartiment.

L'outil de déploiement possède les fonctionnalités suivantes :

- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.

- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

Pour en savoir plus, consultez la présentation de l'outil. Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez Comment installer pour obtenir les instructions d'installation.

AWS Lambda projets

AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Il exécute votre code sur une infrastructure informatique à haute disponibilité et exécute l'ensemble de l'administration des ressources informatiques. Pour plus d'informations sur Lambda, consultez Qu'est-ce que AWS Lambda ? dans le Guide AWS Lambda du développeur.

Vous pouvez déployer des fonctions Lambda à l'aide de l'interface de ligne de commande (CLI) .NET.

Rubriques

- Prérequis
- Commandes Lambda disponibles
- Étapes de déploiement

Prérequis

Avant de commencer à utiliser la CLI .NET pour déployer des fonctions Lambda, vous devez remplir les conditions préalables suivantes :

- Vérifiez que la CLI .NET est installée. Par exemple : `dotnet --version`. Si nécessaire, allez dans https://dotnet.microsoft.com/download pour l'installer.
- Configurez la CLI .NET pour qu'elle fonctionne avec Lambda. Pour une description de la procédure à suivre, consultez la CLI .NET Core dans le guide du AWS Lambda développeur. Dans cette procédure, la commande de déploiement est la suivante :

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

  Si vous ne savez pas comment créer un rôle IAM pour cet exercice, n'incluez pas cette `--function-role role` partie. L'outil vous aidera à créer un nouveau rôle.

Commandes Lambda disponibles

Pour répertorier les commandes Lambda disponibles via la CLI .NET, ouvrez une invite de commande ou un terminal et entrez. `dotnet lambda --help` Le résultat de la commande sera similaire à ce qui suit :

```
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>
```

Le résultat répertorie toutes les commandes actuellement disponibles.

Étapes de déploiement

Les instructions suivantes supposent que vous avez créé un projet AWS Lambda .NET. Aux fins de cette procédure, le projet est nomméDotNetCoreLambdaTest.

1. Ouvrez une invite de commande ou un terminal, puis accédez au dossier contenant votre fichier de projet .NET Lambda.
2. Saisissez `dotnet lambda deploy-function`.
3. Si vous y êtes invité, entrez la AWS région (la région dans laquelle votre fonction Lambda sera déployée).
4. Lorsque vous y êtes invité, entrez le nom de la fonction à déployer, par exempleDotNetCoreLambdaTest. Il peut s'agir du nom d'une fonction qui existe déjà dans votre ordinateur Compte AWS ou d'une fonction qui n'y a pas encore été déployée.
5. Lorsque vous y êtes invité, sélectionnez ou créez le rôle IAM que Lambda assumera lors de l'exécution de la fonction.

Une fois l'opération terminée, le message Nouvelle fonction Lambda créée s'affiche.

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

Si vous déployez une fonction qui existe déjà dans votre compte, la fonction de déploiement demande uniquement la AWS région (si nécessaire). Dans ce cas, la sortie de commande se termine par`Updating code for existing function`.

Une fois votre fonction Lambda déployée, elle est prête à être utilisée. Pour plus d'informations, consultez Exemples d'utilisation de AWS Lambda.

Lambda surveille automatiquement les fonctions Lambda pour vous et fournit des statistiques via Amazon. CloudWatch Pour surveiller et résoudre les problèmes liés à votre fonction Lambda, consultez la section Surveillance et résolution des problèmes liés aux applications Lambda.

AWS Services d'appel depuis le AWS SDK pour .NET

Les sections suivantes contiennent des exemples, des didacticiels, des tâches et des guides qui vous montrent comment utiliser les AWS SDK pour .NET AWS services. Ces exemples et didacticiels s'appuient sur une API AWS SDK pour .NET fournie. Pour connaître les classes et les méthodes disponibles dans l'API, consultez la référence de l'AWS SDK pour .NET API.

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le Création d'une application simple sujet. Il vous présente le SDK.

Vous pouvez trouver d'autres exemples de code dans le référentiel d'exemples de AWS code et dans le référentiel awslabs sur. GitHub

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

- Exemples de code guidés pour AWS SDK pour .NET
- Utilisation AWS Lambda pour le service de calcul
- Bibliothèques et frameworks de haut niveau pour AWS SDK pour .NET
- Support pour d'autres AWS services et configurations

Exemples de code guidés pour AWS SDK pour .NET

Les sections suivantes contiennent des exemples de code et fournissent des conseils pour les exemples. Ils peuvent vous aider à apprendre à utiliser le AWS SDK pour .NET pour utiliser les AWS services. Pour des exemples de code supplémentaires, voirExemples de code.

> **Note**
>
> Des exemples de code spécifiques à la version 3 de AWS SDK pour .NET sont également disponibles. Pour les trouver, consultez les exemples de code avec des instructions et les exemples de code dans le guide du développeur AWS SDK pour .NET (V3). Sachez toutefois que si vous utilisez les exemples de code spécifiques à la version 3 avec la version V4 du SDK (la dernière version), vous devrez peut-être effectuer des ajustements en fonction des informations contenues dans. Migration vers la version 4

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le Création d'une application simple sujet. Il vous présente le SDK.

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

- Accès à CloudFormation l'aide du AWS SDK pour .NET
- Authentification des utilisateurs avec Amazon Cognito
- Utilisation des bases de données NoSQL Amazon DynamoDB
- Utilisation avec Amazon EC2
- Accès Gestion des identités et des accès AWS (IAM) à l'aide du AWS SDK pour .NET
- Utilisation du stockage sur Internet d'Amazon Simple Storage Service
- Envoyer des notifications depuis le cloud à l'aide d'Amazon Simple Notification Service
- Messagerie à l'aide d'Amazon SQS

Accès à CloudFormation l'aide du AWS SDK pour .NET

Les AWS SDK pour .NET supports AWS CloudFormation, qui créent et provisionnent les déploiements AWS d'infrastructures de manière prévisible et répétée.

APIs

Les AWS SDK pour .NET offres APIs pour les CloudFormation clients. Ils vous APIs permettent de travailler avec des CloudFormation fonctionnalités telles que les modèles et les piles. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous les utilisez APIs. Pour voir l'ensemble complet de APIs, consultez la référence de l'AWS SDK pour .NET API (et faites défiler la page jusqu'à « Amazon »). CloudFormation«).

Ils AWS CloudFormation APIs sont fournis par le AWSSDK. CloudFormationcolis.

Conditions préalables

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

Rubriques

- Lister AWS les ressources à l'aide AWS CloudFormation

Lister AWS les ressources à l'aide AWS CloudFormation

Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour répertorier les ressources par CloudFormation piles. L'exemple utilise l'API de bas niveau. L'application ne prend aucun argument, mais rassemble simplement des informations pour toutes les piles accessibles aux informations d'identification de l'utilisateur, puis affiche des informations sur ces piles.

Références du SDK

NuGet colis :

- AWSSDK.CloudFormation

Éléments de programmation :

- Espace de noms Amazon. CloudFormation

  Classe AmazonCloudFormationClient
- Espace de noms Amazon. CloudFormation.Modèle

  Classe ICloudFormationPaginatorFactory. DescribeStacks

  Classe DescribeStackResourcesRequest

  Classe DescribeStackResourcesResponse

  Class Stack

  Classe StackResource

  Tag de 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($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

 Console.WriteLine("\n-----------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
```

```
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n---------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }
```

```
            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

Authentification des utilisateurs avec Amazon Cognito

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

À l'aide d'Amazon Cognito Identity, vous pouvez créer des identités uniques pour vos utilisateurs et les authentifier pour un accès sécurisé à vos AWS ressources telles qu'Amazon S3 ou Amazon DynamoDB. Amazon Cognito Identity prend en charge les fournisseurs d'identité publics tels qu'Amazon, Facebook, Twitter/Digits, Google ou tout autre fournisseur compatible avec OpenID Connect, ainsi que les identités non authentifiées. Amazon Cognito prend également en charge les identités authentifiées par les développeurs, qui vous permettent d'enregistrer et d'authentifier les utilisateurs à l'aide de votre propre processus d'authentification principal, tout en utilisant Amazon Cognito Sync pour synchroniser les données utilisateur et accéder aux ressources. AWS

Pour plus d'informations sur Amazon Cognito, consultez le guide du développeur Amazon Cognito.

Les exemples de code suivants montrent comment utiliser facilement Amazon Cognito Identity. L'Fournisseur d'informations d'identificationexemple montre comment créer et authentifier les

identités des utilisateurs. L'CognitoAuthentication bibliothèque d'extensionsexemple montre comment utiliser la bibliothèque d' CognitoAuthentication extensions pour authentifier les groupes d'utilisateurs Amazon Cognito.

Rubriques

- Fournisseur d'identifiants Amazon Cognito
- Exemples de bibliothèques d' CognitoAuthentication extensions Amazon

Fournisseur d'identifiants Amazon Cognito

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, qui se trouve dans le AWSSDK.CognitoIdentity NuGetpackage, est un objet d'identification qui utilise Amazon Cognito et the AWS Security Token Service (AWS STS) pour récupérer les informations d'identification pour passer AWS des appels.

La première étape de configuration de `CognitoAWSCredentials` consiste à créer un « groupe d'identités ». Un groupe d'identités est un magasin d'informations relatives à l'identité de l'utilisateur qui est propre au compte. Les informations peuvent être récupérées quels que soient les plateformes client, appareils et systèmes d'exploitation. Autrement dit, si un utilisateur commence à utiliser votre application sur un téléphone, puis passe à une tablette, les informations de l'application restent disponibles pour cet utilisateur. Vous pouvez créer un nouveau pool d'identités depuis la console Amazon Cognito. Si vous utilisez la console, celle-ci vous fournira également les autres éléments d'information dont vous avez besoin :

- Votre numéro de compte – Un numéro à 12 chiffres, comme 123456789012, qui est unique pour votre compte.
- L'ARN du rôle non authentifié – Un rôle que les utilisateurs non authentifiés assumeront. Par exemple, ce rôle peut fournir des autorisations d'accès en lecture seule à vos données.
- L'ARN du rôle authentifié – Un rôle que les utilisateurs authentifiés assumeront. Ce rôle peut fournir des autorisations plus étendues à vos données.

Configurer Cognito AWSCredentials

L'exemple de code suivant montre comment procéder à la configuration`CognitoAWSCredentials`, que vous pouvez ensuite utiliser pour appeler Amazon S3 en tant qu'utilisateur non authentifié. Cela vous permet d'effectuer des appels avec seulement un volume minimum de données requises pour authentifier l'utilisateur. Les autorisations utilisateur sont contrôlées par le rôle. Vous pouvez donc configurer l'accès selon vos besoins.

```
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();
}
```

Utiliser AWS en tant qu'utilisateur non authentifié

L'exemple de code suivant montre comment vous pouvez commencer à utiliser en AWS tant qu'utilisateur non authentifié, puis vous authentifier via Facebook et mettre à jour les informations d'identification pour utiliser les informations d'identification Facebook. En utilisant cette approche, vous pouvez accorder différentes capacités aux utilisateurs authentifiés via le rôle authentifié. Par exemple, votre application de téléphone peut autoriser des utilisateurs à afficher un contenu de manière anonyme, mais leur permet de publier s'ils sont connectés avec un ou plusieurs des fournisseurs configurés.

```
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'objet `CognitoAWSCredentials` fournit encore plus de fonctionnalités si vous l'utilisez avec l'objet `AmazonCognitoSyncClient` qui fait partie du kit AWS SDK pour .NET. Si vous utilisez les deux `AmazonCognitoSyncClient` et`CognitoAWSCredentials`, vous n'avez pas à spécifier les `IdentityId` propriétés `IdentityPoolId` et lorsque vous passez des appels avec le`AmazonCognitoSyncClient`. Ces propriétés sont automatiquement renseignées à partir de `CognitoAWSCredentials`. L'exemple de code suivant illustre ceci, ainsi qu'un événement qui vous informe chaque fois que `IdentityId` pour `CognitoAWSCredentials` est modifié. `IdentityId` peut changer dans certains cas, par exemple lors du passage d'un utilisateur non authentifié à un utilisateur authentifié.

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

Exemples de bibliothèques d' CognitoAuthentication extensions Amazon

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

La bibliothèque d' CognitoAuthentication extensions, qui se trouve dans le dossier Amazon.Extensions. CognitoAuthentication NuGet package, simplifie le processus d'authentification des groupes d'utilisateurs Amazon Cognito pour les développeurs .NET Core et Xamarin. La bibliothèque repose sur l'API du fournisseur d'identité Amazon Cognito pour créer et envoyer des appels d'API d'authentification utilisateur.

Utilisation de la bibliothèque d' CognitoAuthentication extensions

Amazon Cognito intègre des `ChallengeName` valeurs `AuthFlow` et des valeurs pour un flux d'authentification standard permettant de valider le nom d'utilisateur et le mot de passe via le Secure Remote Password (SRP). Pour plus d'informations sur le flux d'authentification, consultez Flux d'authentification du groupe d'utilisateurs Amazon Cognito.

Les exemples suivant nécessitent ces instructions `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;
```

Utiliser l'authentification de base

Créez un AmazonCognitoIdentityProviderClienten utilisant Anonymous AWSCredentials, qui ne nécessite pas de demandes signées. Vous n'avez pas besoin de fournir une région ; le code sous-jacent appelle `FallbackRegionFactory.GetRegionEndpoint()` si aucune région n'est indiquée. Créez des objets `CognitoUserPool` et `CognitoUser`. Appelez la méthode

`StartWithSrpAuthAsync` avec une demande `InitiateSrpAuthRequest` qui contient le mot de passe utilisateur.

```
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;

}
```

Authentifiez-vous en relevant les défis

Il est également plus simple de poursuivre le flux d'authentification avec NewPasswordRequired des défis tels que l'authentification multifactorielle (MFA). Les seules exigences sont les CognitoAuthentication objets, le mot de passe de l'utilisateur pour le SRP et les informations nécessaires pour le prochain défi, qui sont acquises après avoir invité l'utilisateur à les saisir. Le code suivant montre un moyen de vérifier le type de défi et d'obtenir les réponses appropriées pour le MFA et les NewPasswordRequired défis pendant le flux d'authentification.

Effectuez une authentification de base comme auparavant, et attendez (`await`) une réponse `AuthFlowResponse`. Lorsque la réponse est reçue, parcourez l'objet `AuthenticationResult` renvoyé. Si le type `ChallengeName` est `NEW_PASSWORD_REQUIRED`, appelez la méthode `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.");
    }

}
```

Utiliser AWS les ressources après l'authentification

Une fois qu'un utilisateur est authentifié à l'aide de la CognitoAuthentication bibliothèque, l'étape suivante consiste à lui permettre d'accéder aux AWS ressources appropriées. Pour ce faire, vous devez créer un pool d'identités via la console Amazon Cognito Federated Identities. En spécifiant le groupe d'utilisateurs Amazon Cognito que vous avez créé en tant que fournisseur, à l'aide de son pool et de son ID client, vous pouvez autoriser les utilisateurs de votre groupe d'utilisateurs Amazon Cognito à accéder aux ressources connectées à votre compte. AWS Vous pouvez également spécifier des rôles différents pour permettre aux utilisateurs non authentifiés et authentifiés d'accéder à des ressources différentes. Vous pouvez modifier ces règles dans la console IAM, où vous pouvez ajouter ou supprimer des autorisations dans le champ Action de la stratégie attachée du rôle. Ensuite, en utilisant le groupe d'identités, le groupe d'utilisateurs et les informations utilisateur Amazon Cognito appropriés, vous pouvez passer des appels vers différentes AWS ressources. L'exemple suivant montre un utilisateur authentifié par SRP accédant aux différents compartiments Amazon S3 autorisés par le rôle du pool d'identités associé.

```
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);
        }
    }
}
```

Options d'authentification supplémentaires

Outre le SRP et le MFA NewPasswordRequired, CognitoAuthentication la bibliothèque d'extensions facilite le flux d'authentification pour :

- Personnalisé - Initier avec un appel à `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Commencez par un appel à `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Initiez par un appel à `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Initiez par un appel à `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Appelez la méthode appropriée selon le flux souhaité. Continuez ensuite à interroger l'utilisateur avec des stimulations comme elles sont présentées dans les objets `AuthFlowResponse` de chaque appel de méthode. Appelez également la méthode de réponse appropriée, comme

`RespondToSmsMfaAuthAsync` pour les stimulations MFA et `RespondToCustomAuthAsync` pour les stimulations personnalisées.

Utilisation des bases de données NoSQL Amazon DynamoDB

> **Note**
>
> Les modèles de programmation décrits dans ces rubriques sont présents à la fois dans .NET Framework et .NET (Core), mais les conventions d'appel diffèrent, qu'elles soient synchrones ou asynchrones.

Il AWS SDK pour .NET prend en charge Amazon DynamoDB, un service de base de données NoSQL rapide proposé par. AWSLe SDK fournit trois modèles de programmation pour communiquer avec DynamoDB : le modèle de bas niveau, le modèle de document et le modèle de persistance des objets.

Les informations suivantes présentent ces modèles et les leurs APIs, fournissent des exemples expliquant comment et quand les utiliser, et vous proposent des liens vers des ressources de programmation DynamoDB supplémentaires dans le. AWS SDK pour .NET

Modèle de bas niveau

Le modèle de programmation de bas niveau englobe les appels directs au service DynamoDB. Vous pouvez accéder à ce modèle via l'espace de noms Amazon.Dynamo. DBv2

Parmi les trois modèles, le modèle de bas niveau est celui qui nécessite le plus de programmation. Par exemple, vous devez convertir les types de données .NET en leurs équivalents dans DynamoDB. Toutefois, ce modèle donne accès à la plupart des fonctions.

Les exemples suivants montrent comment utiliser le modèle de bas niveau pour créer une table, modifier une table et insérer des éléments dans une table dans DynamoDB.

Création d'une table

L'exemple suivant utilise la méthode `CreateTable` de la classe `AmazonDynamoDBClient` pour créer une table. La méthode `CreateTable` utilise une instance de la classe `CreateTableRequest`, qui contient des caractéristiques telles que des noms d'attribut d'élément requis, une définition de clé primaire et une capacité de débit. La méthode `CreateTable` renvoie une instance de la 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);
}
```

Vérification qu'une table est prête pour modification

Pour pouvoir remplacer ou modifier une table, celle-ci doit être prête pour cette opération. L'exemple suivant montre comment utiliser le modèle de bas niveau pour vérifier qu'une table est prête dans DynamoDB. Dans cet exemple, la table cible à vérifier est référencée par l'intermédiaire de la méthode `DescribeTable` de la classe `AmazonDynamoDBClient`. Toutes les cinq secondes, le code vérifie la valeur de la propriété `TableStatus` de la table. Lorsque le statut est défini sur `ACTIVE`, la table est prête à être modifiée.

```
// 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);
```

Insertion d'un élément dans une table

Dans l'exemple suivant, vous utilisez le modèle de bas niveau pour insérer deux éléments dans une table de DynamoDB. Chaque élément est inséré par l'intermédiaire de la méthode `PutItem` de la classe `AmazonDynamoDBClient`, en utilisant une instance de la classe `PutItemRequest`. Chacune des deux instances de la classe `PutItemRequest` prend le nom de la table dans laquelle les éléments seront insérés, avec une série de valeurs d'attribut d'élément.

```
// 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);
```

Modèle de document

Le modèle de programmation de documents permet de travailler plus facilement avec les données dans DynamoDB. Ce modèle est particulièrement conçu pour l'accès aux tables et aux éléments des tables. Vous pouvez accéder à ce modèle par le biais d'Amazon.Dynamo. DBv2 DocumentModelespace de noms.

Comparé au modèle de programmation de bas niveau, le modèle de document est plus facile à coder à partir des données DynamoDB. Par exemple, il n'est pas nécessaire de convertir autant de types de données .NET en leurs équivalents dans DynamoDB. Toutefois, ce modèle ne donne pas accès à autant de fonctions que le modèle de programmation de bas niveau. Par exemple, vous pouvez utiliser ce modèle pour créer, extraire, mettre à jour et supprimer des éléments dans des tables. Toutefois, pour créer les tables, vous devez utiliser le modèle de bas niveau. Comparé au modèle de persistance des objets, ce modèle nécessite davantage de programmation pour stocker, charger et interroger des objets .NET.

Pour plus d'informations sur le modèle de programmation de documents DynamoDB, consultez .NET : modèle de document dans le manuel Amazon DynamoDB Developer Guide.

Les sections suivantes fournissent des informations sur la création d'une représentation de la table DynamoDB souhaitée, ainsi que des exemples sur la manière d'utiliser le modèle de document pour insérer des éléments dans des tableaux et obtenir des éléments à partir de tableaux.

Création d'une représentation de la table

Pour effectuer des opérations sur les données à l'aide du modèle de document, vous devez d'abord créer une instance de la `Table` classe qui représente une table spécifique. Il existe deux méthodes principales pour ce faire.

LoadTable méthode

Le premier mécanisme consiste à utiliser l'une des `LoadTable` méthodes statiques de la `Table`classe, comme dans l'exemple suivant :

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

> **Note**
>
> Bien que ce mécanisme fonctionne, dans certaines conditions, il peut parfois entraîner une latence supplémentaire ou des blocages en raison des comportements de démarrage à froid et de pool de threads. Pour plus d'informations sur ces comportements, consultez le billet de blog Modèles d'initialisation DynamoDB améliorés pour le. AWS SDK pour .NET

TableBuilder

Un mécanisme alternatif, la `TableBuilder`classe, a été introduit dans la version 3.7.203 du package AWSSDK DBv2 NuGet .Dynamo. Ce mécanisme peut remédier aux comportements mentionnés ci-dessus en supprimant certains appels de méthode implicites, en particulier la `DescribeTable` méthode. Ce mécanisme est utilisé de la même manière que dans l'exemple suivant :

```
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();
```

Pour plus d'informations sur ce mécanisme alternatif, consultez à nouveau le billet de blog Modèles d'initialisation DynamoDB améliorés pour le. AWS SDK pour .NET

Insertion d'un élément dans un tableau

Dans l'exemple suivant, une réponse est insérée dans la table Reply via la `PutItemAsync` méthode de la `Table` classe. La méthode `PutItemAsync` prend une instance de la classe `Document` ; la classe `Document` est simplement une collection d'attributs initialisés.

```
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);
```

Obtenir un élément d'une table

Dans l'exemple suivant, une réponse est récupérée via la `GetItemAsync` méthode de la `Table` classe. Pour déterminer la réponse à obtenir, la `GetItemAsync` méthode utilise la clé hash-and-range primaire de la réponse cible.

```
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'exemple précédent convertit implicitement les valeurs de table en chaînes pour la `WriteLine` méthode. Vous pouvez effectuer des conversions explicites en utilisant les différentes méthodes « As [type] » de la `DynamoDBEntry` classe. Par exemple, vous pouvez convertir explicitement la valeur d'un type `Id` de `Primitive` données en GUID par le biais de la `AsGuid()` méthode suivante :

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

Modèle de persistance des objets

Le modèle de programmation de persistance des objets est spécialement conçu pour stocker, charger et interroger des objets .NET dans DynamoDB. Vous pouvez accéder à ce modèle par le biais d'Amazon.Dynamo. DBv2 DataModelespace de noms.

Parmi les trois modèles, le modèle de persistance des objets est le plus facile à utiliser pour coder lorsque vous stockez, chargez ou interrogez des données DynamoDB. Par exemple, vous travaillez directement avec les types de données DynamoDB. Toutefois, ce modèle permet d'accéder uniquement aux opérations qui stockent, chargent et interrogent des objets .NET dans DynamoDB. Par exemple, vous pouvez utiliser ce modèle pour créer, extraire, mettre à jour et supprimer des éléments dans des tables. Toutefois, vous devez tout d'abord créer vos tables à l'aide du modèle de bas niveau, puis utiliser ce modèle pour mapper vos classes .NET avec les tables.

Pour plus d'informations sur le modèle de programmation de persistance des objets DynamoDB, consultez .NET : modèle de persistance des objets dans le manuel Amazon DynamoDB Developer Guide.

Les exemples suivants montrent comment définir une classe .NET qui représente un élément DynamoDB, utiliser une instance de la classe .NET pour insérer un élément dans une table DynamoDB et utiliser une instance de la classe .NET pour obtenir un élément de la table.

Définition d'une classe .NET qui représente un élément d'une table

Dans l'exemple suivant de définition de classe, l'`DynamoDBTable`attribut indique le nom de la table, tandis que les `DynamoDBRangeKey` attributs `DynamoDBHashKey` et modélisent la clé hash-and-range primaire de la table. L'`DynamoDBGlobalSecondaryIndexHashKey`attribut est défini de telle sorte qu'une requête de réponses par un auteur spécifique puisse être créée.

```
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; }
```

```
}
```

Création d'un contexte pour le modèle de persistance des objets

Pour utiliser le modèle de programmation de persistance des objets pour DynamoDB, vous devez créer un contexte qui fournit une connexion à DynamoDB et vous permet d'accéder à des tables, d'effectuer diverses opérations et d'exécuter des requêtes.

Contexte de base

L'exemple suivant montre comment créer le contexte le plus élémentaire.

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

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

Contexte lié à DisableFetchingTableMetadata la propriété

L'exemple suivant montre comment vous pouvez également définir la `DisableFetchingTableMetadata` propriété de la `DynamoDBContextConfig` classe pour empêcher les appels implicites à la `DescribeTable` méthode.

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

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Si la `DisableFetchingTableMetadata` propriété est définie sur `false` (valeur par défaut), comme indiqué dans le premier exemple, vous pouvez omettre les attributs décrivant la structure de clé et d'index des éléments de table dans la `Reply` classe. Ces attributs seront plutôt déduits par un appel implicite à la `DescribeTable` méthode. `DisableFetchingTableMetadata`Il est défini sur`true`, comme indiqué dans le deuxième exemple, sur les méthodes du modèle de persistance des objets telles que `SaveAsync` et `QueryAsync` s'appuyant entièrement sur les attributs définis dans la `Reply` classe. Dans ce cas, aucun appel à la `DescribeTable` méthode n'a lieu.

> **Note**
>
> Dans certaines conditions, les appels à la `DescribeTable` méthode peuvent parfois entraîner une latence supplémentaire ou des blocages en raison des comportements de démarrage à froid et de pool de threads. Pour cette raison, il est parfois avantageux d'éviter d'avoir recours à cette méthode.
> Pour plus d'informations sur ces comportements, consultez le billet de blog Modèles d'initialisation DynamoDB améliorés pour le. AWS SDK pour .NET

Utilisation d'une instance de la classe .NET pour insérer un élément dans un tableau

Dans cet exemple, un élément est inséré par le biais de la `SaveAsync` méthode de la `DynamoDBContext` classe, qui prend une instance initialisée de la classe .NET qui représente l'élément.

```
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"
});
```

Utilisation d'une instance d'une classe .NET pour obtenir des éléments d'une table

Dans cet exemple, une requête est créée pour trouver tous les enregistrements de « Author1" en utilisant la `QueryAsync` méthode de la `DynamoDBContext` classe. Ensuite, les éléments sont récupérés via la `GetNextSetAsync` méthode de la requête.

```
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);
}
```

Informations supplémentaires sur le modèle de persistance des objets

Les exemples et explications présentés ci-dessus incluent parfois une propriété de la `DynamoDBContext` classe appelée`DisableFetchingTableMetadata`. Cette propriété, qui a été introduite dans la version 3.7.203 du DBv2 NuGet package AWSSDK .Dynamo, vous permet d'éviter certaines conditions susceptibles d'entraîner une latence ou des blocages supplémentaires en raison des comportements de démarrage à froid et de pool de threads. Pour plus d'informations, consultez le billet de blog Modèles d'initialisation DynamoDB améliorés pour le. AWS SDK pour .NET

Vous trouverez ci-dessous des informations supplémentaires sur cette propriété.

- Cette propriété peut être définie globalement dans votre `web.config` fichier `app.config` or si vous utilisez .NET Framework.

- Cette propriété peut être définie globalement à l'aide de la `AWSConfigsDynamoDB`classe, comme illustré dans l'exemple suivant.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

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

- Dans certains cas, vous ne pouvez pas ajouter d'attributs DynamoDB à une classe .NET, par exemple si la classe est définie dans une dépendance. Dans de tels cas, il est toujours possible de profiter de la `DisableFetchingTableMetadata` propriété. Pour ce faire, utilisez la `TableBuilder`classe en plus de la `DisableFetchingTableMetadata` propriété. La `TableBuilder` classe a également été introduite dans la version 3.7.203 du package AWSSDK .Dynamo. DBv2 NuGet

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

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

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();

// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
    IndexName = "PostedBy-Message-index"
});
```

En savoir plus

Utilisation du AWS SDK pour .NET pour programmer DynamoDB, informations et exemples

- DynamoDB APIs
- Lancement de DynamoDB Series
- DynamoDB Series - Modèle de document
- DynamoDB Series - Schémas de conversion
- DynamoDB Series - Modèle de persistance des objets
- DynamoDB Series - Expressions
- Utilisation d'expressions avec Amazon DynamoDB et AWS SDK pour .NET
- Support JSON dans Amazon DynamoDB

Modèle de bas niveau, informations et exemples

- Utilisation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau
- Utilisation d'éléments à l'aide de l'API de AWS SDK pour .NET bas niveau
- Interrogation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau
- Numérisation de tables à l'aide de l' AWS SDK pour .NET API de bas niveau
- Utilisation d'index secondaires locaux à l'aide de l'API de AWS SDK pour .NET bas niveau
- Utilisation d'index secondaires globaux à l'aide de l'API de AWS SDK pour .NET bas niveau

Modèle de document, informations et exemples

- Types de données DynamoDB
- Dynamo DBEntry
- .NET : Modèle de document

Modèle de persistance des objets, informations et exemples

- .NET : modèle de persistance des objets

Autres informations utiles

- Consultez Intégration AWS à .NET Aspire pour plus d'informations sur le développement avec Amazon DynamoDB local via .NET Aspire.

Rubriques

- Utilisation d'expressions avec Amazon DynamoDB et AWS SDK pour .NET
- Support JSON dans Amazon DynamoDB

Utilisation d'expressions avec Amazon DynamoDB et AWS SDK pour .NET

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Les exemples de code suivants montrent comment utiliser le AWS SDK pour .NET pour programmer DynamoDB avec des expressions. Les expressions indiquent les attributs que vous souhaitez lire à partir d'un élément d'une table DynamoDB. Vous utilisez également des expressions lors de l'écriture d'un élément, pour signaler toutes les conditions qui doivent être satisfaites (aussi appelées mise à jour conditionnelle) et pour indiquer comment les attributs doivent être mis à jour. Des exemples de mise à jour remplacent l'attribut par une nouvelle valeur, ou ajoutent de nouvelles données à une liste ou une carte. Pour plus d'informations, consultez Lecture et écriture d'éléments à l'aide d'expressions.

Rubriques

- Exemples de données
- Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions
- Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions
- Obtenir plusieurs éléments par l'intermédiaire d'expressions et d'autres attributs d'élément
- Imprimer un élément
- Créer ou remplacer un élément à l'aide d'expressions
- Mettre à jour un élément à l'aide d'expressions
- Supprimer un élément à l'aide d'expressions
- Plus d'informations

Exemples de données

Les exemples de code présentés dans cette rubrique s'appuient sur les deux exemples d'éléments suivants d'une table DynamoDB nommée. `ProductCatalog` Ces éléments décrivent des informations sur les entrées de produit dans un catalogue de magasin de cycles fictif. Ces éléments sont basés sur l'exemple fourni dans Case Study : A ProductCatalog Item. Les descripteurs de types de données tels que `BOOL`, `L`, `M`, `N`, `NS`, `S` et `SS` correspondent à ceux de Format de données 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."
        ]
      }
```

```
    }
  }
}
```

Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` et un jeu d'expressions pour obtenir, puis imprimer, l'élément pour lequel `Id` est égal à `205`. Seuls les attributs suivants de l'élément sont renvoyés : `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` et `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);
```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#ri` pour représenter l'attribut `RelatedItems`. L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans Imprimer un élément.

Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions

Les exemples suivants présentent la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` et un jeu d'expressions pour obtenir, puis imprimer, l'élément pour lequel `Id` est égal à `301`, mais uniquement si la valeur de `Price` est supérieure à `150`. Seuls les attributs suivants de l'élément sont renvoyés : `Id`, `Title` et l'ensemble des attributs `ThreeStar` dans `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("=====");
}
```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#p` pour représenter l'attribut `Price`. `#pr.ThreeStar` spécifie de retourner uniquement l'attribut `ThreeStar`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:val` pour représenter la valeur `150`. La propriété `FilterExpression` spécifie que `#p` (`Price`) doit être supérieur à `:val` (`150`). L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans Imprimer un élément.

Obtenir plusieurs éléments par l'intermédiaire d'expressions et d'autres attributs d'élément

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` et un jeu d'expressions pour obtenir, puis imprimer, l'ensemble des éléments pour lesquels `ProductCategory` est égal à `Bike`. Seuls les attributs suivants de l'élément sont renvoyés : `Id`, `Title` et l'ensemble des attributs dans `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("=====");
}
```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#pc` pour représenter l'attribut `ProductCategory`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:catg` pour représenter la valeur `Bike`. La propriété `FilterExpression` spécifie que `#pc` (`ProductCategory`) doit être égal à `:catg` (`Bike`). L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans Imprimer un élément.

Imprimer un élément

L'exemple suivant présente comment imprimer les attributs et les valeurs d'un élément. Cet exemple est utilisé dans les exemples précédents qui montrent comment Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions, Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions et Obtention de plusieurs éléments à l'aide d'expressions et d'autres attributs d'élément.

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
  foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
  {
    Console.Write(kvp.Key + " = ");
    PrintValue(kvp.Value);
  }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
  // Binary attribute value.
  if (value.B != null)
  {
```

```
    Console.Write("Binary data");
  }
  // Binary set attribute value.
  else if (value.BS.Count > 0)
  {
    foreach (var bValue in value.BS)
    {
      Console.Write("\n  Binary data");
    }
  }
  // List attribute value.
  else if (value.L.Count > 0)
  {
    foreach (AttributeValue attr in value.L)
    {
      PrintValue(attr);
    }
  }
  // Map attribute value.
  else if (value.M.Count > 0)
  {
    Console.Write("\n");
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
```

```
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
  }

  Console.Write("\n");
}
```

Dans l'exemple précédent, chaque valeur d'attribut possède plusieurs data-type-specific propriétés qui peuvent être évaluées afin de déterminer le format correct pour imprimer l'attribut. Ces propriétés incluent `B`, `BOOL`, `BS`, `L`, `M`, `N`, `NS`, `NULL`, `S` et `SS`, qui correspondent à celles du Format de données JSON. Pour des propriétés telles que `B`, `N`, `NULL` et `S`, si la propriété correspondante n'est pas `null`, l'attribut est du type de données non `null` correspondant. Pour les propriétés telles que `BSL`,`M`,`NS`, et`SS`, si la valeur `Count` est supérieure à zéro, l'attribut est du type de non-zero-value données correspondant. Si toutes les data-type-specific propriétés de l'attribut sont égales à zéro `null` ou sont `Count` égales à zéro, l'attribut correspond au type de `BOOL` données.

Créer ou remplacer un élément à l'aide d'expressions

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` et un jeu d'expressions pour mettre à jour l'élément pour lequel `Title` est `18-Bicycle 301`. Si cet élément n'existe pas déjà, un nouvel élément est ajouté.

```
// 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);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:product` pour représenter la valeur `18-Bicycle 301`. La propriété `ConditionExpression` spécifie que `#title` (`Title`) doit être égal à `:product` (`18-Bicycle 301`). L'appel à `CreateItemData` fait référence à la fonction personnalisée suivante :

```
// 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;
}
```

Dans l'exemple précédent, un exemple d'élément avec des exemples de données est renvoyé au mandataire. Une série d'attributs et de valeurs correspondantes est construite, à l'aide des types de données tels que `BOOL`, `L`, `M`, `N`, `NS`, `S` et `SS`, qui correspondent à ceux du format de données JSON.

Mettre à jour un élément à l'aide d'expressions

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` et un jeu d'expressions pour modifier `Title` en `18" Girl's Bike` pour l'élément pour lequel `Id` est égal à `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);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:newproduct` pour représenter la valeur `18" Girl's Bike`. La propriété `UpdateExpression` spécifie de modifier `#title` (`Title`) en `:newproduct` (`18" Girl's Bike`).

Supprimer un élément à l'aide d'expressions

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` et un jeu d'expressions pour supprimer les éléments pour lesquels `Id` est égal à `301` et `Title` est égal à `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);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:product` pour représenter la valeur `18-Bicycle 301`. La propriété

`ConditionExpression` spécifie que `#title` (`Title`) doit être égal à `:product` (`18-Bicycle 301`).

Plus d'informations

Pour plus d'informations et des exemples de code, consultez :

- DynamoDB Series - Expressions
- Accès à des attributs d'élément avec des expressions de projections
- Utilisation d'espaces réservés pour valeurs et noms d'attributs
- Spécification de conditions avec des expressions de condition
- Modification d'éléments et d'attributs avec des expressions de mises à jour
- Utilisation d'éléments à l'aide de l'API de AWS SDK pour .NET bas niveau
- Interrogation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau
- Numérisation de tables à l'aide de l' AWS SDK pour .NET API de bas niveau
- Utilisation d'index secondaires locaux à l'aide de l'API de AWS SDK pour .NET bas niveau
- Utilisation d'index secondaires globaux à l'aide de l'API de AWS SDK pour .NET bas niveau

Support JSON dans Amazon DynamoDB

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Il AWS SDK pour .NET prend en charge les données JSON lorsque vous travaillez avec Amazon DynamoDB. Cela vous permet d'obtenir plus facilement des données au format JSON à partir de tables DynamoDB et d'y insérer des documents JSON.

Rubriques

- Obtenir des données à partir d'une table DynamoDB au format JSON
- Insérer des données au format JSON dans une table DynamoDB
- Conversions des types de données DynamoDB en JSON
- Plus d'informations

Obtenir des données à partir d'une table DynamoDB au format JSON

L'exemple suivant montre comment obtenir des données d'une table DynamoDB au format JSON :

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
//   {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
//   {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
//   }
```

Dans l'exemple précédent, la méthode `ToJson` de la classe `Document` convertit un élément de la table en chaîne au format JSON. La méthode `GetItem` de la classe `Table` permet de récupérer cet élément. Pour déterminer l'élément à obtenir, dans cet exemple, la `GetItem` méthode utilise la clé hash-and-range primaire de l'élément cible. Pour déterminer la table à partir de laquelle obtenir l'élément, la `LoadTable` méthode de la `Table` classe utilise une instance de la `AmazonDynamoDBClient` classe et le nom de la table cible dans DynamoDB.

Insérer des données au format JSON dans une table DynamoDB

L'exemple suivant montre comment utiliser le format JSON pour insérer un élément dans une table 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);
```

Dans l'exemple précédent, la méthode `FromJson` de la classe `Document` convertit une chaîne au format JSON en élément. L'élément est inséré dans la table par l'intermédiaire de la méthode `PutItem` de la classe `Table`, qui utilise l'instance de la classe `Document` contenant l'élément. Pour déterminer la table dans laquelle insérer l'élément, la `LoadTable` méthode de la `Table` classe est appelée, en spécifiant une instance de la `AmazonDynamoDBClient` classe et le nom de la table cible dans DynamoDB.

Conversions des types de données DynamoDB en JSON

Chaque fois que vous appelez la `ToJson` méthode de la `Document` classe, puis que vous appelez la `FromJson` méthode pour reconvertir les données JSON en instance de `Document` classe, certains types de données DynamoDB ne sont pas convertis comme prévu. En particulier :

- Les ensembles DynamoDB (`SS`les types`NS`, `BS` et) seront convertis en tableaux JSON.
- Les scalaires et ensembles binaires DynamoDB (`B`les types `BS` et) seront convertis en chaînes JSON codées en base64 ou en listes de chaînes.

  Dans ce scénario, il convient d'appeler la méthode `DecodeBase64Attributes` de la classe `Document` pour remplacer les données JSON codées en base64 par la représentation binaire correcte. L'exemple suivant remplace par la représentation binaire correcte un attribut d'élément binary scalar codé en base64 dans une instance d'une classe `Document` nommée `Picture`. Cet exemple procède de même pour un attribut d'élément binary set codé en base64 dans la même instance de la classe `Document` nommée `RelatedPictures` :

  ```
  item.DecodeBase64Attributes("Picture", "RelatedPictures");
  ```

Plus d'informations

Pour plus d'informations et des exemples de programmation de JSON avec DynamoDB avec AWS SDK pour .NET le, voir :

- Support DynamoDB pour JSON

- Mise à jour d'Amazon DynamoDB : JSON, offre gratuite élargie, dimensionnement flexible, éléments plus volumineux

Utilisation avec Amazon EC2

Il AWS SDK pour .NET prend en charge Amazon EC2, un service Web qui fournit une capacité de calcul redimensionnable. Vous utilisez cette capacité de calcul pour créer et héberger vos systèmes logiciels.

APIs

AWS SDK pour .NET Il fournit des API pour les clients Amazon EC2. Les API vous permettent de travailler avec des fonctionnalités EC2 telles que les groupes de sécurité et les paires de clés. Les API vous permettent également de contrôler les instances Amazon EC2. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la référence de l'AWS SDK pour .NET API (et faites défiler la page jusqu'à « Amazon.ec2 »).

Les Amazon EC2 APIs sont fournis par le package AWSSDK NuGet .EC2.

Conditions préalables

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

À propos des exemples

Les exemples présentés dans cette section vous montrent comment travailler avec les clients Amazon EC2 et gérer les instances Amazon EC2.

Le didacticiel sur les instances Spot EC2 explique comment demander des instances Spot Amazon EC2. Les instances Spot vous permettent d'accéder à de la capacité EC2 inutilisée à un prix inférieur au prix à la demande.

Rubriques

- Travailler avec des groupes de sécurité dans Amazon EC2
- Utilisation des paires de clés Amazon EC2
- Afficher vos régions et zones de disponibilité Amazon EC2
- Utilisation des instances Amazon EC2

- Tutoriel sur les instances Spot Amazon EC2

Travailler avec des groupes de sécurité dans Amazon EC2

Dans Amazon EC2, un groupe de sécurité agit comme un pare-feu virtuel qui contrôle le trafic réseau pour une ou plusieurs instances EC2. Par défaut, EC2 associe vos instances à un groupe de sécurité qui n'autorise aucun trafic entrant. Vous pouvez créer un groupe de sécurité qui autorise vos instances EC2 à accepter un certain trafic. Par exemple, si vous devez vous connecter à une instance Windows EC2, vous devez configurer le groupe de sécurité afin d'autoriser le trafic RDP.

Pour en savoir plus sur les groupes de sécurité, consultez les groupes de sécurité Amazon EC2 dans le guide de l'utilisateur Amazon EC2.

> **Warning**
>
> EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog EC2-Classic Networking is Retiring — Here's How to Prepare.

Pour plus d'informations sur le APIs et les prérequis, consultez la section parent (Utilisation avec Amazon EC2).

Rubriques

- Énumération des groupes de sécurité
- Création de groupes de sécurité
- Mise à jour des groupes de sécurité

Énumération des groupes de sécurité

Cet exemple montre comment utiliser le pour AWS SDK pour .NET énumérer les groupes de sécurité. Si vous fournissez un Amazon Virtual Private Cloud ID, l'application énumère les groupes de sécurité pour ce VPC en particulier. Dans le cas contraire, l'application affiche simplement la liste de tous les groupes de sécurité disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Énumérer les groupes de sécurité
- Code complet
- Considérations supplémentaires

Énumérer les groupes de sécurité

L'extrait suivant répertorie vos groupes de sécurité. Il énumère tous les groupes ou les groupes d'un VPC en particulier s'il en existe un.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
//
// 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();
      }
    }
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe DescribeSecurityGroupsRequest

  Classe DescribeSecurityGroupsResponse

  Filtre de classe

  Classe SecurityGroup

Le code

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

namespace EC2EnumerateSecGroups
{
  class Program
  {
    static async Task Main(string[] args)
```

```
    {
      // Parse the command line
       string vpcID = string.Empty;
      if(args.Length == 0)
      {
        Console.WriteLine("\nEC2EnumerateSecGroups [vpc_id]");
        Console.WriteLine("  vpc_id - The ID of the VPC for which you want to see
security groups.");
        Console.WriteLine("\nSince you specified no arguments, showing all available
security groups.");
      }
      else
      {
        vpcID = args[0];
      }

      if(vpcID.StartsWith("vpc-") || string.IsNullOrEmpty(vpcID))
      {
        // Create an EC2 client object
        var ec2Client = new AmazonEC2Client();

        // Enumerate the security groups
        await EnumerateGroups(ec2Client, vpcID);
      }
      else
      {
        Console.WriteLine("Could not find a valid VPC ID in the command-line
arguments:");
        Console.WriteLine($"{args[0]}");
      }
    }


    //
    // Method to enumerate the security groups
    private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
    {
      // A request object, in case we need it.
      var request = new DescribeSecurityGroupsRequest();

      // Put together the properties, if needed
      if(!string.IsNullOrEmpty(vpcID))
      {
        // We have a VPC ID. Find the security groups for just that VPC.
```

```
        Console.WriteLine($"\nGetting security groups for VPC {vpcID}...\n");
        request.Filters.Add(new Filter
        {
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });
      }

      // Get the list of security groups
      DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

      // Display the list of security groups.
      foreach (SecurityGroup item in response.SecurityGroups)
      {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
        Console.WriteLine();
      }
    }
  }
}
```

Considérations supplémentaires

- Notez que dans le cas d'un VPC, le filtre est construit avec la `Name` partie de la paire nom-valeur définie sur « vpc-id ». Ce nom provient de la description de la `Filters` propriété de la DescribeSecurityGroupsRequestclasse.

- Pour obtenir la liste complète de vos groupes de sécurité, vous pouvez également utiliser DescribeSecurityGroupsAsync sans paramètres.

- Vous pouvez vérifier les résultats en consultant la liste des groupes de sécurité dans la console Amazon EC2.

Création de groupes de sécurité

Cet exemple montre comment utiliser le AWS SDK pour .NET pour créer un groupe de sécurité. Vous pouvez fournir l'ID d'un VPC existant afin de créer un groupe de sécurité pour EC2 dans un VPC. Si vous ne fournissez pas un tel identifiant, le nouveau groupe de sécurité sera destiné à EC2-Classic si votre AWS compte le permet.

Si vous ne fournissez pas d'identifiant VPC et que votre AWS compte ne prend pas en charge EC2-Classic, le nouveau groupe de sécurité appartiendra au VPC par défaut de votre compte.

> **Warning**
>
> EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog EC2-Classic Networking is Retiring — Here's How to Prepare.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Rechercher des groupes de sécurité existants
- Création d'un groupe de sécurité
- Code complet

Rechercher des groupes de sécurité existants

L'extrait de code suivant recherche les groupes de sécurité existants portant le nom donné dans le VPC donné.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Création d'un groupe de sécurité

L'extrait suivant crée un nouveau groupe de sécurité si aucun groupe portant ce nom n'existe dans le VPC donné. Si aucun VPC n'est indiqué et qu'un ou plusieurs groupes portant ce nom existent, l'extrait renvoie simplement la liste des groupes.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
```

```
    {
      createRequest.Description = "My .NET example security group for EC2-Classic";
    }
    else
    {
      createRequest.VpcId = vpcID;
      createRequest.Description = "My .NET example security group for EC2-VPC";
    }
    CreateSecurityGroupResponse createResponse =
      await ec2Client.CreateSecurityGroupAsync(createRequest);

    // Return the new security group
    DescribeSecurityGroupsResponse describeResponse =
      await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
        GroupIds = new List<string>() { createResponse.GroupId }
      });
    return describeResponse.SecurityGroups;
  }
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe CreateSecurityGroupRequest

  Classe CreateSecurityGroupResponse

  Classe DescribeSecurityGroupsRequest

  Classe DescribeSecurityGroupsResponse

Filtre de classe

Classe SecurityGroup

Le code

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

namespace EC2CreateSecGroup
{
  // = = =
```

```
    // groupName has a value and vpcID either has a value or is null (which is fine)
    // Create the new security group and display information about it
    var securityGroups =
      await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
    Console.WriteLine("Information about the security group(s):");
    foreach(var group in securityGroups)
    {
      Console.WriteLine($"\nGroupName: {group.GroupName}");
      Console.WriteLine($"GroupId: {group.GroupId}");
      Console.WriteLine($"Description: {group.Description}");
      Console.WriteLine($"VpcId (if any): {group.VpcId}");
    }
  }


  //
  // Method to create a new security group (either EC2-Classic or EC2-VPC)
  // If vpcID is empty, the security group will be for EC2-Classic
  private static async Task<List<SecurityGroup>> CreateSecurityGroup(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
  {
    // See if one or more security groups with that name
    // already exist in the given VPC. If so, return the list of them.
    var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
    if (securityGroups.Count > 0)
    {
      Console.WriteLine(
        $"\nOne or more security groups with name {groupName} already exist.\n");
      return securityGroups;
    }

    // If the security group doesn't already exists, create it.
    var createRequest = new CreateSecurityGroupRequest{
      GroupName = groupName
    };
    if(string.IsNullOrEmpty(vpcID))
    {
      createRequest.Description = "Security group for .NET code example (no VPC
specified)";
    }
    else
    {
      createRequest.VpcId = vpcID;
```

```
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }


    //
    // Method to determine if a security group with the specified name
    // already exists in the VPC
    private static async Task<List<SecurityGroup>> FindSecurityGroups(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      var request = new DescribeSecurityGroupsRequest();
      request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
      });
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
```

```
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n     If vpc-id isn't present, the security group will be" +
        "\n     for EC2-Classic (if your AWS account supports this)" +
        "\n     or will use the default VCP for EC2-VPC.");
    }
 }


 // = = =
```

```
          if((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);
    }
  }
```

```
}
```

Mise à jour des groupes de sécurité

Cet exemple montre comment utiliser le pour AWS SDK pour .NET ajouter une règle à un groupe de sécurité. En particulier, l'exemple ajoute une règle pour autoriser le trafic entrant sur un port TCP donné, qui peut être utilisée, par exemple, pour les connexions à distance à une instance EC2. L'application prend l'identifiant d'un groupe de sécurité existant, une adresse IP (ou plage d'adresses) au format CIDR et éventuellement un numéro de port TCP. Il ajoute ensuite une règle entrante au groupe de sécurité donné.

> **Note**
>
> Pour utiliser cet exemple, vous avez besoin d'une adresse IP (ou d'une plage d'adresses) au format CIDR. Consultez la section Considérations supplémentaires à la fin de cette rubrique pour connaître les méthodes permettant d'obtenir l'adresse IP de votre ordinateur local.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Ajouter une règle de trafic entrant
- Code complet
- Considérations supplémentaires

Ajouter une règle de trafic entrant

L'extrait suivant ajoute une règle entrante à un groupe de sécurité pour une adresse IP (ou plage) et un port TCP particuliers.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
    //
    // Method that adds a TCP ingress rule to a security group
    private static async Task AddIngressRule(
      IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
    {
```

```
    // Create an object to hold the request information for the rule.
    // It uses an IpPermission object to hold the IP information for the rule.
    var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
      GroupId = groupID};
    ingressRequest.IpPermissions.Add(new IpPermission{
      IpProtocol = "tcp",
      FromPort = port,
      ToPort = port,
      Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
    });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
      await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
  }
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe AuthorizeSecurityGroupIngressRequest

  Classe AuthorizeSecurityGroupIngressResponse

  Classe IpPermission

  Classe IpRange

Le code

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

namespace EC2AddRuleForRDP
{
  // = = =
```

```
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
    }


    //
    // Method that adds a TCP ingress rule to a security group
    private static async Task AddIngressRule(
      IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
    {
      // Create an object to hold the request information for the rule.
      // It uses an IpPermission object to hold the IP information for the rule.
      var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
      ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
      });

      // Create the inbound rule for the security group
      AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
      Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2AddRuleForRDP -g <group-id> -i <ip-address> [-p <port>]" +
        "\n  -g, --group-id: The ID of the security group to which you want to add the
inbound rule." +
        "\n  -i, --ip-address: An IP address or address range in CIDR format." +
        "\n  -p, --port: The TCP port number. Defaults to 3389.");
    }
  }


  // = = =
```

```
 // 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);
    }
  }

}
```

Considérations supplémentaires

- Si vous ne fournissez pas de numéro de port, l'application utilise par défaut le port 3389. Il s'agit du port pour Windows RDP, qui vous permet de vous connecter à une instance EC2 exécutant Windows. Si vous lancez une instance EC2 exécutant Linux, vous pouvez utiliser le port TCP 22 (SSH) à la place.

- Notez que l'exemple prend la valeur `IpProtocol` « tcp ». Les valeurs de se `IpProtocol` trouvent dans la description de la `IpProtocol` propriété de la IpPermissionclasse.

- Vous aurez peut-être besoin de l'adresse IP de votre ordinateur local lorsque vous utiliserez cet exemple. Voici certains des moyens par lesquels vous pouvez obtenir l'adresse.
  - Si votre ordinateur local (à partir duquel vous allez vous connecter à votre instance EC2) possède une adresse IP publique statique, vous pouvez utiliser un service pour obtenir cette adresse. L'un de ces services est http://checkip.amazonaws.com/. Pour en savoir plus sur l'autorisation du trafic entrant, consultez les sections Ajouter des règles à un groupe de sécurité et Règles de groupe de sécurité pour différents cas d'utilisation dans le guide de l'utilisateur Amazon EC2.
  - Une autre méthode pour obtenir l'adresse IP de votre ordinateur local consiste à utiliser la console Amazon EC2.

    Sélectionnez l'un de vos groupes de sécurité, sélectionnez l'onglet Règles entrantes, puis choisissez Modifier les règles entrantes. Dans une règle entrante, ouvrez le menu déroulant dans la colonne Source et choisissez Mon adresse IP pour voir l'adresse IP de votre ordinateur local au format CIDR. Assurez-vous d'annuler l'opération.

- Vous pouvez vérifier les résultats de cet exemple en examinant la liste des groupes de sécurité dans la console Amazon EC2.

Utilisation des paires de clés Amazon EC2

Amazon EC2 utilise le chiffrement à clé publique pour chiffrer et déchiffrer les informations de connexion. Le chiffrement à clé publique utilise une clé publique pour chiffrer les données, puis le destinataire utilise la clé privée pour déchiffrer les données. La clé publique et la clé privée constituent une paire de clés. Si vous souhaitez vous connecter à une instance EC2, vous devez spécifier une paire de clés lorsque vous la lancez, puis fournir la clé privée de la paire lorsque vous vous y connectez.

Lorsque vous lancez une instance EC2, vous pouvez créer une paire de clés pour celle-ci ou utiliser une paire de clés que vous avez déjà utilisée lors du lancement d'autres instances. Pour en savoir plus sur les paires de clés Amazon EC2, consultez la section Utilisation des paires de clés Amazon EC2 dans le guide de l'utilisateur Amazon EC2.

Pour plus d'informations sur le APIs et les prérequis, consultez la section parent (Utilisation avec Amazon EC2).

Rubriques

- Création et affichage de paires de clés
- Supprimer des paires de clés

Création et affichage de paires de clés

Cet exemple montre comment utiliser le AWS SDK pour .NET pour créer une paire de clés. L'application prend le nom de la nouvelle paire de clés et le nom d'un fichier PEM (avec une extension « .pem »). Il crée la paire de clés, écrit la clé privée dans le fichier PEM, puis affiche toutes les paires de clés disponibles. Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement toutes les paires de clés disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Créer la paire de clés
- Afficher les paires de clés disponibles
- Code complet
- Considérations supplémentaires

Créer la paire de clés

L'extrait suivant crée une paire de clés, puis stocke la clé privée dans le fichier PEM donné.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
  IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
  // Create the key pair
  CreateKeyPairResponse response =
    await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
      KeyName = keyPairName
    });
  Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

  // Save the private key in a PEM file
```

```
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
      writer.WriteLine(response.KeyPair.KeyMaterial);
    }
  }
```

Afficher les paires de clés disponibles

L'extrait suivant affiche la liste des paires de clés disponibles.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
  //
  // Method to show the key pairs that are available
  private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
  {
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyPairInfo item in response.KeyPairs)
      Console.WriteLine($"  {item.KeyName}");
  }
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe CreateKeyPairRequest

Classe CreateKeyPairResponse

Classe DescribeKeyPairsResponse

Classe KeyPairInfo

Le code

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

namespace EC2CreateKeyPair
{
  // = = =
```

```
  // Get the application arguments from the parsed list
  string keyPairName =
    CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
  string pemFileName =
    CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
  if(string.IsNullOrEmpty(keyPairName))
    CommandLine.ErrorExit("\nNo key pair name specified." +
      "\nRun the command with no arguments to see help.");
  if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
    CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
      "\nRun the command with no arguments to see help.");

  // Create the key pair
  await CreateKeyPair(ec2Client, keyPairName, pemFileName);
  await EnumerateKeyPairs(ec2Client);
}


//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
  IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
  // Create the key pair
  CreateKeyPairResponse response =
    await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
      KeyName = keyPairName
    });
  Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

  // Save the private key in a PEM file
  using (var s = new FileStream(pemFileName, FileMode.Create))
  using (var writer = new StreamWriter(s))
  {
    writer.WriteLine(response.KeyPair.KeyMaterial);
  }
}


//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
```

```
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
    }
 }


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

}
```

Considérations supplémentaires

- Après avoir exécuté l'exemple, vous pouvez voir la nouvelle paire de clés dans la console Amazon EC2.

- Lorsque vous créez une paire de clés, vous devez enregistrer la clé privée renvoyée car vous ne pourrez pas la récupérer ultérieurement.

Supprimer des paires de clés

Cet exemple montre comment utiliser le AWS SDK pour .NET pour supprimer une paire de clés. L'application prend le nom d'une paire de clés. Il supprime la paire de clés, puis affiche toutes les paires de clés disponibles. Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement toutes les paires de clés disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Suppression de la paire de clés
- Afficher les paires de clés disponibles
- Code complet

Suppression de la paire de clés

L'extrait de code suivant supprime une paire de clés.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Afficher les paires de clés disponibles

L'extrait suivant affiche la liste des paires de clés disponibles.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

Classe DeleteKeyPairRequest

Classe DescribeKeyPairsResponse

Classe KeyPairInfo

Le code

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2DeleteKeyPair
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      if(args.Length == 1)
      {
        // Delete a key pair (if it exists)
        await DeleteKeyPair(ec2Client, args[0]);

        // Display the key pairs that are left
        await EnumerateKeyPairs(ec2Client);
      }
      else
      {
        Console.WriteLine("\nUsage: EC2DeleteKeyPair keypair-name");
        Console.WriteLine("  keypair-name - The name of the key pair you want to
 delete.");
        Console.WriteLine("\nNo arguments specified.");
        Console.Write(
          "Do you want to see a list of the existing key pairs? ((y) or n): ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await EnumerateKeyPairs(ec2Client);
      }
```

```
    }


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


    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
  }
}
```

Afficher vos régions et zones de disponibilité Amazon EC2

Amazon EC2 est hébergé sur plusieurs sites dans le monde entier. Ces emplacements sont composés de régions et de zones de disponibilité. Chaque région est une zone géographique distincte qui compte plusieurs emplacements isolés appelés zones de disponibilité.

Pour en savoir plus sur les régions et les zones de disponibilité, consultez Régions et zones dans le guide de l'utilisateur Amazon EC2.

Cet exemple vous montre comment utiliser le pour AWS SDK pour .NET obtenir des informations sur les régions et les zones de disponibilité associées à un client EC2. L'application affiche les listes des régions et des zones de disponibilité disponibles pour un client EC2.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe DescribeAvailabilityZonesResponse

  Classe DescribeRegionsResponse

  Classe AvailabilityZone

  Région de 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);
    }
  }
}
```

Utilisation des instances Amazon EC2

Vous pouvez utiliser le AWS SDK pour .NET pour contrôler les instances Amazon EC2 à l'aide d'opérations telles que la création, le démarrage et la fin. Les rubriques de cette section fournissent quelques exemples de la manière de procéder. Pour en savoir plus sur les instances EC2, consultez les instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2.

Pour plus d'informations sur le APIs et les prérequis, consultez la section parent (Utilisation avec Amazon EC2).

Rubriques

- Lancement d'une instance Amazon EC2
- Interruption d'une instance Amazon EC2

Lancement d'une instance Amazon EC2

Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour lancer une ou plusieurs instances Amazon EC2 configurées de manière identique à partir de la même Amazon Machine Image (AMI). À l'aide de plusieurs entrées que vous fournissez, l'application lance une instance EC2, puis surveille l'instance jusqu'à ce qu'elle sorte de l'état « En attente ».

Lorsque votre instance EC2 est en cours d'exécution, vous pouvez vous y connecter à distance, comme décrit dans(facultatif) Connectez-vous à l'instance.

> **Warning**
>
> EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog EC2-Classic Networking is Retiring — Here's How to Prepare.

Les sections suivantes fournissent des extraits et d'autres informations pour cet exemple. Le code complet de l'exemple est affiché après les extraits et peut être créé et exécuté tel quel.

Rubriques

- Rassemblez ce dont vous avez besoin
- Lancer une instance
- Surveiller l'instance
- Code complet
- Considérations supplémentaires
- (facultatif) Connectez-vous à l'instance
- Nettoyage

Rassemblez ce dont vous avez besoin

Pour lancer une instance EC2, vous aurez besoin de plusieurs éléments.

- Un VPC dans lequel l'instance sera lancée. S'il s'agit d'une instance Windows et que vous vous y connectez via RDP, le VPC aura probablement besoin d'une passerelle Internet attachée, ainsi que d'une entrée pour la passerelle Internet dans la table de routage. Pour plus d'informations, consultez Passerelles Internet dans le Guide de l'utilisateur Amazon VPC.

- L'ID d'un sous-réseau existant dans le VPC où l'instance sera lancée. Pour le trouver ou le créer facilement, vous pouvez vous connecter à la console Amazon VPC, mais vous pouvez également l'obtenir par programmation en utilisant les méthodes et. CreateSubnetAsyncDescribeSubnetsAsync

> **Note**
>
> Si vous ne fournissez pas ce paramètre, la nouvelle instance est lancée dans le VPC par défaut de votre compte.

- L'ID d'un groupe de sécurité existant appartenant au VPC où l'instance sera lancée. Pour de plus amples informations, veuillez consulter Travailler avec des groupes de sécurité dans Amazon EC2.

- Si vous souhaitez vous connecter à la nouvelle instance, le groupe de sécurité mentionné précédemment doit disposer d'une règle entrante appropriée autorisant le trafic SSH sur le port 22 (instance Linux) ou le trafic RDP sur le port 3389 (instance Windows). Pour plus d'informations sur la procédure à suivreMise à jour des groupes de sécurité, voir Considérations supplémentaires notamment la fin de cette rubrique.

- L'Amazon Machine Image (AMI) qui sera utilisée pour créer l'instance. Pour plus d'informations AMIs, consultez Amazon Machine Images (AMIs) dans le guide de l'utilisateur Amazon EC2. Consultez notamment les sections Rechercher une AMI et Partager AMIs.

- Le nom d'une paire de clés EC2 existante, qui est utilisée pour se connecter à la nouvelle instance. Pour de plus amples informations, veuillez consulter Utilisation des paires de clés Amazon EC2.

- Nom du fichier PEM qui contient la clé privée de la paire de clés EC2 mentionnée précédemment. Le fichier PEM est utilisé lorsque vous vous connectez à distance à l'instance.

Lancer une instance

L'extrait de code suivant lance une instance EC2.

L'exemple situé à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Surveiller l'instance

L'extrait de code suivant surveille l'instance jusqu'à ce qu'elle sorte de l'état « En attente ».

L'exemple situé à la fin de cette rubrique montre cet extrait en cours d'utilisation.

Consultez la InstanceStateclasse pour connaître les valeurs valides de la `Instance.State.Code` propriété.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client

  Classe InstanceType
- Espace de noms Amazon.ec2.model

  Classe DescribeInstancesRequest

  Classe DescribeInstancesResponse

  Instance de classe

  Classe InstanceNetworkInterfaceSpecification

  Classe RunInstancesRequest

  Classe RunInstancesResponse

Le code

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

namespace EC2LaunchInstance
{
  // = = =
```

```
        return;
      }

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

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

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

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


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

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

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

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



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

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

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

      return instanceIds;
    }


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

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

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
```

```
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


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

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

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

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

      return parsedArgs;
    }

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

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

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

}
```

Considérations supplémentaires

- Lorsque vous vérifiez l'état d'une instance EC2, vous pouvez ajouter un filtre à la `Filter` propriété de l'DescribeInstancesRequestobjet. Grâce à cette technique, vous pouvez limiter la demande à certaines instances, par exemple les instances dotées d'une balise spécifique spécifiée par l'utilisateur.

- Par souci de concision, des valeurs typiques ont été attribuées à certaines propriétés. L'une ou l'ensemble de ces propriétés peuvent plutôt être déterminées par programmation ou par saisie par l'utilisateur.

- Les valeurs que vous pouvez utiliser pour les `MaxCount` propriétés `MinCount` et de l'RunInstancesRequestobjet sont déterminées par la zone de disponibilité cible et le nombre maximum d'instances autorisées pour le type d'instance. Pour plus d'informations, consultez la section Combien d'instances puis-je exécuter dans Amazon EC2 dans la FAQ générale d'Amazon EC2.

- Si vous souhaitez utiliser un type d'instance différent de celui de cet exemple, vous avez le choix entre plusieurs types d'instance. Pour plus d'informations, consultez les types d'instances Amazon EC2 dans le guide de l'utilisateur Amazon EC2. Consultez également les détails du type d'instance et l'explorateur de types d'instance.

- Vous pouvez également associer un rôle IAM à une instance lorsque vous la lancez. Pour ce faire, créez un IamInstanceProfileSpecificationobjet dont la `Name` propriété est définie sur le nom d'un rôle IAM. Ajoutez ensuite cet objet à la `IamInstanceProfile` propriété de l'RunInstancesRequestobjet.

  > **Note**
  >
  > Pour lancer une instance EC2 à laquelle un rôle IAM est attaché, la configuration d'un utilisateur IAM doit inclure certaines autorisations. Pour plus d'informations sur les autorisations requises, consultez la section Accorder à un utilisateur l'autorisation de transmettre un rôle IAM à une instance dans le guide de l'utilisateur Amazon EC2.

(facultatif) Connectez-vous à l'instance

Une fois qu'une instance est en cours d'exécution, vous pouvez vous y connecter à distance à l'aide du client distant approprié. Pour les instances Linux et Windows, vous avez besoin de l'adresse IP publique ou du nom DNS public de l'instance. Vous avez également besoin des éléments suivants.

Pour les instances Linux

Vous pouvez utiliser un client SSH pour vous connecter à votre instance Linux. Assurez-vous que le groupe de sécurité que vous avez utilisé lorsque vous avez lancé l'instance autorise le trafic SSH sur le port 22, comme décrit dansMise à jour des groupes de sécurité.

Vous avez également besoin de la partie privée de la paire de clés que vous avez utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations, consultez Connect to your Linux instance dans le guide de l'utilisateur Amazon EC2.

Pour les instances Windows

Vous pouvez utiliser un client RDP pour vous connecter à votre instance. Assurez-vous que le groupe de sécurité que vous avez utilisé lorsque vous avez lancé l'instance autorise le trafic RDP sur le port 3389, comme décrit dans. Mise à jour des groupes de sécurité

Vous avez également besoin du mot de passe administrateur. Vous pouvez l'obtenir en utilisant l'exemple de code suivant, qui nécessite l'ID de l'instance et la partie privée de la paire de clés utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations, consultez Connect to your Windows instance dans le guide de l'utilisateur Amazon EC2.

> ⚠ Warning
>
> Cet exemple de code renvoie le mot de passe administrateur en texte brut pour votre instance.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client
- Espace de noms Amazon.ec2.model

  Classe GetPasswordDataRequest

  Classe GetPasswordDataResponse

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
```

```
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
    {
      string password = string.Empty;
      GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
          InstanceId = instanceID});
      if(response.PasswordData != null)
      {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
      }
      else
      {
        Console.WriteLine($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
      }
      return password;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\n  -i, --instance-id: The name of the EC2 instance." +
        "\n  -p, --pem-filename: The name of the PEM file with the private key.");
    }
  }

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

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

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

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

      return parsedArgs;
    }

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

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

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

}
```

Nettoyage

Lorsque vous n'avez plus besoin de votre instance EC2, veillez à la mettre hors service, comme décrit dansInterruption d'une instance Amazon EC2.

Interruption d'une instance Amazon EC2

Lorsque vous n'avez plus besoin d'une ou de plusieurs de vos instances Amazon EC2, vous pouvez les résilier.

Cet exemple vous montre comment utiliser le pour mettre fin AWS SDK pour .NET à des instances EC2. Il prend un ID d'instance en entrée.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client

- Espace de noms 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);
      }
    }
  }
}
```

Après avoir exécuté l'exemple, il est conseillé de vous connecter à la console Amazon EC2 pour vérifier que l'instance EC2 a été résiliée.

Tutoriel sur les instances Spot Amazon EC2

Ce didacticiel explique comment utiliser le AWS SDK pour .NET pour gérer les instances Spot Amazon EC2.

Présentation de

Les instances Spot vous permettent de demander de la capacité Amazon EC2 inutilisée à un prix inférieur au prix à la demande. Cela peut réduire considérablement vos coûts EC2 pour les applications susceptibles d'être interrompues.

Vous trouverez ci-dessous un résumé détaillé de la manière dont les instances Spot sont demandées et utilisées.

1. Créez une demande d'instance Spot en spécifiant le prix maximum que vous êtes prêt à payer.
2. Lorsque la demande est satisfaite, exécutez l'instance comme vous le feriez pour n'importe quelle autre instance Amazon EC2.
3. Exécutez l'instance aussi longtemps que vous le souhaitez, puis mettez-la hors service, sauf si le prix au comptant change de telle sorte que l'instance soit résiliée pour vous.
4. Nettoyez la demande d'instance Spot lorsque vous n'en avez plus besoin afin que les instances Spot ne soient plus créées.

Il s'agit d'un aperçu de très haut niveau des instances Spot. Pour mieux comprendre les instances Spot, consultez la section Instances Spot dans le guide de l'utilisateur Amazon EC2.

À propos de ce didacticiel

Au fur et à mesure que vous suivez ce didacticiel, vous utilisez le AWS SDK pour .NET pour effectuer les opérations suivantes :

- Créer une demande d'instance Spot
- Déterminez à quel moment la demande d'instance Spot a été satisfaite
- Annuler la demande d'instance Spot
- Résilier les instances associées

Les sections suivantes fournissent des extraits de code et d'autres informations pour cet exemple. Le code complet de l'exemple est affiché après les extraits et peut être créé et exécuté tel quel.

Rubriques

- Conditions préalables
- Rassemblez ce dont vous avez besoin
- Création d'une demande d'instance Spot
- Déterminez l'état de votre demande d'instance Spot
- Nettoyez vos demandes d'instance Spot
- Nettoyez vos instances Spot
- Code complet
- Considérations supplémentaires

Conditions préalables

Pour plus d'informations sur le APIs et les prérequis, consultez la section parent (Utilisation avec Amazon EC2).

Rassemblez ce dont vous avez besoin

Pour créer une demande d'instance Spot, vous aurez besoin de plusieurs éléments.

- Le nombre d'instances et leur type d'instance. Vous avez le choix entre plusieurs types d'instances. Pour plus d'informations, consultez les types d'instances Amazon EC2 dans le guide de l'utilisateur

Amazon EC2. Consultez également les détails du type d'instance et l'explorateur de types d'instance.

Le nombre par défaut pour ce didacticiel est 1.

- L'Amazon Machine Image (AMI) qui sera utilisée pour créer l'instance. Pour plus d'informations AMIs, consultez Amazon Machine Images (AMIs) dans le guide de l'utilisateur Amazon EC2. Consultez notamment les sections Rechercher une AMI et Partager AMIs.

- Le prix maximum que vous êtes prêt à payer par heure d'instance. Vous pouvez consulter les prix de tous les types d'instances (pour les instances à la demande et les instances ponctuelles) sur la page de tarification d'Amazon EC2. Le prix par défaut de ce didacticiel est expliqué plus loin.

- Si vous souhaitez vous connecter à distance à une instance, un groupe de sécurité doté de la configuration et des ressources appropriées. Ceci est décrit dans Travailler avec des groupes de sécurité dans Amazon EC2 les informations relatives à la collecte de ce dont vous avez besoin et à la connexion à une instance dansLancement d'une instance Amazon EC2. Pour des raisons de simplicité, ce didacticiel utilise le groupe de sécurité nommé default que possèdent tous les nouveaux AWS comptes.

Il existe de nombreuses façons d'aborder la demande d'instances Spot. Les stratégies les plus courantes sont les suivantes :

- Faites des demandes qui coûteront certainement moins cher que la tarification à la demande.
- Effectuez des demandes en fonction de la valeur du calcul obtenu.
- Faites des demandes afin d'acquérir de la capacité de calcul le plus rapidement possible.

Les explications suivantes font référence à l'historique des tarifs des instances Spot dans le guide de l'utilisateur Amazon EC2.

Réduisez les coûts en dessous de la demande

Vous avez une tâche de traitement par lot dont l'exécution prendra un certain nombre d'heures ou de jours. Toutefois, vous êtes flexible quant aux dates et heures de début et de fin de la tâche. Vous voulez savoir si vous pouvez exécuter cette tâche à un coût inférieur à celui des instances à la demande.

Vous examinez l'historique des prix au comptant pour les types d'instances à l'aide de la console Amazon EC2 ou de l'API Amazon EC2. Une fois que vous avez analysé l'historique des prix pour le type d'instance souhaité dans une zone de disponibilité donnée, deux approches sont possibles pour votre demande :

- Spécifiez une demande se situant dans la partie supérieure de la fourchette des prix ponctuels, qui sont toujours inférieurs au prix à la demande, en anticipant que votre demande d'instance ponctuelle unique sera probablement satisfaite et exécutée pendant une durée de calcul consécutive suffisante pour terminer le travail.
- Spécifiez une demande à la limite inférieure de la plage de prix et envisagez de combiner plusieurs instances lancées au fil du temps via une demande persistante. Les instances vont s'exécuter suffisamment longtemps, au total, pour terminer la tâche avec un coût total plus faible.

Ne payez pas plus que la valeur du résultat

Vous avez une tâche de traitement de données à exécuter. Vous comprenez suffisamment la valeur des résultats du travail pour savoir combien ils valent en termes de coûts informatiques.

Après avoir analysé l'historique des prix au comptant pour votre type d'instance, vous choisissez un prix dont le coût du temps de calcul ne dépasse pas la valeur des résultats de la tâche. Vous créez une demande persistante et faites en sorte qu'elle s'exécute de façon intermittente selon que le prix Spot est égal ou inférieur à votre demande.

Acquérez rapidement de la capacité informatique

Vous avez un besoin imprévu à court terme de capacité supplémentaire qui n'est pas disponible via les instances à la demande. Après avoir analysé l'historique des prix au comptant pour votre type d'instance, vous choisissez un prix supérieur au prix historique le plus élevé afin d'améliorer considérablement les chances que votre demande soit traitée rapidement et vous poursuivez le calcul jusqu'à ce qu'elle soit complète.

Une fois que vous avez rassemblé ce dont vous avez besoin et que vous avez choisi une stratégie, vous êtes prêt à demander une instance Spot. Pour ce didacticiel, le prix maximal d'instance spot par défaut est défini pour être le même que celui à la demande (qui est de 0,003 $pour ce didacticiel). Fixer le prix de cette manière maximise les chances de satisfaire la demande.

Création d'une demande d'instance Spot

L'extrait suivant explique comment créer une demande d'instance Spot avec les éléments que vous avez collectés précédemment.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

La valeur importante renvoyée par cette méthode est l'ID de demande d'instance Spot, qui est contenu dans le `SpotInstanceRequestId` membre de l'SpotInstanceRequestobjet renvoyé.

> **Note**
>
> Toutes les instances Spot lancées vous seront facturées. Pour éviter des coûts inutiles, veillez à annuler toutes les demandes et à mettre fin à toutes les instances.

Déterminez l'état de votre demande d'instance Spot

L'extrait suivant vous montre comment obtenir des informations sur votre demande d'instance Spot. Vous pouvez utiliser ces informations pour prendre certaines décisions dans votre code, par exemple si vous souhaitez continuer à attendre qu'une demande d'instance Spot soit traitée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
//
```

```
    // 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];
    }
```

La méthode renvoie des informations sur la demande d'instance Spot, telles que l'ID de l'instance, son état et son code d'état. Pour plus d'informations sur les codes de statut des demandes d'instance Spot, consultez la section État des demandes Spot dans le guide de l'utilisateur Amazon EC2.

Nettoyez vos demandes d'instance Spot

Lorsque vous n'avez plus besoin de demander des instances ponctuelles, il est important d'annuler toutes les demandes en suspens afin d'éviter qu'elles ne soient traitées à nouveau. L'extrait suivant explique comment annuler une demande d'instance Spot.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Nettoyez vos instances Spot

Pour éviter des coûts inutiles, il est important de résilier toutes les instances créées à partir de demandes d'instances Spot ; le simple fait d'annuler les demandes d'instance Spot ne mettra pas fin à vos instances, ce qui signifie qu'elles continueront à vous être facturées. L'extrait suivant explique

comment mettre fin à une instance après avoir obtenu l'identifiant d'instance d'une instance Spot active.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

L'exemple de code suivant appelle les méthodes décrites précédemment pour créer et annuler une demande d'instance Spot et mettre fin à une instance Spot.

Références du SDK

NuGet colis :

- AWSSDK.EC2

Éléments de programmation :

- Espace de noms Amazon.ec2

  Classe Amazon EC2 Client

  Classe InstanceType

- Espace de noms Amazon.ec2.model

  Classe CancelSpotInstanceRequestsRequest

  Classe DescribeSpotInstanceRequestsRequest

  Classe DescribeSpotInstanceRequestsResponse

  Classe InstanceStateChange

  Classe LaunchSpecification

  Classe RequestSpotInstancesRequest

  Classe RequestSpotInstancesResponse

  Classe SpotInstanceRequest

  Classe TerminateInstancesRequest

  Classe TerminateInstancesResponse

Le code

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

```
namespace EC2SpotInstanceRequests
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Some default values.
      // These could be made into command-line arguments instead.
      var instanceType = InstanceType.T1Micro;
      string securityGroupName = "default";
      string spotPrice = "0.003";
      int instanceCount = 1;

      // Parse the command line arguments
      if((args.Length != 1) || (!args[0].StartsWith("ami-")))
      {
        Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
        Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
 Instances.");
        return;
      }

      // Create the Amazon EC2 client.
      var ec2Client = new AmazonEC2Client();

      // Create the Spot Instance request and record its ID
      Console.WriteLine("\nCreating spot instance request...");
      var req = await CreateSpotInstanceRequest(
        ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
      string requestId = req.SpotInstanceRequestId;

      // Wait for an EC2 Spot Instance to become active
      Console.WriteLine(
        $"Waiting for Spot Instance request with ID {requestId} to become active...");
      int wait = 1;
      var start = DateTime.Now;
      while(true)
      {
        Console.Write(".");

        // Get and check the status to see if the request has been fulfilled.
        var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
        if(requestInfo.Status.Code == "fulfilled")
        {
```

```
        Console.WriteLine($"\nSpot Instance request {requestId} " +
          $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
        break;
      }

      // Wait a bit and try again, longer each time (1, 2, 4, ...)
      Thread.Sleep(wait);
      wait = wait * 2;
    }

    // Show the user how long it took to fulfill the Spot Instance request.
    TimeSpan span = DateTime.Now.Subtract(start);
    Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

    // Perform actions here as needed.
    // For this example, simply wait for the user to hit a key.
    // That gives them a chance to look at the EC2 console to see
    // the running instance if they want to.
    Console.WriteLine("Press any key to start the cleanup...");
    Console.ReadKey(true);

    // Cancel the request.
    // Do this first to make sure that the request can't be re-fulfilled
    // once the Spot Instance has been terminated.
    Console.WriteLine("Canceling Spot Instance request...");
    await CancelSpotInstanceRequest(ec2Client, requestId);

    // Terminate the Spot Instance that's running.
    Console.WriteLine("Terminating the running Spot Instance...");
    await TerminateSpotInstance(ec2Client, requestId);

    Console.WriteLine("Done. Press any key to exit...");
    Console.ReadKey(true);
  }


  //
  // Method to create a Spot Instance request
  private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,
    InstanceType instanceType, string spotPrice, int instanceCount)
  {
    var launchSpecification = new LaunchSpecification{
      ImageId = amiId,
```

```
        InstanceType = instanceType
      };
      launchSpecification.SecurityGroups.Add(securityGroupName);
      var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
      };

      RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
      return result.SpotInstanceRequests[0];
    }


    //
    // Method to get information about a Spot Instance request, including the status,
    // instance ID, etc.
    // It gets the information for a specific request (as opposed to all requests).
    private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
      return describeResponse.SpotInstanceRequests[0];
    }


    //
    // Method to cancel a Spot Instance request
    private static async Task CancelSpotInstanceRequest(
      IAmazonEC2 ec2Client, string requestId)
    {
      var cancelRequest = new CancelSpotInstanceRequestsRequest();
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }


    //
```

```
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
 SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
    }
  }
}
```

Considérations supplémentaires

- Après avoir exécuté le didacticiel, il est conseillé de vous connecter à la console Amazon EC2 pour vérifier que la demande d'instance Spot a été annulée et que l'instance Spot a été résiliée.

Accès Gestion des identités et des accès AWS (IAM) à l'aide du AWS SDK pour .NET

The AWS SDK pour .NET supports Gestion des identités et des accès AWS, qui est un service Web qui permet aux AWS clients de gérer les utilisateurs et les autorisations des utilisateurs dans AWS.

Un utilisateur Gestion des identités et des accès AWS (IAM) est une entité que vous créez dans AWS. L'entité représente une personne ou une application qui interagit avec AWS. Pour plus d'informations sur les utilisateurs IAM, voir Utilisateurs IAM et limites IAM et STS dans le Guide de l'utilisateur IAM.

Vous accordez des autorisations à un utilisateur en créant une politique IAM. La politique contient un document de politique qui répertorie les actions qu'un utilisateur peut effectuer et les ressources que ces actions peuvent affecter. Pour plus d'informations sur les politiques IAM, consultez la section Politiques et autorisations du Guide de l'utilisateur IAM.

> **Warning**
>
> Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que AWS IAM Identity Center.

APIs

Les AWS SDK pour .NET offres APIs pour les clients IAM. Ils vous APIs permettent de travailler avec les fonctionnalités IAM telles que les utilisateurs, les rôles et les clés d'accès.

Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la référence de l'AWS SDK pour .NET API (et faites défiler la page jusqu'à « Amazon »). IdentityManagement«).

Cette section contient également un exemple qui explique comment associer un rôle IAM aux instances Amazon EC2 afin de faciliter la gestion des informations d'identification.

Les IAM APIs sont fournis par leAWSSDK. IdentityManagement NuGetcolis.

Conditions préalables

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

Rubriques

- Création de politiques gérées par IAM à partir de JSON
- Afficher le document de politique d'une politique gérée par IAM
- Octroi d'accès à l'aide d'un rôle IAM

Création de politiques gérées par IAM à partir de JSON

Cet exemple vous montre comment utiliser le pour créer une politique gérée par IAM AWS SDK pour .NET à partir d'un document de stratégie donné au format JSON. L'application crée un objet client IAM, lit le document de politique à partir d'un fichier, puis crée la stratégie.

> **Note**
>
> Pour un exemple de document de politique au format JSON, consultez les considérations supplémentaires à la fin de cette rubrique.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Création de la stratégie
- Code complet
- Considérations supplémentaires

Création de la stratégie

L'extrait de code suivant crée une politique gérée par IAM avec le nom et le document de stratégie indiqués.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.IdentityManagement

Éléments de programmation :

- Espace de noms Amazon. IdentityManagement

  Classe AmazonIdentityManagementServiceClient
- Espace de noms Amazon. IdentityManagement.Modèle

  Classe CreatePolicyRequest

  Classe CreatePolicyResponse

Le code

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

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
    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($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  Arn: {response.Policy.Arn}");
    }


    //
    // Method to create an IAM policy from a JSON file
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
 jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
    }
```

```
    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
document.");
    }
 }


 // = = =
```

```
          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);
```

```
    }
  }

}
```

Considérations supplémentaires

- Voici un exemple de document de politique que vous pouvez copier dans un fichier JSON et utiliser comme entrée pour cette application :

  JSON

```
{
  "Version":"2012-10-17",
  "Id"  : "DotnetTutorialPolicy",
  "Statement" : [
    {
      "Sid" : "DotnetTutorialPolicyS3",
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "DotnetTutorialPolicyPolly",
      "Effect": "Allow",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": "*"
    }
  ]
}
```

- Vous pouvez vérifier que la politique a été créée en consultant la console IAM. Dans la liste déroulante des politiques de filtrage, sélectionnez Géré par le client. Supprimez la politique lorsque vous n'en avez plus besoin.

- Pour plus d'informations sur la création de politiques, consultez Création de politiques IAM et la référence de politique IAM JSON dans le guide de l'utilisateur IAM

Afficher le document de politique d'une politique gérée par IAM

Cet exemple montre comment utiliser le AWS SDK pour .NET pour afficher un document de politique. L'application crée un objet client IAM, trouve la version par défaut de la politique gérée IAM donnée, puis affiche le document de stratégie au format JSON.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Trouvez la version par défaut
- Afficher le document de politique
- Code complet

Trouvez la version par défaut

L'extrait suivant trouve la version par défaut de la politique IAM donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Afficher le document de politique

L'extrait suivant affiche le document de politique au format JSON de la stratégie IAM donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.IdentityManagement

Éléments de programmation :

- Espace de noms Amazon. IdentityManagement

  Classe AmazonIdentityManagementServiceClient
- Espace de noms Amazon. IdentityManagement.Modèle

  Classe GetPolicyVersionRequest

  Classe GetPolicyVersionResponse

  Classe ListPolicyVersionsRequest

  Classe ListPolicyVersionsResponse

  Classe PolicyVersion

Le code

```
using System;
using System.Web;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamDisplayPolicyJson
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      if(args.Length != 1)
      {
        Console.WriteLine("\nUsage: IamDisplayPolicyJson policy-arn");
        Console.WriteLine("   policy-arn: The ARN of the policy to retrieve.");
        return;
      }
      if(!args[0].StartsWith("arn:"))
      {
        Console.WriteLine("\nCould not find policy ARN in the command-line
 arguments:");
        Console.WriteLine($"{args[0]}");
        return;
```

```
    }

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

    // Retrieve and display the policy document of the given policy
    string defaultVersion = await GetDefaultVersion(iamClient, args[0]);
    if(string.IsNullOrEmpty(defaultVersion))
      Console.WriteLine($"Could not find the default version for policy {args[0]}.");
    else
      await ShowPolicyDocument(iamClient, args[0], defaultVersion);
  }


  //
  // Method to determine the default version of an IAM policy
  // Returns a string with the version
  private static async Task<string> GetDefaultVersion(
    IAmazonIdentityManagementService iamClient, string policyArn)
  {
    // Retrieve all the versions of this policy
    string defaultVersion = string.Empty;
    ListPolicyVersionsResponse reponseVersions =
      await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
        PolicyArn = policyArn});

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

    return defaultVersion;
  }


  //
  // Method to retrieve and display the policy document of an IAM policy
  private static async Task ShowPolicyDocument(
```

```
      IAmazonIdentityManagementService iamClient, string policyArn, string
 defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
  }
}
```

Octroi d'accès à l'aide d'un rôle IAM

Ce didacticiel explique comment utiliser le AWS SDK pour .NET pour activer les rôles IAM sur les instances Amazon EC2.

Présentation de

Toutes les demandes AWS doivent être signées cryptographiquement à l'aide des informations d'identification émises par AWS. Par conséquent, vous avez besoin d'une stratégie pour gérer les informations d'identification des applications qui s'exécutent sur des instances Amazon EC2. Vous devez distribuer, stocker et faire pivoter ces informations d'identification en toute sécurité, mais également les garder accessibles aux applications.

Les rôles IAM vous permettent de gérer efficacement ces informations d'identification. Vous créez un rôle IAM et vous le configurez avec les autorisations requises par une application, puis vous attachez ce rôle à une instance EC2. Pour en savoir plus sur les avantages de l'utilisation des rôles IAM, consultez la section Rôles IAM pour Amazon EC2 dans le guide de l'utilisateur Amazon EC2. Consultez également les informations sur les rôles IAM dans le guide de l'utilisateur IAM.

Pour une application créée à l'aide du AWS SDK pour .NET, lorsque l'application construit un objet client pour un AWS service, l'objet recherche des informations d'identification provenant de plusieurs sources potentielles. L'ordre dans lequel la recherche est effectué est indiqué dansRésolution des informations d'identification et des profils.

Si l'objet client ne trouve pas d'informations d'identification provenant d'une autre source, il récupère des informations d'identification temporaires dotées des mêmes autorisations que celles qui ont été configurées dans le rôle IAM et figurent dans les métadonnées de l'instance EC2. Ces informations d'identification sont utilisées pour effectuer des appels AWS depuis l'objet client.

À propos de ce didacticiel

Au fur et à mesure que vous suivez ce didacticiel, vous utilisez le AWS SDK pour .NET (et d'autres outils) pour lancer une instance Amazon EC2 avec un rôle IAM attaché, puis vous verrez une application sur l'instance utilisant les autorisations du rôle IAM.

Rubriques

- Création d'un exemple d'application Amazon S3
- Créer un rôle IAM
- Lancez une instance EC2 et attachez le rôle IAM
- Connectez-vous à l'instance EC2
- Exécutez l'exemple d'application sur l'instance EC2
- Nettoyage

Création d'un exemple d'application Amazon S3

Cet exemple d'application récupère un objet depuis Amazon S3. Pour exécuter l'application, vous avez besoin des éléments suivants :

- Un compartiment Amazon S3 contenant un fichier texte.
- AWS informations d'identification sur votre machine de développement qui vous permettent d'accéder au bucket.

Pour plus d'informations sur la création d'un compartiment Amazon S3 et le téléchargement d'un objet, consultez le guide de l'utilisateur d'Amazon Simple Storage Service. Pour plus d'informations sur les AWS informations d'identification, consultezAuthentifier le AWS SDK pour .NET avec AWS.

Créez un projet .NET Core avec le code suivant. Testez ensuite l'application sur votre machine de développement.

> **Note**
>
> Sur votre machine de développement, le .NET Core Runtime est installé, ce qui vous permet d'exécuter l'application sans la publier. Lorsque vous créerez une instance EC2 ultérieurement dans ce didacticiel, vous pouvez choisir d'installer le .NET Core Runtime sur l'instance. Cela vous donne une expérience similaire et un transfert de fichiers plus réduit. Toutefois, vous pouvez également choisir de ne pas installer le .NET Core Runtime sur l'instance. Si vous choisissez cette option, vous devez publier l'application afin que toutes les dépendances soient incluses lorsque vous la transférez vers l'instance.

Références du SDK

NuGet colis :

- AWSSDKS.3

Éléments de programmation :

- Espace de noms Amazon.S3

  Classe : Amazon S3 Client
- Espace de noms Amazon.S3.Model

  Classe GetObjectResponse

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
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.");
    }
 }


 // = = =
```

```
        // 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);
    }
  }

}
```

Si vous le souhaitez, vous pouvez supprimer temporairement les informations d'identification que vous utilisez sur votre machine de développement pour voir comment l'application répond. (Mais assurez-vous de restaurer les informations d'identification lorsque vous aurez terminé.)

Créer un rôle IAM

Créez un rôle IAM doté des autorisations appropriées pour accéder à Amazon S3.

1. Ouvrez la console IAM.
2. Dans le volet de navigation, choisissez Rôles, puis Create Role.
3. Sélectionnez le AWS service, recherchez et choisissez EC2, puis choisissez Next : Permissions.
4. Sous Joindre les politiques d'autorisation, recherchez et sélectionnez AmazonS3 ReadOnlyAccess. Passez en revue la politique si vous le souhaitez, puis choisissez Next : Tags.
5. Ajoutez des balises si vous le souhaitez, puis choisissez Suivant : Révision.
6. Tapez un nom et une description pour le rôle, puis choisissez Créer un rôle. Conservez bien ce nom, car il vous sera utile lorsque vous lancerez votre instance EC2.

Lancez une instance EC2 et attachez le rôle IAM

Lancez une instance EC2 avec le rôle IAM que vous avez créé précédemment. Vous pouvez le faire de différentes manières.

- Utilisation de la console EC2

  Pour lancer une instance à l'aide de la console EC2, consultez Lancer une instance à l'aide du nouvel assistant de lancement d'instance dans le guide de l'utilisateur Amazon EC2.

  Lorsque vous parcourez la page de lancement, vous devez au moins développer le volet Détails avancés afin de pouvoir spécifier le rôle IAM que vous avez créé précédemment dans le profil d'instance IAM.

- En utilisant le AWS SDK pour .NET

  Pour plus d'informations à ce sujetLancement d'une instance Amazon EC2, voir, y compris la Considérations supplémentaires fin de cette rubrique.

Pour lancer une instance EC2 à laquelle un rôle IAM est attaché, la configuration d'un utilisateur IAM doit inclure certaines autorisations. Pour plus d'informations sur les autorisations requises, consultez la section Accorder à un utilisateur l'autorisation de transmettre un rôle IAM à une instance dans le guide de l'utilisateur Amazon EC2.

Connectez-vous à l'instance EC2

Connectez-vous à l'instance EC2 afin de pouvoir y transférer l'exemple d'application, puis exécuter l'application. Vous aurez besoin du fichier contenant la partie privée de la paire de clés que vous avez utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations sur la connexion à une instance, consultez Connect to your Linux instance ou Connect to your Windows instance dans le guide de l'utilisateur Amazon EC2. Lorsque vous vous connectez, faites-le de manière à pouvoir transférer des fichiers de votre machine de développement vers votre instance.

Si vous utilisez Visual Studio sous Windows, vous pouvez également vous connecter à l'instance à l'aide du Toolkit for Visual Studio. Pour plus d'informations, consultez la section Connexion à une instance Amazon EC2 dans le guide de l' AWS Toolkit for Visual Studio utilisateur.

Exécutez l'exemple d'application sur l'instance EC2

1. Copiez les fichiers d'application de votre disque local vers votre instance.

   Les fichiers que vous transférez dépendent de la façon dont vous avez créé l'application et de l'installation du .NET Core Runtime sur votre instance. Pour plus d'informations sur le transfert de fichiers vers votre instance, consultez Connect to your Linux (voir la sous-section appropriée) ou Transférer des fichiers vers des instances Windows dans le guide de l'utilisateur Amazon EC2.
2. Lancez l'application et vérifiez qu'elle s'exécute avec les mêmes résultats que sur votre machine de développement.
3. Vérifiez que l'application utilise les informations d'identification fournies par le rôle IAM.

   a. Ouvrez la console Amazon EC2.

b. Sélectionnez l'instance et détachez le rôle IAM via Actions, Paramètres de l'instance, Attacher/remplacer le rôle IAM.

c. Exécutez à nouveau l'application et vérifiez qu'elle renvoie une erreur d'autorisation.

Nettoyage

Lorsque vous aurez terminé ce didacticiel, et si vous ne souhaitez plus utiliser l'instance EC2 que vous avez créée, veillez à mettre fin à l'instance pour éviter des coûts inutiles. Vous pouvez le faire dans la console Amazon EC2 ou par programmation, comme décrit dans. Interruption d'une instance Amazon EC2 Si vous le souhaitez, vous pouvez également supprimer d'autres ressources que vous avez créées pour ce didacticiel. Il peut s'agir d'un rôle IAM, d'une paire de clés EC2 et d'un fichier PEM, d'un groupe de sécurité, etc.

Utilisation du stockage sur Internet d'Amazon Simple Storage Service

Il AWS SDK pour .NET prend en charge Amazon S3, qui est un stockage pour Internet. Il est conçu pour faciliter l'informatique à l'échelle d'Internet pour les développeurs.

APIs

Les AWS SDK pour .NET offres APIs pour les clients Amazon S3. Ils vous APIs permettent de travailler avec des ressources Amazon S3 telles que des compartiments et des articles. Pour consulter l'ensemble complet APIs de Amazon S3, consultez ce qui suit :

- AWS SDK pour .NET Référence d'API (et faites défiler la page jusqu'à « Amazon.S3 »).
- Amazon.Extensions.S3. Documentation sur le chiffrement

Les Amazon S3 APIs sont fournis par les NuGet packages suivants :

- AWSSDKS.3
- Amazon Extensions.S3. Chiffrement

Conditions préalables

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Exemples présentés dans ce document

Les rubriques suivantes de ce document vous montrent comment utiliser le AWS SDK pour .NET pour travailler avec Amazon S3.

- Utilisation de clés KMS pour le chiffrement S3

Exemples dans d'autres documents

Les liens suivants vers le guide du développeur Amazon S3 fournissent des exemples supplémentaires de la façon d'utiliser le AWS SDK pour .NET pour travailler avec Amazon S3.

> **Note**
>
> Bien que ces exemples et considérations de programmation supplémentaires aient été créés pour la AWS SDK pour .NET version 3 du .NET Framework, ils sont également viables pour les versions ultérieures AWS SDK pour .NET de .NET Core. De petits ajustements dans le code sont parfois nécessaires.

Exemples de programmation Amazon S3

- Gérer ACLs
- Création d'un compartiment
- Chargement d'un objet
- Chargement en plusieurs parties avec l'API de haut niveau (Amazon.S3.Transfer). TransferUtility)
- Chargement partitionné avec l'API de bas niveau
- Liste des objets
- Etablissement de la liste des clés
- Obtention d'un objet
- Copie d'un objet
- Copie d'un objet grâce à l'API de chargement partitionné
- Suppression d'un objet
- Suppression de plusieurs objets
- Restauration d'un objet

- Configuration des notifications d'un compartiment
- Gestion du cycle de vie d'un objet
- Génération d'une URL d'objet présignée
- Gestion de sites web
- Activation du partage des ressources de plusieurs origines (CORS)

Considérations supplémentaires relatives à la programmation

- Utilisation du AWS SDK pour .NET pour la programmation d'Amazon S3
- Demandes grâce aux informations d'identification temporaires de l'utilisateur IAM
- Demandes grâce aux informations d'identification de sécurité temporaires de l'utilisateur fédéré
- Spécification du chiffrement côté serveur
- Spécification du chiffrement côté serveur avec clés de chiffrement fournies par le client

Utilisation de AWS KMS clés pour le chiffrement Amazon S3 dans AWS SDK pour .NET

Cet exemple montre comment utiliser des AWS Key Management Service clés pour chiffrer des objets Amazon S3. L'application crée une clé principale client (CMK) et l'utilise pour créer un objet AmazonS3 EncryptionClient V2 pour le chiffrement côté client. L'application utilise ce client pour créer un objet chiffré à partir d'un fichier texte donné dans un compartiment Amazon S3 existant. Il déchiffre ensuite l'objet et affiche son contenu.

> **Warning**
>
> Une classe similaire appelée `AmazonS3EncryptionClient` est obsolète et est moins sécurisée que la `AmazonS3EncryptionClientV2` classe. Pour migrer le code existant qui utilise`AmazonS3EncryptionClient`, voirMigration du client de chiffrement S3 (V1 vers V2).

Rubriques

- Création de matériel de chiffrement
- Création et chiffrement d'un objet Amazon S3
- Code complet

- Considérations supplémentaires

Création de matériel de chiffrement

L'extrait de code suivant crée un `EncryptionMaterials` objet contenant un ID de clé KMS.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

```
      // 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);
```

Création et chiffrement d'un objet Amazon S3

L'extrait de code suivant crée un `AmazonS3EncryptionClientV2` objet qui utilise les matériaux de chiffrement créés précédemment. Il utilise ensuite le client pour créer et chiffrer un nouvel objet Amazon S3.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- Amazon Extensions.S3. Chiffrement

Éléments de programmation :

- Espace de noms Amazon.Extensions.S3.Encryption

  Classe Amazon S3 V2 EncryptionClient

  Classe Amazon S3 V2 CryptoConfiguration

  Classe CryptoStorageMode

  Classe EncryptionMaterialsV2
- Espace de noms Amazon.Extensions.S3.Encryption.Primitives

  Classe KmsType
- Espace de noms Amazon.S3.Model

  Classe GetObjectRequest

  Classe GetObjectResponse

  Classe PutObjectRequest
- Espace de noms Amazon. KeyManagementService

Classe AmazonKeyManagementServiceClient

- Espace de noms Amazon. KeyManagementService.Modèle

  Classe CreateKeyRequest

  Classe CreateKeyResponse

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace KmsS3Encryption
{
  // = = =
```

```
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
      string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
      if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");
      if(!File.Exists(fileName))
        CommandLine.ErrorExit($"\nThe given file {fileName} doesn't exist.");
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

      // Create a customer master key (CMK) and store the result
      CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
      var kmsEncryptionContext = new Dictionary<string, string>();
      var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

      // Create the object in the bucket, then display the content of the object
      var putObjectResponse =
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
      Stream stream = putObjectResponse.ResponseStream;
      StreamReader reader = new StreamReader(stream);
      Console.WriteLine(reader.ReadToEnd());
    }


    //
    // Method to create and encrypt an object in an S3 bucket
    static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
      EncryptionMaterialsV2 materials, string bucketName,
      string fileName, string itemName)
    {
      // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
      var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

      // Create, encrypt, and put the object
```

```
      await s3EncClient.PutObjectAsync(new PutObjectRequest
      {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
      {
        BucketName = bucketName,
        Key = itemName
      });
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
    }

 }

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

}
```

Considérations supplémentaires

- Vous pouvez vérifier les résultats de cet exemple. Pour ce faire, accédez à la console Amazon S3 et ouvrez le compartiment que vous avez fourni à l'application. Recherchez ensuite le nouvel objet, téléchargez-le et ouvrez-le dans un éditeur de texte.

- La classe AmazonS3 EncryptionClient V2 implémente la même interface que la classe standard`AmazonS3Client`. Cela facilite le portage de votre code vers la `AmazonS3EncryptionClientV2` classe afin que le chiffrement et le déchiffrement se fassent automatiquement et de manière transparente dans le client.

- L'un des avantages de l'utilisation d'une AWS KMS clé comme clé principale est que vous n'avez pas besoin de stocker et de gérer vos propres clés principales ; cela se fait par AWS. Un deuxième avantage est que la `AmazonS3EncryptionClientV2` classe du AWS SDK pour .NET est interopérable avec la `AmazonS3EncryptionClientV2` classe du AWS SDK pour Java. Cela signifie que vous pouvez chiffrer avec le AWS SDK pour Java et déchiffrer avec le AWS SDK pour .NET, et vice versa.

> **Note**
>
> La `AmazonS3EncryptionClientV2` classe des AWS SDK pour .NET prend en charge les clés principales KMS uniquement lorsqu'elle est exécutée en mode métadonnées. Le mode de fichier d'instructions de la `AmazonS3EncryptionClientV2` classe du AWS SDK pour .NET est incompatible avec la `AmazonS3EncryptionClientV2` classe du AWS SDK pour Java.

- Pour plus d'informations sur le chiffrement côté client avec cette `AmazonS3EncryptionClientV2` classe et sur le fonctionnement du chiffrement d'enveloppe, consultez la section Chiffrement des données côté client avec Amazon AWS SDK pour .NET S3.

Envoyer des notifications depuis le cloud à l'aide d'Amazon Simple Notification Service

> **Note**
>
> Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Il AWS SDK pour .NET prend en charge Amazon Simple Notification Service (Amazon SNS), un service Web qui permet aux applications, aux utilisateurs finaux et aux appareils d'envoyer instantanément des notifications depuis le cloud. Pour plus d'informations, consultez Amazon SNS.

Répertorier vos rubriques Amazon SNS

L'exemple suivant montre comment répertorier vos rubriques Amazon SNS, les abonnements à chaque rubrique et les attributs de chaque rubrique. Cet exemple utilise la valeur par défaut 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));
```

Envoyer un message à une rubrique Amazon SNS

L'exemple suivant montre comment envoyer un message à une rubrique Amazon SNS. L'exemple prend un argument, l'ARN de la rubrique 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);
            }
        }
    }
}
```

Consultez l'exemple complet, y compris les informations sur la façon de créer et d'exécuter l'exemple à partir de la ligne de commande, sur GitHub.

Envoi d'un SMS à un numéro de téléphone

L'exemple suivant montre comment envoyer un SMS à un numéro de téléphone. L'exemple utilise un argument, le numéro de téléphone, qui doit être dans l'un des deux formats décrits dans les commentaires.

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

Consultez l'exemple complet, y compris les informations sur la façon de créer et d'exécuter l'exemple à partir de la ligne de commande, sur GitHub.

Messagerie à l'aide d'Amazon SQS

Il AWS SDK pour .NET prend en charge Amazon Simple Queue Service (Amazon SQS), un service de mise en file d'attente de messages qui gère les messages ou les flux de travail entre les composants d'un système.

Les files d'attente Amazon SQS fournissent un mécanisme qui vous permet d'envoyer, de stocker et de recevoir des messages entre des composants logiciels tels que des microservices, des systèmes distribués et des applications sans serveur. Cela vous permet de dissocier ces composants et vous évite de devoir concevoir et exploiter votre propre système de messagerie. Pour plus d'informations sur le fonctionnement des files d'attente et des messages dans Amazon SQS,

consultez les didacticiels Amazon SQS et l'architecture de base d'Amazon SQS dans le manuel Amazon Simple Queue Service Developer Guide.

> **Important**
>
> En raison de la nature distribuée des files d'attente, Amazon SQS ne peut garantir que vous recevrez les messages dans l'ordre précis dans lequel ils sont envoyés. Si vous devez préserver l'ordre des messages, utilisez une file d'attente FIFO Amazon SQS.

APIs

Les AWS SDK pour .NET offres APIs pour les clients Amazon SQS. Ils vous APIs permettent de travailler avec les fonctionnalités d'Amazon SQS telles que les files d'attente et les messages. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la référence de l'AWS SDK pour .NET API (et faites défiler la page jusqu'à « Amazon.sqs »).

Les Amazon SQS APIs sont fournis par le package AWSSDK NuGet .SQS.

Conditions préalables

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

Rubriques

- Création de files d'attente Amazon SQS
- Mise à jour des files d'attente Amazon SQS
- Suppression des files d'attente Amazon SQS
- Envoi de messages Amazon SQS
- Réception de messages Amazon SQS

Création de files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le pour AWS SDK pour .NET créer une file d'attente Amazon SQS. L'application crée une file d'attente contenant des lettres mortes si vous n'en

fournissez pas l'ARN. Il crée ensuite une file de messages standard, qui inclut une file de lettres mortes (celle que vous avez fournie ou celle qui a été créée).

Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement des informations sur toutes les files d'attente existantes.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Afficher les files d'attente existantes
- Création de la file d'attente
- Obtenir l'ARN d'une file d'attente
- Code complet
- Considérations supplémentaires

Afficher les files d'attente existantes

L'extrait suivant présente une liste des files d'attente existantes dans la région du client SQS et les attributs de chaque file d'attente.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Création de la file d'attente

L'extrait suivant crée une file d'attente. L'extrait inclut l'utilisation d'une file d'attente de lettres mortes, mais une file d'attente de lettres mortes n'est pas nécessairement requise pour vos files d'attente.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Obtenir l'ARN d'une file d'attente

L'extrait suivant obtient l'ARN de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.SQS

Éléments de programmation :

- Espace de noms Amazon.sqs

  Classe Amazon SQSClient

  Classe QueueAttributeName
- Espace de noms Amazon.SQS.Model

  Classe CreateQueueRequest

Classe CreateQueueResponse

Classe GetQueueAttributesResponse

Classe ListQueuesResponse

Le code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
 ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ShowQueues(sqsClient);
        return;
      }

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

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

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

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


    //
```

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


    //
    // Method to create a queue. Returns the queue URL.
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();

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

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

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

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


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


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

```
    }
  }


  // = = =
```

```
        {
          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);
    }
  }

}
```

Considérations supplémentaires

- Le nom de votre file d'attente doit être composé de caractères alphanumériques, de traits d'union et de traits de soulignement.

- Les noms et les files d'attente URLs font la distinction majuscules/minuscules

- Si vous avez besoin de l'URL de la file d'attente mais que vous n'avez que le nom de la file d'attente, utilisez l'une `AmazonSQSClient.GetQueueUrlAsync` des méthodes.

- Pour plus d'informations sur les différents attributs de file d'attente que vous pouvez définir, consultez CreateQueueRequestla référence d'AWS SDK pour .NET API ou SetQueueAttributesla référence d'API Amazon Simple Queue Service.

- Cet exemple indique une longue période d'interrogation pour tous les messages de la file d'attente que vous créez. Cela se fait à l'aide de l'`ReceiveMessageWaitTimeSeconds`attribut.

  Vous pouvez également spécifier un long sondage lors d'un appel aux `ReceiveMessageAsync` méthodes de la SQSClient classe Amazon. Pour de plus amples informations, veuillez consulter Réception de messages Amazon SQS.

  Pour plus d'informations sur les sondages courts par rapport aux sondages longs, consultez la section Sondage court et long du manuel Amazon Simple Queue Service Developer Guide.

- Une file d'attente de lettres mortes est une file que les autres files d'attente (source) peuvent cibler pour les messages qui ne sont pas traités correctement. Pour plus d'informations, consultez les files d'attente contenant des lettres mortes Amazon SQS dans le guide du développeur Amazon Simple Queue Service.

- Vous pouvez également consulter la liste des files d'attente et les résultats de cet exemple dans la console Amazon SQS.

Mise à jour des files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le pour mettre AWS SDK pour .NET à jour une file d'attente Amazon SQS. Après quelques vérifications, l'application met à jour l'attribut donné avec la valeur donnée, puis affiche tous les attributs de la file d'attente.

Si seule l'URL de la file d'attente est incluse dans les arguments de la ligne de commande, l'application affiche simplement tous les attributs de la file d'attente.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Afficher les attributs de file d'attente
- Valider le nom d'attribut
- Mettre à jour l'attribut de file
- Code complet
- Considérations supplémentaires

Afficher les attributs de file d'attente

L'extrait suivant montre les attributs de la file d'attente identifiés par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Valider le nom d'attribut

L'extrait suivant valide le nom de l'attribut mis à jour.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Mettre à jour l'attribut de file

L'extrait suivant met à jour un attribut de la file d'attente identifié par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.SQS

Éléments de programmation :

- Espace de noms Amazon.sqs

  Classe Amazon SQSClient

  Classe QueueAttributeName

- Espace de noms Amazon.SQS.Model

  Classe GetQueueAttributesResponse

Le code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

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

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

      // Otherwise, attempt to update the given queue attribute with the given value
      else
      {
        // Check to see if the attribute is valid
        if(ValidAttribute(attribute))
        {
          // Perform the update and then show all the attributes of the queue
          await UpdateAttribute(sqsClient, qUrl, attribute, value);
          await ShowAllAttributes(sqsClient, qUrl);
        }
        else
        {
          Console.WriteLine($"\nThe given attribute name, {attribute}, isn't valid.");
        }
      }
    }


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


    //
    // Method to check the name of the attribute
    private static bool ValidAttribute(string attribute)
    {
```

```
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }


    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
    }


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


  // = = =
```

```
    // 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);
    }
  }

}
```

Considérations supplémentaires

- Pour mettre à jour l'`RedrivePolicy`attribut, vous devez mettre la valeur entière entre guillemets et éviter les guillemets pour les key/value paires, en fonction de votre système d'exploitation.

  Sous Windows, par exemple, la valeur est construite de la manière suivante :

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

Suppression des files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le pour AWS SDK pour .NET supprimer une file d'attente Amazon SQS. L'application supprime la file d'attente, attend qu'elle disparaisse pendant un certain temps, puis affiche la liste des files d'attente restantes.

Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement une liste des files d'attente existantes.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Supprimer la file d'attente
- Attendez que la file d'attente soit terminée
- Afficher la liste des files d'attente existantes
- Code complet
- Considérations supplémentaires

Supprimer la file d'attente

L'extrait suivant supprime la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Attendez que la file d'attente soit terminée

L'extrait suivant attend la fin du processus de suppression, ce qui peut prendre 60 secondes.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Afficher la liste des files d'attente existantes

L'extrait suivant présente une liste des files d'attente existantes dans la région du client SQS.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.SQS

Éléments de programmation :

- Espace de noms Amazon.sqs

  Classe Amazon SQSClient
- Espace de noms Amazon.SQS.Model

  Classe ListQueuesResponse

Le code

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

namespace SQSDeleteQueue
{
  // = = =
```

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

Considérations supplémentaires

- L'appel `DeleteQueueAsync` d'API ne vérifie pas si la file d'attente que vous supprimez est utilisée comme file d'attente de lettres mortes. Une procédure plus sophistiquée pourrait vérifier cela.

- Vous pouvez également consulter la liste des files d'attente et les résultats de cet exemple dans la console Amazon SQS.

Envoi de messages Amazon SQS

Cet exemple vous montre comment utiliser le pour envoyer des messages AWS SDK pour .NET à une file d'attente Amazon SQS, que vous pouvez créer par programmation ou à l'aide de la console Amazon SQS. L'application envoie un seul message à la file d'attente, puis un lot de messages. L'application attend ensuite l'entrée de l'utilisateur, qui peut prendre la forme de messages supplémentaires à envoyer à la file d'attente ou d'une demande de sortie de l'application.

Cet exemple et le suivant concernant la réception de messages peuvent être utilisés conjointement pour voir le flux des messages dans Amazon SQS.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Envoyer un message
- Envoyer un lot de messages
- Supprimer tous les messages de la file d'attente
- Code complet
- Considérations supplémentaires

Envoyer un message

L'extrait suivant envoie un message à la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Envoyer un lot de messages

L'extrait suivant envoie un lot de messages à la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Supprimer tous les messages de la file d'attente

L'extrait suivant supprime tous les messages de la file d'attente identifiée par l'URL de file d'attente donnée. Cette opération est également connue sous le nom de purge de la file d'attente.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.SQS

Éléments de programmation :

- Espace de noms Amazon.sqs

  Classe Amazon SQSClient
- Espace de noms Amazon.SQS.Model

  Classe PurgeQueueResponse

  Classe SendMessageBatchResponse

  Classe SendMessageResponse

  Classe SendMessageBatchRequestEntry

  Classe SendMessageBatchResultEntry

Le code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

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

    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSSendMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

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

      // (could verify that the queue exists)
      // Send some example messages to the given queue
      // A single message
      await SendMessage(sqsClient, args[0], JsonMessage);

      // A batch of messages
      var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
      await SendMessageBatch(sqsClient, args[0], batchMessages);

      // Let the user send their own messages or quit
      await InteractWithUser(sqsClient, args[0]);

      // Delete all messages that are still in the queue
```

```
    await DeleteAllMessages(sqsClient, args[0]);
  }


  //
  // Method to put a message on a queue
  // Could be expanded to include message attributes, etc., in a SendMessageRequest
  private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
  {
    SendMessageResponse responseSendMsg =
      await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n  {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
  }


  //
  // Method to put a batch of messages on a queue
  // Could be expanded to include message attributes, etc.,
  // in the SendMessageBatchRequestEntry objects
  private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
  {
    Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
    SendMessageBatchResponse responseSendBatch =
      await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
      Console.WriteLine($"Message {entry.Id} successfully queued.");
  }


  //
  // Method to get input from the user
  // They can provide messages to put in the queue or exit the application
  private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
  {
    string response;
    while (true)
    {
      // Get the user's input
      Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
      response = Console.ReadLine();
```

```
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
      }
    }


    //
    // Method to delete all messages from the queue
    private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
    }
  }
}
```

Considérations supplémentaires

- Pour plus d'informations sur les différentes limitations relatives aux messages, y compris les caractères autorisés, consultez la section Quotas relatifs aux messages dans le manuel Amazon Simple Queue Service Developer Guide.

- Les messages restent dans les files d'attente jusqu'à ce qu'ils soient supprimés ou que la file d'attente soit purgée. Lorsqu'un message est reçu par une application, il n'est pas visible dans la file d'attente même s'il existe toujours dans la file d'attente. Pour plus d'informations sur les délais de visibilité, consultez Amazon SQS Visibility Timeout.

- Outre le corps du message, vous pouvez également ajouter des attributs aux messages. Pour plus d'informations, consultez la section Métadonnées des messages.

Réception de messages Amazon SQS

Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour recevoir des messages d'une file d'attente Amazon SQS, que vous pouvez créer par programmation ou à l'aide de la console Amazon SQS. L'application lit un seul message dans la file d'attente, traite le message (dans ce cas,

affiche le corps du message sur la console), puis le supprime de la file d'attente. L'application répète ces étapes jusqu'à ce que l'utilisateur tape une touche sur le clavier.

Cet exemple et l'exemple précédent concernant l'envoi de messages peuvent être utilisés conjointement pour voir le flux des messages dans Amazon SQS.

Les sections suivantes fournissent des extraits de cet exemple. Le code complet de l'exemple est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- Recevez un message
- Supprimer un message
- Code complet
- Considérations supplémentaires

Recevez un message

L'extrait suivant reçoit un message de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Supprimer un message

L'extrait suivant supprime un message de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple à la fin de cette rubrique montre cet extrait en cours d'utilisation.

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

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- AWSSDK.SQS

Éléments de programmation :

- Espace de noms Amazon.sqs

  Classe Amazon SQSClient
- Espace de noms Amazon.SQS.Model

  Classe ReceiveMessageRequest

  Classe ReceiveMessageResponse

Le code

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

namespace SQSReceiveMessages
{
  class Program
  {
```

```
    private const int MaxMessages = 1;
    private const int WaitTime = 2;
    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSReceiveMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

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

      // (could verify that the queue exists)
      // Read messages from the queue and perform appropriate actions
      Console.WriteLine($"Reading messages from queue\n  {args[0]}");
      Console.WriteLine("Press any key to stop. (Response might be slightly
delayed.)");
      do
      {
        var msg = await GetMessage(sqsClient, args[0], WaitTime);
        if(msg.Messages.Count != 0)
        {
          if(ProcessMessage(msg.Messages[0]))
            await DeleteMessage(sqsClient, msg.Messages[0], args[0]);
        }
      } while(!Console.KeyAvailable);
    }


    //
    // Method to read a message from the given queue
    // In this example, it gets one message at a time
    private static async Task<ReceiveMessageResponse> GetMessage(
      IAmazonSQS sqsClient, string qUrl, int waitTime=0)
    {
```

```
      return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
        QueueUrl=qUrl,
        MaxNumberOfMessages=MaxMessages,
        WaitTimeSeconds=waitTime
        // (Could also request attributes, set visibility timeout, etc.)
      });
    }


    //
    // Method to process a message
    // In this example, it simply prints the message
    private static bool ProcessMessage(Message message)
    {
      Console.WriteLine($"\nMessage body of {message.MessageId}:");
      Console.WriteLine($"{message.Body}");
      return true;
    }


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

Considérations supplémentaires

- Pour spécifier un long sondage, cet exemple utilise la `WaitTimeSeconds` propriété pour chaque appel à la `ReceiveMessageAsync` méthode.

  Vous pouvez également spécifier un long sondage pour tous les messages d'une file d'attente en utilisant l'`ReceiveMessageWaitTimeSeconds`attribut lors de la création ou de la mise à jour de la file d'attente.

  Pour plus d'informations sur les sondages courts par rapport aux sondages longs, consultez la section Sondage court et long du manuel Amazon Simple Queue Service Developer Guide.

- Pendant le traitement des messages, vous pouvez utiliser le descripteur de réception pour modifier le délai de visibilité des messages. Pour plus d'informations sur la procédure à suivre, consultez les `ChangeMessageVisibilityAsync` méthodes de la SQSClient classe Amazon.

- L'appel de la `DeleteMessageAsync` méthode supprime le message de la file d'attente de manière inconditionnelle, quel que soit le paramètre de délai d'expiration de visibilité.

Utilisation AWS Lambda pour le service de calcul

Les AWS SDK pour .NET supports AWS Lambda, qui vous permettent d'exécuter du code sans provisionner ni gérer de serveurs. Pour plus d'informations, consultez la page AWS Lambda du produit et le guide du AWS Lambda développeur, en particulier la section consacrée à l'utilisation de C#.

APIs

Le AWS SDK pour .NET APIs prévoit AWS Lambda. Ils vous APIs permettent d'utiliser les fonctionnalités Lambda telles que les fonctions, lesdéclencheurs et les événements. Pour voir l'ensemble complet de APIs, consultez Lambda dans la référence de l'AWS SDK pour .NET API.

Les Lambda APIs sont fournis par NuGet des packages.

Prérequis

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Informations supplémentaires

Consultez Intégration AWS à .NET Aspire pour plus d'informations sur le développement AWS Lambda avec .NET Aspire.

Rubriques

Rubriques

- Utiliser des annotations pour écrire des fonctions AWS Lambda

Utiliser des annotations pour écrire des fonctions AWS Lambda

Lorsque vous écrivez des fonctions Lambda, vous devez parfois écrire une grande quantité de code de gestionnaire et mettre à jour des AWS CloudFormation modèles, entre autres tâches. Lambda Annotations est un framework destiné à alléger ces charges pour les fonctions Lambda de .NET 6, rendant ainsi l'expérience d'écriture Lambda plus naturelle en C#.

À titre d'exemple des avantages de l'utilisation du framework Lambda Annotations, considérez les extraits de code suivants qui ajoutent deux nombres.

Sans annotations 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" } }
        };
    }
```

```
}
```

Avec des annotations Lambda

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

Comme le montre l'exemple, les annotations Lambda peuvent supprimer le besoin d'un certain code de plaque de cuisson.

Pour plus de détails sur l'utilisation du framework ainsi que des informations supplémentaires, consultez les ressources suivantes :

- Le GitHub fichier README pour la documentation sur les attributs APIs et les attributs des annotations Lambda.
- L'article de blog sur les annotations Lambda.
- Le package `Amazon.Lambda.Annotations` NuGet .
- Le projet Photo Asset Management sur GitHub. Plus précisément, consultez le PamApiAnnotationsdossier et les références aux annotations Lambda dans le fichier README du projet.

> **Note**
>
> L'exemple précédent est spécifique à la version 3 du AWS SDK pour .NET. Si vous utilisez l'exemple avec la version V4 du SDK (la dernière version), vous devrez peut-être effectuer des ajustements en fonction des informations contenues dansMigration vers la version 4.

Bibliothèques et frameworks de haut niveau pour AWS SDK pour .NET

Les sections suivantes contiennent des informations sur les bibliothèques et les frameworks de haut niveau qui ne font pas partie des fonctionnalités principales du SDK. Ces bibliothèques et frameworks utilisent les fonctionnalités de base du SDK pour créer des fonctionnalités qui facilitent certaines tâches.

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le Création d'une application simple sujet. Il vous présente le SDK.

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Rubriques

- AWS Framework de traitement des messages pour .NET
- Intégration AWS à .NET Aspire dans le AWS SDK pour .NET

AWS Framework de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET est un framework AWS natif qui simplifie le développement d'applications de traitement de messages .NET utilisant AWS des services tels qu'Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) et Amazon. EventBridge Le framework réduit la quantité de code standard que les développeurs doivent écrire, ce qui vous permet de vous concentrer sur votre logique métier lors de la publication et de la consommation de messages. Pour plus de détails sur la façon dont le framework peut simplifier votre développement, consultez le billet de blog Présentation du framework de traitement des AWS messages pour .NET (version préliminaire). La première partie en particulier fournit une démonstration qui montre la différence entre l'utilisation d'appels d'API de bas niveau et l'utilisation du framework.

Le cadre de traitement des messages prend en charge les activités et fonctionnalités suivantes :

- Envoi de messages à SQS et publication d'événements sur SNS et. EventBridge
- Réception et gestion des messages de SQS à l'aide d'un sondeur de longue durée, généralement utilisé dans les services d'arrière-plan. Cela inclut la gestion du délai de visibilité pendant le traitement d'un message afin d'empêcher d'autres clients de le traiter.

- Gestion des messages dans AWS Lambda les fonctions.
- FIFO (first-in-first-out), files d'attente SQS et sujets SNS.
- OpenTelemetry pour la journalisation.

Pour plus de détails sur ces activités et fonctionnalités, consultez la section Fonctionnalités du billet de blog et les sujets répertoriés ci-dessous.

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Ressources supplémentaires

- Le `AWS.Messaging`package sur NuGet.org.
- La référence de l'API.
- Le `README` fichier dans le GitHub dépôt à l'adresse https://github.com/aws/aws-dotnet-messaging/
- Injection de dépendances .NET par Microsoft.
- .NET Generic Host de Microsoft.

Rubriques

- Commencez avec le cadre de traitement des AWS messages pour .NET
- Publiez des messages avec le AWS cadre de traitement des messages pour .NET
- Consommez des messages avec le AWS cadre de traitement des messages pour .NET
- Utilisation du FIFO avec le cadre de traitement des AWS messages pour .NET
- Journalisation et télémétrie ouverte pour le cadre de traitement des AWS messages pour .NET
- Personnalisation de l'infrastructure de traitement des AWS messages pour .NET
- Sécurité de l'infrastructure de traitement des AWS messages pour .NET

Commencez avec le cadre de traitement des AWS messages pour .NET

Avant de commencer, assurez-vous d'avoir configuré votre environnement et votre projet. Consultez également les informations contenues dansUtilisation de l'SDK.

Cette rubrique fournit des informations qui vous aideront à commencer à utiliser le cadre de traitement des messages. Outre les informations relatives aux prérequis et à la configuration, un didacticiel est fourni pour vous montrer comment implémenter un scénario courant.

Prérequis et configuration

- Les informations d'identification que vous fournissez pour votre application doivent disposer des autorisations appropriées pour le service de messagerie et les opérations qu'il utilise. Pour plus d'informations, consultez les rubriques relatives à la sécurité pour SQS et SNS, ainsi que EventBridgedans leurs guides de développement respectifs. Consultez également la partie du fichier README GitHub qui traite des autorisations spécifiques.
- Pour utiliser le cadre de traitement des AWS messages pour .NET, vous devez ajouter le `AWS.Messaging` NuGetpackage à votre projet. Par exemple :

```
dotnet add package AWS.Messaging
```

- Le framework s'intègre au conteneur de services d'injection de dépendances (DI) de .NET. Vous pouvez configurer le framework lors du démarrage de votre application en appelant `AddAWSMessageBus` pour l'ajouter au conteneur 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");
});
```

didacticiel

Ce didacticiel explique comment utiliser le cadre de traitement des AWS messages pour .NET. Elle crée deux applications : une API ASP.NET Core Minimal qui envoie des messages à une file d'attente Amazon SQS lorsqu'elle reçoit une demande sur un point de terminaison d'API, et une application de console de longue durée qui interroge ces messages et les gère.

- Les instructions de ce didacticiel privilégient l'interface de ligne de commande .NET, mais vous pouvez exécuter ce didacticiel à l'aide d'outils multiplateformes tels que .NET CLI ou Microsoft Visual Studio. Pour plus d'informations sur les outils, consultezInstallation et configuration de votre chaîne d'outils pour AWS SDK pour .NET.

- Ce didacticiel part du principe que vous utilisez votre `[default]` profil comme identifiant. Cela suppose également que des informations d'identification à court terme sont disponibles avec les autorisations appropriées pour envoyer et recevoir des messages Amazon SQS. Pour plus d'informations, consultez Authentifier le AWS SDK pour .NET avec AWS les rubriques relatives à la sécurité relatives à SQS.

> **Note**
>
> L'exécution de ce didacticiel peut entraîner des coûts liés à la messagerie SQS.

Étapes

- Création d'une file d'attente SQS
- Création et exécution de l'application de publication
- Création et exécution de l'application de gestion
- Nettoyage

Création d'une file d'attente SQS

Ce didacticiel nécessite une file d'attente SQS pour envoyer et recevoir des messages. Une file d'attente peut être créée à l'aide de l'une des commandes suivantes pour le AWS CLI ou le Outils AWS pour PowerShell. Prenez note de l'URL de la file d'attente renvoyée afin de pouvoir la spécifier dans la configuration du framework qui suit.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Outils AWS pour PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Création et exécution de l'application de publication

Suivez la procédure ci-dessous pour créer et exécuter l'application de publication.

1. Ouvrez une invite de commande ou un terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

```
dotnet new webapi --name Publisher
```

3. Accédez au dossier du nouveau projet. Ajoutez une dépendance à l'infrastructure de traitement des AWS messages pour .NET.

```
cd Publisher
dotnet add package AWS.Messaging
```

> **Note**
>
> Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veillez également à ajouter `AWSSDK.SSO` et`AWSSDK.SSOOIDC`.

4. Remplacez le code `Program.cs` par le code suivant.

```
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. Exécutez la commande suivante. Cela devrait ouvrir une fenêtre de navigateur avec l'interface utilisateur Swagger, qui vous permettra d'explorer et de tester votre API.

```
dotnet watch run <queue URL created earlier>
```

6. Ouvrez le `/greeting` point de terminaison et choisissez Try it out.
7. Spécifiez `greeting` les valeurs `senderName` et les valeurs du message, puis choisissez Execute. Cela appelle votre API, qui envoie le message SQS.

Création et exécution de l'application de gestion

Utilisez la procédure suivante pour créer et exécuter l'application de gestion.

1. Ouvrez une invite de commande ou un terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

```
dotnet new console --name Handler
```

3. Accédez au dossier du nouveau projet. Ajoutez une dépendance à l'infrastructure de traitement des AWS messages pour .NET. Ajoutez également le `Microsoft.Extensions.Hosting` package, qui vous permet de configurer le framework via le .NET Generic Host.

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veillez également à ajouter `AWSSDK.SSO` et`AWSSDK.SSOOIDC`.

4. Remplacez le code `Program.cs` par le code suivant.

```
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. Exécutez la commande suivante. Cela lance un sondage de longue haleine.

```
dotnet run <queue URL created earlier>
```

Peu de temps après le démarrage, l'application recevra le message envoyé dans la première partie de ce didacticiel et enregistrera le message suivant :

```
Received message {greeting} from {senderName}
```

6. Appuyez `Ctrl+C` pour arrêter le sondeur.

Nettoyage

Utilisez l'une des commandes suivantes pour AWS CLI ou pour Outils AWS pour PowerShell supprimer la file d'attente.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Outils AWS pour PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publiez des messages avec le AWS cadre de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET permet de publier un ou plusieurs types de messages, de traiter un ou plusieurs types de messages ou d'effectuer les deux dans la même application.

Le code suivant montre la configuration d'une application qui publie différents types de messages vers différents AWS services.

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

Une fois que vous avez enregistré le framework au démarrage, injectez le générique `IMessagePublisher` dans votre code. Appelez sa `PublishAsync` méthode pour publier l'un des types de messages configurés ci-dessus. L'éditeur générique déterminera la destination vers laquelle acheminer le message en fonction de son type.

Dans l'exemple suivant, un contrôleur ASP.NET MVC reçoit à la fois `ChatMessage` des messages et des `OrderInfo` événements de la part des utilisateurs, puis les publie sur Amazon SQS et Amazon SNS respectivement. Les deux types de messages peuvent être publiés à l'aide de l'éditeur générique configuré ci-dessus.

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;

    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }
```

```
        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

Afin d'acheminer un message vers la logique de traitement appropriée, le framework utilise des métadonnées appelées identifiant de type de message. Par défaut, il s'agit du nom complet du type .NET du message, y compris son nom d'assembly. Si vous envoyez et gérez à la fois des messages, ce mécanisme fonctionne bien si vous partagez la définition des objets de vos messages entre différents projets. Toutefois, si les messages sont redéfinis dans différents espaces de noms, ou si vous échangez des messages avec d'autres frameworks ou langages de programmation, vous devrez peut-être remplacer l'identifiant du type de message.

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

```
});
```

Éditeurs spécifiques aux services

L'exemple ci-dessus utilise le génériqueIMessagePublisher, qui peut être publié sur n'importe quel AWS service pris en charge en fonction du type de message configuré. Le framework fournit également des éditeurs spécifiques aux services pour Amazon SQS, Amazon SNS et Amazon. EventBridge Ces éditeurs spécifiques présentent des options qui ne s'appliquent qu'à ce service et peuvent être injectées à l'aide des types ISQSPublisherISNSPublisher, etIEventBridgePublisher.

Par exemple, lorsque vous envoyez des messages à une file d'attente FIFO SQS, vous devez définir l'ID de groupe de messages approprié. Le code suivant montre à nouveau l'ChatMessageexemple, mais utilise maintenant un ISQSPublisher pour définir des options spécifiques à 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();
    }
}
```

La même chose peut être faite pour les réseaux sociaux et EventBridge, `IEventBridgePublisher` respectivement, en utilisant `ISNSPublisher` et.

```
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>
});
```

Par défaut, les messages d'un type donné sont envoyés à la destination configurée à l'avance. Cependant, vous pouvez modifier la destination d'un seul message en utilisant les éditeurs spécifiques au message. Vous pouvez également remplacer le AWS SDK pour .NET client sous-jacent utilisé pour publier le message, ce qui peut être utile dans les applications mutualisées où vous devez modifier les rôles ou les informations d'identification, en fonction de la destination.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consommez des messages avec le AWS cadre de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET vous permet de consommer des messages publiés à l'aide du framework ou de l'un des services de messagerie. Les messages peuvent être consommés de différentes manières, dont certaines sont décrites ci-dessous.

Gestionnaires de messages

Pour consommer des messages, implémentez un gestionnaire de messages utilisant l'`IMessageHandler`interface pour chaque type de message que vous souhaitez traiter. Le mappage entre les types de messages et les gestionnaires de messages est configuré au démarrage du projet.

```
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();
```

Le code suivant montre un exemple de gestionnaire de messages pour un `ChatMessage` message.

```
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'extérieur `MessageEnvelope` contient les métadonnées utilisées par le framework. Sa `message` propriété est le type de message (dans ce cas`ChatMessage`).

Vous pouvez revenir `MessageProcessStatus.Success()` pour indiquer que le message a été traité avec succès et que le framework supprimera le message de la file d'attente Amazon SQS. Au retour`MessageProcessStatus.Failed()`, le message restera dans la file d'attente où il pourra être traité à nouveau ou déplacé vers une file d'attente contenant des lettres mortes, si cela est configuré.

Gestion des messages dans le cadre d'un processus de longue haleine

Vous pouvez appeler `AddSQSPoller` avec une URL de file d'attente SQS pour démarrer une longue série `BackgroundService`qui interrogera continuellement la file d'attente et traitera les messages.

```
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();
```

Configuration du SQS Message Poller

Le sondeur de messages SQS peut être configuré par le `SQSMessagePollerOptions` lors de l'appel. `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- Le nombre maximum de messages de la file d'attente à traiter simultanément. La valeur par défaut est 10.
- `WaitTimeSeconds`- Durée (en secondes) pendant laquelle l'appel `ReceiveMessage` SQS attend l'arrivée d'un message dans la file d'attente avant de revenir. Si un message est disponible, l'appel revient plus tôt que prévu`WaitTimeSeconds`. La valeur par défaut est 20.

Gestion des délais de visibilité des messages

Les messages SQS ont un délai d'expiration de visibilité. Lorsqu'un client commence à traiter un message donné, celui-ci reste dans la file d'attente, mais il est caché aux autres consommateurs pour éviter de le traiter plusieurs fois. Si le message n'est pas traité et supprimé avant de redevenir visible, un autre consommateur peut tenter de traiter le même message.

Le framework suivra et tentera de prolonger le délai de visibilité pour les messages qu'il gère actuellement. Vous pouvez configurer ce comportement `SQSMessagePollerOptions` lors de l'appel`AddSQSPoller`.

- `VisibilityTimeout`- Durée en secondes pendant laquelle les messages reçus sont masqués lors des demandes de récupération ultérieures. La valeur par défaut est 30.
- `VisibilityTimeoutExtensionThreshold`- Lorsque le délai de visibilité d'un message est inférieur à ce nombre de secondes après son expiration, le cadre prolonge le délai de visibilité (de quelques secondes supplémentaires`VisibilityTimeout`). La valeur par défaut est 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Fréquence en secondes à laquelle le framework vérifiera les messages arrivés à expiration dans les `VisibilityTimeoutExtensionThreshold` secondes qui suivent leur expiration, puis prolongera leur délai de visibilité. La valeur par défaut est 1.

Dans l'exemple suivant, le framework vérifiera toutes les secondes les messages toujours en cours de traitement. Pour les messages redevenus visibles 5 secondes, le framework prolongera automatiquement le délai de visibilité de chaque message de 30 secondes supplémentaires.

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
 options =>
{
    options.VisibilityTimeout = 30;
    options.VisibilityTimeoutExtensionThreshold = 5;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

Gestion des messages dans les AWS Lambda fonctions

Vous pouvez utiliser le cadre de traitement des AWS messages pour .NET avec l'intégration de SQS à Lambda. Ceci est fourni par le `AWS.Messaging.Lambda` package. Reportez-vous à son fichier README pour commencer.

Utilisation du FIFO avec le cadre de traitement des AWS messages pour .NET

Pour les cas d'utilisation où l'ordre et la déduplication des messages sont essentiels, le cadre de traitement des AWS messages pour .NET prend en charge les files d'attente Amazon SQS first-in-first-out (FIFO) et les rubriques Amazon SNS.

Publication

Lorsque vous publiez des messages dans une file d'attente ou une rubrique FIFO, vous devez définir l'ID du groupe de messages, qui indique le groupe auquel appartient le message. Les messages

au sein d'un groupe sont traités dans l'ordre. Vous pouvez le définir sur les éditeurs de messages spécifiques à SQS et spécifiques au SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Abonnement en cours

Lors de la gestion des messages d'une file d'attente FIFO, le framework gère les messages d'un groupe de messages donné dans l'ordre dans lequel ils ont été reçus pour chaque `ReceiveMessages` appel. Le framework entre automatiquement dans ce mode de fonctionnement lorsqu'il est configuré avec une file d'attente se terminant par `.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();
```

Journalisation et télémétrie ouverte pour le cadre de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET est conçu OpenTelemetry pour enregistrer les traces de chaque message publié ou traité par le framework. Ceci est fourni par le `AWS.Messaging.Telemetry.OpenTelemetry`package. Reportez-vous à son fichier README pour commencer.

> **Note**
>
> Pour obtenir des informations de sécurité relatives à la journalisation, consultezSécurité de l'infrastructure de traitement des AWS messages pour .NET.

Personnalisation de l'infrastructure de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET crée, envoie et gère les messages dans trois « couches » différentes :

1. Au niveau de la couche la plus externe, le framework crée la requête ou la réponse AWS-native spécifique à un service. Avec Amazon SQS, par exemple, il crée des `SendMessage`requêtes et travaille avec les `Message`objets définis par le service.
2. Dans la demande et la réponse SQS, le framework définit l'`MessageBody`élément (ou `Message` pour Amazon SNS ou `Detail` pour EventBridge Amazon) au format JSON. CloudEvent Il contient les métadonnées définies par le framework qui sont accessibles sur l'`MessageEnvelope`objet lors de la gestion d'un message.
3. Au niveau de la couche la plus interne, l'`data`attribut à l'intérieur de l'objet CloudEvent JSON contient une sérialisation JSON de l'objet .NET envoyé ou reçu sous forme de message.

```
{
    "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>"
}
```

Vous pouvez personnaliser la façon dont l'enveloppe du message est configurée et lue :

- `"id"`identifie le message de manière unique. Par défaut, il est défini sur un nouveau GUID, mais celui-ci peut être remplacé en implémentant le vôtre `IMessageIdGenerator` et en l'injectant dans le conteneur DI.
- `"type"`contrôle la manière dont le message est acheminé vers les gestionnaires. Par défaut, le nom complet du type .NET correspondant au message est utilisé. Vous pouvez le remplacer via le

`messageTypeIdentifier` paramètre lorsque vous mappez le type de message à la destination via `AddSQSPublisher`AddSNSPublisher, ou`AddEventBridgePublisher`.

- `"source"`indique le système ou le serveur qui a envoyé le message.
  - Il s'agira du nom de la fonction en cas de publication depuis AWS Lambda, du nom du cluster et de l'ARN de la tâche si vous utilisez Amazon ECS, de l'ID de l'instance si vous publiez depuis Amazon EC2, sinon d'une valeur de remplacement de`/aws/messaging`.
  - Vous pouvez annuler cela via `AddMessageSource` ou `AddMessageSourceSuffix` sur le`MessageBusBuilder`.
- `"time"`réglé sur le courant DateTime en UTC. Cela peut être contourné en implémentant le vôtre `IDateTimeHandler` et en l'injectant dans le conteneur DI.
- `"data"`contient une représentation JSON de l'objet .NET envoyé ou reçu sous forme de message :
  - `ConfigureSerializationOptions`on vous `MessageBusBuilder` permet de configurer le `System.Text.Json.JsonSerializerOptions`qui sera utilisé lors de la sérialisation et de la désérialisation du message.
  - Pour injecter des attributs supplémentaires ou transformer l'enveloppe du message une fois que le framework l'a créée, vous pouvez l'implémenter `ISerializationCallback` et l'enregistrer via `AddSerializationCallback` on`MessageBusBuilder`.

Sécurité de l'infrastructure de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET repose sur le AWS SDK pour .NET pour communiquer avec AWS. Pour plus d'informations sur la sécurité dans le AWS SDK pour .NET, voirSécurité de ce AWS produit ou service.

Pour des raisons de sécurité, le framework n'enregistre pas les messages de données envoyés par l'utilisateur. Si vous souhaitez activer cette fonctionnalité à des fins de débogage, vous devez appeler `EnableDataMessageLogging()` le bus de messages comme suit :

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Si vous découvrez un problème de sécurité potentiel, reportez-vous à la politique de sécurité pour obtenir des informations de rapport.

Intégration AWS à .NET Aspire dans le AWS SDK pour .NET

.NET Aspire est une nouvelle façon de créer des applications prêtes pour le cloud. En particulier, il fournit une orchestration pour les environnements locaux dans lesquels exécuter, connecter et déboguer les composants des applications distribuées. Pour améliorer la boucle de développement interne des applications prêtes pour le cloud, des intégrations avec .NET Aspire ont été créées pour connecter vos applications .NET aux AWS ressources. Ces intégrations sont disponibles via le package NuGet Aspire.Hosting.aws.

Les intégrations .NET Aspire suivantes sont disponibles :

- La capacité de fournir vos AWS ressources par le biais de CloudFormation. Cette intégration est utilisée dans le AppHost projet .NET Aspire.

  Pour plus d'informations, consultez le billet de blog Integrating AWS with .NET Aspire.
- Installation, configuration et connexion du AWS SDK pour .NET à Amazon DynamoDB local. Cette intégration est utilisée dans le AppHost projet .NET Aspire.

  Pour plus d'informations, consultez le billet de blog Integrating AWS with .NET Aspire.
- Activez un environnement de développement local pour les AWS Lambdafonctions. Cette intégration est utilisée dans le AppHost projet .NET Aspire.

  Pour plus d'informations, consultez les articles de blog Création et débogage d'applications .NET Lambda avec .NET Aspire (partie 1) et Création et débogage d'applications .NET Lambda avec .NET Aspire (partie 2).

  > **Note**
  >
  > Ceci est une documentation préliminaire pour une fonctionnalité en version de prévisualisation. Elle est susceptible d'être modifiée.

  Comme cette fonctionnalité est en version préliminaire, vous devrez vous inscrire pour les fonctionnalités de prévisualisation. Pour plus d'informations sur cette fonctionnalité de prévisualisation et sur la procédure à suivre pour s'y inscrire, consultez le problème du suivi du développement sur GitHub.

Informations supplémentaires

Pour plus d'informations et de détails sur la façon dont vous pouvez utiliser les intégrations disponibles dans Aspire.Hosting.aws, consultez les ressources suivantes.

- Article de blog Intégration AWS avec .NET Aspire.
- Articles de blog Création et débogage d'applications .NET Lambda avec .NET Aspire (partie 1) et Création et débogage d'applications .NET Lambda avec .NET Aspire (partie 2).
- Le dépôt integrations-on-dotnet-aspire-for-aws est activé. GitHub
- Le fichier README détaillé pour le package NuGet Aspire.Hosting.aws.

Support pour d'autres AWS services et configurations

Les AWS services de AWS SDK pour .NET soutien s'ajoutent à ceux décrits dans les sections précédentes. Pour plus d'informations sur tous les services pris en charge, consultez la référence des AWS SDK pour .NET API. APIs

Outre les espaces de noms pour les AWS services individuels, il fournit AWS SDK pour .NET également les éléments suivants : APIs

Area	Description	Ressources
AWS Support	Accès programmatique aux dossiers de AWS Support et aux fonctionnalités de Trusted Advisor.	Voir Amazon. AWSSupport et Amazon. AWSSupport.Modèle.
Général	Énumérations et classes d'assistance.	Consultez Amazon et Amazon.Util.

SDK pour .NET exemples de code (v4)

Les exemples de code présentés dans cette rubrique vous montrent comment utiliser le AWS SDK pour .NET (v4) avec AWS.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Certains services contiennent des exemples de catégories supplémentaires qui montrent comment tirer parti des bibliothèques ou des fonctions spécifiques au service.

Services

- Exemples d'Aurora utilisant SDK pour .NET (v4)
- Exemples d'Auto Scaling utilisant SDK pour .NET (v4)
- Exemples d'utilisation d'Amazon Bedrock SDK pour .NET (v4)
- Exemples d'Amazon Bedrock Runtime utilisant SDK pour .NET (v4)
- CloudFormation exemples utilisant SDK pour .NET (v4)
- CloudWatch exemples utilisant SDK pour .NET (v4)
- CloudWatch Exemples de logs utilisant SDK pour .NET (v4)
- Exemples de fournisseurs d'identité Amazon Cognito utilisant SDK pour .NET (v4)
- AWS Control Tower exemples utilisant SDK pour .NET (v4)
- Exemples SDK pour .NET DynamoDB utilisant (v4)
- Exemples d'Amazon EC2 utilisant SDK pour .NET (v4)
- Exemples d'Amazon ECS utilisant SDK pour .NET (v4)
- AWS IoT exemples utilisant SDK pour .NET (v4)
- AWS IoT data exemples utilisant SDK pour .NET (v4)
- Exemples d'utilisation d'Amazon Redshift SDK pour .NET (v4)

- Exemples d'Amazon S3 utilisant SDK pour .NET (v4)

Exemples d'Aurora utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Aurora.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour Aurora

L'exemple de code suivant montre comment démarrer avec Aurora.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Pour plus de détails sur l'API, voir Description DBClusters dans le manuel de référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un groupe de paramètres pour le cluster de bases de données Aurora personnalisé et définissez des valeurs pour les paramètres.
- Créez un cluster de bases de données qui utilise le groupe de paramètres.
- Créez une instance de base de données qui contient une base de données.
- Prenez un instantané du cluster de bases de données, puis nettoyez les ressources.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;
```

```
/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using the
 DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group using
 the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the DescribeDBClusterParametersAsync
 method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
 method.
    13. Wait for the DB instance to be ready using the DescribeDBInstances method.
    14. Display the connection endpoint string for the new DB cluster.
    15. Create a snapshot of the DB cluster using the CreateDBClusterSnapshotAsync
 method.
    16. Wait for DB snapshot to be ready using the DescribeDBClusterSnapshotsAsync
 method.
    17. Delete the DB instance using the DeleteDBInstanceAsync method.
    18. Delete the DB cluster using the DeleteDBClusterAsync method.
    19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync methods.
```

```
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
       Console.WriteLine(
           "Welcome to the Amazon Aurora: get started with DB clusters example.");
       Console.WriteLine(sepBar);

       DBClusterParameterGroup parameterGroup = null!;
       DBCluster? newCluster = null;
       DBInstance? newInstance = null;

       try
       {
           var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();
```

```
            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName,
parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);
```

```
            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family by
entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
```

```
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
 DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParameterG

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroupAsync(string
 parameterGroupName, List<string>? parameterNames = null)
    {
```

```
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
```

```
   /// Describe the user source parameters in the group.
   /// </summary>
   /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
   /// <returns>Async task.</returns>
   public static async Task DescribeUserSourceParameters(string parameterGroupName)
   {
       Console.WriteLine(sepBar);
       Console.WriteLine("7. Describe updated user source parameters in the
group.");

       var parameters =
           await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName, "user");

       parameters.ForEach(p =>
           Console.WriteLine(
               $"\n\tParameter: {p.ParameterName}." +
               $"\n\tDescription: {p.Description}." +
               $"\n\tAllowed Values: {p.AllowedValues}." +
               $"\n\tValue: {p.ParameterValue}."));

       Console.WriteLine(sepBar);
   }

   /// <summary>
   /// Choose a DB engine version.
   /// </summary>
   /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
   /// <returns>The selected engine version.</returns>
   public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
   {
       Console.WriteLine(sepBar);
       // Get a list of allowed engines.
       var allowedEngines =
           await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

       Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
       int i = 1;
       foreach (var version in allowedEngines)
       {
```

```
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
```

```
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
```

```
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
 a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
```

```
   /// <param name="instanceIdentifier">Instance identifier to use for the DB
instance.</param>
   /// <returns>The new DB instance.</returns>
   public static async Task<DBInstance?> CreateNewInstance(
       string clusterIdentifier,
       string engineName,
       string engineVersion,
       string instanceClass,
       string instanceIdentifier)
   {
       Console.WriteLine(sepBar);
       Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
       bool isInstanceReady = false;
       DBInstance newInstance;
       var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
       isInstanceReady = instances.FirstOrDefault(i =>
           i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

       if (isInstanceReady)
       {
           Console.WriteLine("Instance already created.");
           newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
       }
       else
       {
           newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
               clusterIdentifier,
               instanceIdentifier,
               engineName,
               engineVersion,
               instanceClass
           );

           Console.WriteLine("13. Waiting for DB instance to be ready...");
           while (!isInstanceReady)
           {
               Console.Write(".");
               Thread.Sleep(5000);
               instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
```

```
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;
```

```
        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }
```

```
        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
```

```
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Méthodes d'encapsulation appelées par le scénario pour gérer les actions Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
```

```
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
```

```
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
```

```
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
```

```
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
```

```
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
```

```
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
```

```
   public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
   {
       var results = new List<DBClusterSnapshot>();

       DescribeDBClusterSnapshotsResponse response;
       DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
       {
           DBClusterIdentifier = dbClusterIdentifier
       };
       // Get the full list if there are multiple pages.
       do
       {
           response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
           results.AddRange(response.DBClusterSnapshots);
           request.Marker = response.Marker;
       }
       while (response.Marker is not null);
       return results;
   }

   /// <summary>
   /// Delete a particular DB cluster.
   /// </summary>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <returns>DB cluster object.</returns>
   public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
   {
       var response = await _amazonRDS.DeleteDBClusterAsync(
           new DeleteDBClusterRequest()
           {
               DBClusterIdentifier = dbClusterIdentifier,
               SkipFinalSnapshot = true
           });

       return response.DBCluster;
   }

   /// <summary>
   /// Delete a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
  - CréerDBCluster
  - CréerDBClusterParameterGroup
  - Créer un DBCluster instantané
  - CréerDBInstance
  - SuppressionDBCluster
  - SuppressionDBClusterParameterGroup
  - SuppressionDBInstance
  - DécrireDBClusterParameterGroups
  - Décrire DBCluster les paramètres
  - Décrire les DBCluster instantanés
  - DécrireDBClusters
  - Décrire DBEngine les versions
  - DécrireDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Actions

`CreateDBCluster`

L'exemple de code suivant montre comment utiliser`CreateDBCluster`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, consultez Create DBCluster in AWS SDK pour .NET API Reference.

`CreateDBClusterParameterGroup`

L'exemple de code suivant montre comment utiliser`CreateDBClusterParameterGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, consultez Create DBCluster ParameterGroup in AWS SDK pour .NET API Reference.

`CreateDBClusterSnapshot`

L'exemple de code suivant montre comment utiliser`CreateDBClusterSnapshot`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
```

```
    }
```

- Pour plus de détails sur l'API, consultez la section Créer un DBCluster instantané dans le manuel de référence des AWS SDK pour .NET API.

`CreateDBInstance`

L'exemple de code suivant montre comment utiliser`CreateDBInstance`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, consultez Create DBInstance in AWS SDK pour .NET API Reference.

`DeleteDBCluster`

L'exemple de code suivant montre comment utiliser`DeleteDBCluster`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, voir Supprimer DBCluster dans le manuel de référence des AWS SDK pour .NET API.

`DeleteDBClusterParameterGroup`

L'exemple de code suivant montre comment utiliser`DeleteDBClusterParameterGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Pour plus de détails sur l'API, voir Supprimer DBCluster ParameterGroup dans le manuel de référence des AWS SDK pour .NET API.

`DeleteDBInstance`

L'exemple de code suivant montre comment utiliser`DeleteDBInstance`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, voir Supprimer DBInstance dans le manuel de référence des AWS SDK pour .NET API.

`DescribeDBClusterParameterGroups`

L'exemple de code suivant montre comment utiliser`DescribeDBClusterParameterGroups`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
    }
```

- Pour plus de détails sur l'API, voir Description DBCluster ParameterGroups dans le manuel de référence des AWS SDK pour .NET API.

`DescribeDBClusterParameters`

L'exemple de code suivant montre comment utiliser`DescribeDBClusterParameters`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, consultez la section Décrire DBCluster les paramètres dans le AWS SDK pour .NET manuel de référence des API.

`DescribeDBClusterSnapshots`

L'exemple de code suivant montre comment utiliser`DescribeDBClusterSnapshots`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, consultez la section Décrire les DBCluster instantanés dans le manuel de référence des AWS SDK pour .NET API.

`DescribeDBClusters`

L'exemple de code suivant montre comment utiliser`DescribeDBClusters`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Pour plus de détails sur l'API, voir Description DBClusters dans le manuel de référence des AWS SDK pour .NET API.

`DescribeDBEngineVersions`

L'exemple de code suivant montre comment utiliser`DescribeDBEngineVersions`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, consultez la section Description DBEngine des versions dans le manuel de référence des AWS SDK pour .NET API.

`DescribeDBInstances`

L'exemple de code suivant montre comment utiliser`DescribeDBInstances`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, voir Description DBInstances dans le manuel de référence des AWS SDK pour .NET API.

`DescribeOrderableDBInstanceOptions`

L'exemple de code suivant montre comment utiliser`DescribeOrderableDBInstanceOptions`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, consultez la section DescribeOrderableDBInstanceOptions du manuel de référence des AWS SDK pour .NET API.

`ModifyDBClusterParameterGroup`

L'exemple de code suivant montre comment utiliser`ModifyDBClusterParameterGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.GetValueOrDefault() && p.DataType == "integer")
           {
               while (newValue == 0)
               {
                   Console.WriteLine(
                       $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                   var choice = Console.ReadLine();
                   int.TryParse(choice, out newValue);
               }

               p.ParameterValue = newValue.ToString();
           }
       }
```

```
        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Pour plus de détails sur l'API, voir Modifier DBCluster ParameterGroup dans le manuel de référence des AWS SDK pour .NET API.

Exemples d'Auto Scaling utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Auto Scaling.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour Auto Scaling

L'exemple de code suivant montre comment commencer à utiliser Auto Scaling.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();

        if (response.AutoScalingGroups == null || response.AutoScalingGroups.Count
 == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
            return;
        }
```

```
        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availability
        });

    }
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeAutoScalingGroupsà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer un groupe Amazon EC2 Auto Scaling avec un modèle de lancement et des zones de disponibilité, et obtenir des informations sur l'exécution des instances ;
- Activez la collecte CloudWatch de métriques Amazon.
- mettre à jour la capacité souhaitée du groupe et attendre qu'une instance démarre ;
- résilier la plus ancienne instance du groupe ;
- répertorier les activités de mise à l'échelle qui se produisent en réponse aux demandes des utilisateurs et aux modifications de capacité ;
- Obtenez des statistiques pour CloudWatch les métriques, puis nettoyez les ressources.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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[0].ZoneName);

        // Keep checking the details of the new group until its lifecycle state
```

```
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);

        Console.WriteLine($"Auto scaling group {groupName} successfully created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);
```

```
        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {
```

```
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are {activity.Details}");
            });
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
        if (metrics.Any())
        {
            Console.WriteLine($"Metrics collected for {groupName}:");
            metrics.ForEach(metric =>
            {
                Console.Write($"Metric name: {metric.MetricName}\t");
                Console.WriteLine($"Namespace: {metric.Namespace}");
            });
        }

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        if (dataPoints.Any())
        {
            Console.WriteLine("Details for the metrics collected:");
            dataPoints.ForEach(detail => { Console.WriteLine(detail); });
        }

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for {groupName}.");
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
```

```
        Console.WriteLine("Now terminating all instances in the Auto Scaling
 group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
 autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
 ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
```

```
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
 the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
 group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;
```

```
        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }
```

```
    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Définissez des fonctions appelées par le scénario pour gérer les modèles et les métriques de lancement. Ces fonctions englobent Auto Scaling, Amazon EC2 et CloudWatch les actions.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
 client.</param>
```

```
    public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
    {
        _amazonAutoScaling = amazonAutoScaling;
    }


    /// <summary>
    /// Create a new Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name to use for the new Auto Scaling
    /// group.</param>
    /// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
    /// launch template to use to create instances in the group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }



    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
 response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
 response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };
```

```
        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }



    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };
```

```
            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groups = new List<AutoScalingGroup>();
        var groupList = new List<string>
            {
                groupName,
            };

        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        if (response.AutoScalingGroups != null)
        {
            groups = response.AutoScalingGroups;
        }

        return groups;
    }
```

```
    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }


    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };
```

```
        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
```

```
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
 _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;
```

```
using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }
```

```
    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }
```

```
        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
```

```
    /// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
    public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
    {
        var metrics = new List<Metric>();
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };

        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);
        if (response.Metrics != null)
        {
            metrics = response.Metrics;
        }
        return metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var dataPoints = new List<Datapoint>();
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
```

```
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
        if (response.Datapoints != null)
        {
            dataPoints = response.Datapoints;
        }

        return dataPoints;
    }

}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Actions

`CreateAutoScalingGroup`

L'exemple de code suivant montre comment utiliser`CreateAutoScalingGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateAutoScalingGroupà la section Référence des AWS SDK pour .NET API.

`DescribeAutoScalingGroups`

L'exemple de code suivant montre comment utiliser`DescribeAutoScalingGroups`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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>();
       var instanceDetails = new List<AutoScalingInstanceDetails>();
       if (groups != null)
       {
           groups.ForEach(group =>
           {
               if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
               {
                   group.Instances.ForEach(instance =>
                   {
                       instanceIds.Add(instance.InstanceId);
                   });
               }
           });

           var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
           {
               MaxRecords = 10,
               InstanceIds = instanceIds,
           };

           var response =
               await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                   scalingGroupsRequest);
           if (response.AutoScalingInstances != null)
           {
               instanceDetails = response.AutoScalingInstances;
           }
       }

       return instanceDetails;
   }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAutoScalingGroupsà la section Référence des AWS SDK pour .NET API.

`DescribeAutoScalingInstances`

L'exemple de code suivant montre comment utiliser`DescribeAutoScalingInstances`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });
```

```
            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAutoScalingInstancesà la section Référence des AWS SDK pour .NET API.

`DescribeScalingActivities`

L'exemple de code suivant montre comment utiliser`DescribeScalingActivities`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeScalingActivitiesà la section Référence des AWS SDK pour .NET API.

`DisableMetricsCollection`

L'exemple de code suivant montre comment utiliser`DisableMetricsCollection`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous DisableMetricsCollectionà la section Référence des AWS SDK pour .NET API.

`EnableMetricsCollection`

L'exemple de code suivant montre comment utiliser`EnableMetricsCollection`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous EnableMetricsCollectionà la section Référence des AWS SDK pour .NET API.

`SetDesiredCapacity`

L'exemple de code suivant montre comment utiliser`SetDesiredCapacity`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous SetDesiredCapacityà la section Référence des AWS SDK pour .NET API.

`TerminateInstanceInAutoScalingGroup`

L'exemple de code suivant montre comment utiliser`TerminateInstanceInAutoScalingGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous TerminateInstanceInAutoScalingGroupà la section Référence des AWS SDK pour .NET API.

`UpdateAutoScalingGroup`

L'exemple de code suivant montre comment utiliser`UpdateAutoScalingGroup`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateAutoScalingGroupà la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation d'Amazon Bedrock SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon Bedrock.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Actions

Mise en route

Bonjour Amazon Bedrock

L'exemple de code suivant montre comment commencer à utiliser Amazon Bedrock.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace BedrockActions;
```

```
/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
 of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-
is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);

    }


    /// <summary>
    /// List foundation models.
    /// </summary>
    /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
    private static async Task ListFoundationModelsAsync(AmazonBedrockClient
 bedrockClient)
    {
        Console.WriteLine("List foundation models with no filter.");

        try
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
 ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
```

```
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }


    /// <summary>
    /// Write the foundation model summary to console.
    /// </summary>
    /// <param name="foundationModel"> The foundation model summary to write to
 console. </param>
    private static void WriteToConsole(FoundationModelSummary foundationModel)
    {
        Console.WriteLine($"{foundationModel.ModelId}, Customization:
 {string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
 {foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
 ", foundationModel.InputModalities)}, Output: {string.Join(", ",
 foundationModel.OutputModalities)}");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK pour .NET API.

Actions

`ListFoundationModels`

L'exemple de code suivant montre comment utiliser`ListFoundationModels`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez les modèles de fondation Bedrock disponibles.

```
   /// <summary>
   /// List foundation models.
   /// </summary>
   /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
   private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
   {
       Console.WriteLine("List foundation models with no filter.");

       try
       {
           var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
           {
           });

           if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
           {
               foreach (var fm in response.ModelSummaries)
               {
                   WriteToConsole(fm);
               }
           }
           else
           {
               Console.WriteLine("Something wrong happened");
           }
       }
       catch (AmazonBedrockException e)
       {
           Console.WriteLine(e.Message);
       }
```

```
    }
```

- Pour plus de détails sur l'API, reportez-vous ListFoundationModelsà la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon Bedrock Runtime utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon Bedrock Runtime.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Anthropic Claude

Converse

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock.

```
// Use the Converse API to send a text message to Anthropic Claude.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);
```

```
    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour .NET .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Anthropic Claude à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// Use the Converse API to send a text message to Anthropic Claude
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
```

```
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour .NET API.

InvokeModel

L'exemple de code suivant montre comment envoyer un message texte à Anthropic Claude à l'aide de l'API Invoke Model.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Utilisez l'API Invoke Model pour envoyer un message texte.

```
// Use the native inference API to send a text message to Anthropic Claude.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
```

```
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous InvokeModelà la section Référence des AWS SDK pour .NET API.

Cohere Command

Converse

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command à l'aide de l'API Converse de Bedrock.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Cohere Command à l'aide de l'API Converse de Bedrock.

```
// Use the Converse API to send a text message to Cohere Command.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
```

```
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour .NET .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Cohere Command à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
```

```
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour .NET API.

Meta Llama

Converse

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama à l'aide de l'API Converse de Bedrock.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Meta Llama à l'aide de l'API Converse de Bedrock.

```
// Use the Converse API to send a text message to Meta Llama.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
```

```
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour .NET .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Meta Llama à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// Use the Converse API to send a text message to Meta Llama
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
```

```
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour .NET API.

Mistral AI

Converse

L'exemple de code suivant montre comment envoyer un message texte à Mistral à l'aide de l'API Converse de Bedrock.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Mistral à l'aide de l'API Converse de Bedrock.

```
// Use the Converse API to send a text message to Mistral.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, consultez Converse dans la Référence des API du kit AWS SDK pour .NET .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Mistral à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message texte à Mistral à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// Use the Converse API to send a text message to Mistral
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
```

```
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous ConverseStreamà la section Référence des AWS SDK pour .NET API.

CloudFormation exemples utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec CloudFormation.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route

Mise en route

Bonjour CloudFormation

L'exemple de code suivant montre comment démarrer avec CloudFormation.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
```

```
                    new DescribeStacksRequest());
            if (paginatorForDescribeStacks.Stacks != null)
            {
                await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
                {
                    // Basic information for each stack
                    Console.WriteLine(
                        "\n-----------------------------------------------");
                    Console.WriteLine($"\nStack: {stack.StackName}");
                    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                    Console.WriteLine($"  Created: {stack.CreationTime}");

                    // The tags of each stack (etc.)
                    if (stack.Tags != null && stack.Tags.Count > 0)
                    {
                        Console.WriteLine("  Tags:");
                        foreach (Tag tag in stack.Tags)
                            Console.WriteLine($"    {tag.Key}, {tag.Value}");
                    }

                    // The resources of each stack
                    DescribeStackResourcesResponse responseDescribeResources =
                        await _amazonCloudFormation.DescribeStackResourcesAsync(
                            new DescribeStackResourcesRequest
                            {
                                StackName = stack.StackName
                            });
                    if (responseDescribeResources.StackResources != null &&
responseDescribeResources.StackResources.Count > 0)
                    {
                        Console.WriteLine("  Resources:");
                        foreach (StackResource resource in responseDescribeResources
                                    .StackResources)
                            Console.WriteLine(
                                $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                    }
                }
            }

            Console.WriteLine("\n-----------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
```

```
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeStackResourcesà la section Référence des AWS SDK pour .NET API.

CloudWatch exemples utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec CloudWatch.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour CloudWatch

L'exemple de code suivant montre comment démarrer avec CloudWatch.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m =>
 $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListMetricsà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- CloudWatch Répertoriez les espaces de noms et les métriques.
- obtenir les statistiques d'une métrique et de la facturation estimée ;
- créer et mettre à jour un tableau de bord ;
- créer et ajouter des données à une métrique ;
- créer et déclencher une alerte, puis consulter l'historique des alertes ;
- créer un détecteur d'anomalies ;
- obtenez une image de métrique, puis nettoyer les ressources.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics
```

```
    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */

    private static ILogger logger = null!;
    private static CloudWatchWrapper _cloudWatchWrapper = null!;
    private static IConfiguration _configuration = null!;
    private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
    private static SingleMetricAnomalyDetector? anomalyDetector = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
            .AddTransient<CloudWatchWrapper>()
        )
        .Build();

        _configuration = new ConfigurationBuilder()
```

```
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }
```

```
    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");
```

```
        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
```

```
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
```

```
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
```

```
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
```

```
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
```

```
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }
```

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
                { new() { customMetricNamespace, customMetricName } },
            View = "timeSeries",
            Region = "us-east-1",
            Stat = "Sum",
            Period = 86400,
            YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
            Title = "Custom Metric Widget",
            LiveData = true,
            Sparkline = true,
            Trend = true,
            Stacked = false,
            SetPeriodToTimeRange = false
        }
    });

    var newDashboardString = JsonSerializer.Serialize(newDashboard,
        new JsonSerializerOptions
```

```
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);
```

```
        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
```

```
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
```

```
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
```

```
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
```

```
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Méthodes d'encapsulation utilisées par le scénario pour les CloudWatch actions.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }
```

```
   /// <summary>
   /// List metrics available, optionally within a namespace.
   /// </summary>
   /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
   /// <param name="filter">Optional dimension filter.</param>
   /// <param name="metricName">Optional metric name filter.</param>
   /// <returns>The list of metrics.</returns>
   public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
   {
       var results = new List<Metric>();
       var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
           new ListMetricsRequest
           {
               Namespace = metricNamespace,
               Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
               MetricName = metricName
           });
       // Get the entire list using the paginator.
       await foreach (var metric in paginateMetrics.Metrics)
       {
           results.Add(metric);
       }

       return results;
   }

   /// <summary>
   /// Wrapper to get statistics for a specific CloudWatch metric.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricName">The name of the metric.</param>
   /// <param name="statistics">The list of statistics to include.</param>
   /// <param name="dimensions">The list of dimensions to include.</param>
   /// <param name="days">The number of days in the past to include.</param>
   /// <param name="period">The period for the data.</param>
   /// <returns>A list of DataPoint objects for the statistics.</returns>
   public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
       string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
   {
```

```
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
```

```
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
```

```
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
```

```
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
```

```
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
```

```
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
```

```
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
```

```
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
```

```
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
```

```
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });
```

```
        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Exemples de valeurs settings.json pour le scénario.

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
 example code.\n"
```

```
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
            "left": {
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
        "type": "metric",
        "properties": {
          "metrics": [
```

```
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
 "Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      }
    ]
  }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors

- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

Actions

`DeleteAlarms`

L'exemple de code suivant montre comment utiliser`DeleteAlarms`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteAlarmsà la section Référence des AWS SDK pour .NET API.

`DeleteAnomalyDetector`

L'exemple de code suivant montre comment utiliser`DeleteAnomalyDetector`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous DeleteAnomalyDetectorà la section Référence des AWS SDK pour .NET API.

`DeleteDashboards`

L'exemple de code suivant montre comment utiliser`DeleteDashboards`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteDashboardsà la section Référence des AWS SDK pour .NET API.

`DescribeAlarmHistory`

L'exemple de code suivant montre comment utiliser`DescribeAlarmHistory`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAlarmHistoryà la section Référence des AWS SDK pour .NET API.

`DescribeAlarms`

L'exemple de code suivant montre comment utiliser`DescribeAlarms`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAlarmsà la section Référence des AWS SDK pour .NET API.

`DescribeAlarmsForMetric`

L'exemple de code suivant montre comment utiliser`DescribeAlarmsForMetric`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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 ?? new List<MetricAlarm>();
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAlarmsForMetricà la section Référence des AWS SDK pour .NET API.

`DescribeAnomalyDetectors`

L'exemple de code suivant montre comment utiliser`DescribeAnomalyDetectors`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous DescribeAnomalyDetectorsà la section Référence des AWS SDK pour .NET API.

`DisableAlarmActions`

L'exemple de code suivant montre comment utiliser`DisableAlarmActions`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous DisableAlarmActionsà la section Référence des AWS SDK pour .NET API.

`EnableAlarmActions`

L'exemple de code suivant montre comment utiliser`EnableAlarmActions`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous EnableAlarmActionsà la section Référence des AWS SDK pour .NET API.

`GetDashboard`

L'exemple de code suivant montre comment utiliser`GetDashboard`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous GetDashboardà la section Référence des AWS SDK pour .NET API.

`GetMetricData`

L'exemple de code suivant montre comment utiliser`GetMetricData`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- Pour plus de détails sur l'API, reportez-vous GetMetricDataà la section Référence des AWS SDK pour .NET API.

`GetMetricStatistics`

L'exemple de code suivant montre comment utiliser`GetMetricStatistics`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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 ?? new List<Datapoint>();
    }
```

- Pour plus de détails sur l'API, reportez-vous GetMetricStatisticsà la section Référence des AWS SDK pour .NET API.

`GetMetricWidgetImage`

L'exemple de code suivant montre comment utiliser`GetMetricWidgetImage`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous GetMetricWidgetImageà la section Référence des AWS SDK pour .NET API.

`ListDashboards`

L'exemple de code suivant montre comment utiliser`ListDashboards`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous ListDashboardsà la section Référence des AWS SDK pour .NET API.

`ListMetrics`

L'exemple de code suivant montre comment utiliser`ListMetrics`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
```

```
    }
```

- Pour plus de détails sur l'API, reportez-vous ListMetricsà la section Référence des AWS SDK pour .NET API.

`PutAnomalyDetector`

L'exemple de code suivant montre comment utiliser`PutAnomalyDetector`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
   }
```

- Pour plus de détails sur l'API, reportez-vous PutAnomalyDetectorà la section Référence des AWS SDK pour .NET API.

`PutDashboard`

L'exemple de code suivant montre comment utiliser`PutDashboard`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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 ?? new
 List<DashboardValidationMessage>();
    }
```

- Pour plus de détails sur l'API, reportez-vous PutDashboardà la section Référence des AWS SDK pour .NET API.

`PutMetricAlarm`

L'exemple de code suivant montre comment utiliser`PutMetricAlarm`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous PutMetricAlarmà la section Référence des AWS SDK pour .NET API.

`PutMetricData`

L'exemple de code suivant montre comment utiliser`PutMetricData`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous PutMetricDataà la section Référence des AWS SDK pour .NET API.

CloudWatch Exemples de logs utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour .NET version (v4) avec CloudWatch des journaux.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions
- Scénarios

Actions

`GetQueryResults`

L'exemple de code suivant montre comment utiliser`GetQueryResults`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Gets the results of a CloudWatch Logs Insights query.
/// </summary>
/// <param name="queryId">The ID of the query.</param>
/// <returns>The query results response.</returns>
public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
{
    try
    {
        var request = new GetQueryResultsRequest
        {
            QueryId = queryId
        };
```

```
            var response = await
_amazonCloudWatchLogs.GetQueryResultsAsync(request);
            return response;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while getting query results:
{ex.Message}");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetQueryResultsà la section Référence des AWS SDK pour .NET API.

`StartQuery`

L'exemple de code suivant montre comment utiliser`StartQuery`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Starts a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
epoch).</param>
```

```
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
 {ex.Message}");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous StartQueryà la section Référence des AWS SDK pour .NET API.

Scénarios

Exécution d'une requête volumineuse

L'exemple de code suivant montre comment utiliser CloudWatch Logs pour interroger plus de 10 000 enregistrements.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Il s'agit du flux de travail principal qui illustre le scénario de requêtes volumineuses.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:
```

```
    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
```

```
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();

            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }
```

```
        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
```

```
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
```

```
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);

                        Console.WriteLine("Application preparation complete.");
                        return true;
                    }
                }
            }
            else
            {
                _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
                _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

                var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
                var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

                _queryStartDate = startDateMs / 1000;
                _queryEndDate = endDateMs / 1000;

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }

        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName)
```

```
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
```

```
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
```

```
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
```

```
                    logEvents.Add(new InputLogEvent
                    {
                        Timestamp =
 DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }

                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
```

```
    /// Executes the large query workflow.
    /// </summary>
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
        if (queryLimit > 10000) queryLimit = 10000;

        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
```

```
        }
        Console.WriteLine();

        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }
```

```
        var startDate =
 DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
```

```
        {
            offsetLastLogTime++; // Move to the next full second
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
```

```
        var results1 = await PerformLargeQuery(logGroupName, queryString,
range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");

        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
```

```
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }
```

```
        _logger.LogError("Timed out waiting for query results.");
        return null;
    }

    /// <summary>
    /// Cleans up the resources created during the scenario.
    /// </summary>
    public static async Task<bool> Cleanup()
    {
        var cleanup = !_interactive || GetYesNoResponse(
            "Do you want to delete the CloudFormation stack and all resources? (y/n)
");

        if (cleanup)
        {
            try
            {
                var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
                return stackDeleteSuccess;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
                return false;
            }
        }

        Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
        _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
        return true;
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
    {
        var request = new DeleteStackRequest
        {
            StackName = stackName,
```

```
        };

        if (forceDelete)
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };
```

```
            try
            {
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }

                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
```

```
        }
        Console.WriteLine("\rWait complete.                       ");
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
```

```
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
```

```
        return 0;
    }

    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
 FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - GetQueryResults

- StartQuery

Exemples de fournisseurs d'identité Amazon Cognito utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du fournisseur d'identité AWS SDK pour .NET (v4) avec Amazon Cognito.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`ListUserPools`

L'exemple de code suivant montre comment utiliser`ListUserPools`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, reportez-vous ListUserPoolsà la section Référence des AWS SDK pour .NET API.

AWS Control Tower exemples utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec AWS Control Tower.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour AWS Control Tower

L'exemple de code suivant montre comment démarrer avec AWS Control Tower.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloControlTower>();

        var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

        Console.Clear();
        Console.WriteLine("Hello, AWS Control Tower! Let's list available
baselines:");
        Console.WriteLine();

        var baselines = new List<BaselineSummary>();

        try
        {
            var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
            foreach (var baseline in baselines)
            {
                Console.WriteLine($"\t{baseline.Name}");
            }
        }
        catch (Amazon.ControlTower.Model.AccessDeniedException)
        {
            Console.WriteLine("Access denied. Please ensure you have the necessary
permissions.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous ListBaselinesà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- répertorier les zones de destination ;
- répertorier, activer, obtenir, réinitialiser et désactiver les références ;
- répertorier, activer, obtenir et désactiver les contrôles.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif illustrant AWS Control Tower les fonctionnalités.

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
```

```
namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
```

```
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
 example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
 Tower for landing zones,");
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
 GetCallerIdentityRequest())).Account;
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
 zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
 these operations in the demo.");
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }
```

```
            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;

            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
```

```
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
                                    alreadyEnabled = true;
                                    Console.WriteLine("No change, the selected
baseline was already enabled.");
                                    break;
                                }
                            }
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
```

```
                                    // Re-enable the Control Tower baseline if it
 was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "5.0",
 icBaselineArn ?? "");
                                }
                            }
                        }
                    }
                }
            }

            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
 {controls[i].Arn}");
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
```

```
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");

                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
 wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
 {disableOpId}");
                            }
                        }
                    }
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
 scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
```

```
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;

        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
"Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
```

```
    /// Gets user confirmation by waiting for input or returning true if not
 interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
 false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
}
```

Méthodes d'encapsulation appelées par le scénario pour gérer les actions Aurora.

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
```

```
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());
```

```
            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
```

```
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
```

```
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };
```

```
            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
```

```
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
```

```
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
 operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
 _controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };
```

```
            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
_controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
```

```
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

```
   /// <summary>
   /// Disable a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to disable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
```

```
            Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
```

```
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();

            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
        catch (AmazonControlCatalogException ex)
        {
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }

}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation

- ListBaselines
- ListEnabledBaselines
- ListEnabledControls
- ListLandingZones
- ResetEnabledBaseline

Actions

`DisableBaseline`

L'exemple de code suivant montre comment utiliser`DisableBaseline`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Disable a baseline for a specific target and wait for the operation to
complete.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
   /// <returns>The operation ID or null if there was a conflict.</returns>
   public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new DisableBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await _controlTowerService.DisableBaselineAsync(request);
           var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DisableBaselineà la section Référence des AWS SDK pour .NET API.

`DisableControl`

L'exemple de code suivant montre comment utiliser`DisableControl`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Disable a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to disable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }
```

```
            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable control. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DisableControlà la section Référence des AWS SDK pour .NET API.

`EnableBaseline`

L'exemple de code suivant montre comment utiliser`EnableBaseline`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
```

```
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
```

```
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous EnableBaselineà la section Référence des AWS SDK pour .NET API.

`EnableControl`

L'exemple de code suivant montre comment utiliser`EnableControl`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
```

```
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
```

```
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous EnableControlà la section Référence des AWS SDK pour .NET API.

`GetBaselineOperation`

L'exemple de code suivant montre comment utiliser`GetBaselineOperation`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
```

```
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetBaselineOperationà la section Référence des AWS SDK pour .NET API.

`GetControlOperation`

L'exemple de code suivant montre comment utiliser`GetControlOperation`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
```

```
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetControlOperationà la section Référence des AWS SDK pour .NET API.

`ListBaselines`

L'exemple de code suivant montre comment utiliser`ListBaselines`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// List all baselines.
```

```
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListBaselinesà la section Référence des AWS SDK pour .NET API.

`ListEnabledBaselines`

L'exemple de code suivant montre comment utiliser`ListEnabledBaselines`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListEnabledBaselinesà la section Référence des AWS SDK pour .NET API.

`ListEnabledControls`

L'exemple de code suivant montre comment utiliser`ListEnabledControls`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
 (ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
 enabling controls.");
            return new List<EnabledControlSummary>();
```

```
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListEnabledControlsà la section Référence des AWS SDK pour .NET API.

`ListLandingZones`

L'exemple de code suivant montre comment utiliser`ListLandingZones`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
```

```
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListLandingZonesà la section Référence des AWS SDK pour .NET API.

`ResetEnabledBaseline`

L'exemple de code suivant montre comment utiliser`ResetEnabledBaseline`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
```

```
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ResetEnabledBaselineà la section Référence des AWS SDK pour .NET API.

Exemples SDK pour .NET DynamoDB utilisant (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec DynamoDB.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Hello DynamoDB

L'exemple de code suivant montre comment démarrer avec DynamoDB.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;
```

```
namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListTablesà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une table pouvant contenir des données vidéo.
- Insérer, récupérez et mettez à jour un seul film dans la table.
- Écrivez des données vidéo dans la table à partir d'un exemple de fichier JSON.
- Recherchez les films sortis au cours d'une année donnée.
- Recherchez les films sortis au cours d'une plage d'années spécifique.
- Supprimez un film de la table, puis supprimez la table.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";
```

```
        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous" +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
```

```
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
```

```
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
```

```
        }

        Console.WriteLine("The DynamoDB Basics example application is complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using DynamoDB
with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
        WaitForEnter();
    }

    /// <summary>
```

```
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Utilisez le client injecté pour les opérations sur les tables.

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

Crée un tableau contenant des données vidéo.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });
```

```
            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
```

```
    }
```

Ajoute un seul film à la table.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

Met à jour un seul élément d'une table.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
```

```
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

Extrait un seul élément de la table des films.

```
   /// <summary>
   /// Gets information about an existing movie from the table.
   /// </summary>
   /// <param name="newMovie">A Movie object containing information about
   /// the movie to retrieve.</param>
   /// <param name="tableName">The name of the table containing the movie.</param>
   /// <returns>A Dictionary object containing information about the item
   /// retrieved.</returns>
   public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };

           var request = new GetItemRequest
           {
               Key = key,
               TableName = tableName,
           };

           var response = await _amazonDynamoDB.GetItemAsync(request);
           return response.Item;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return new Dictionary<string, AttributeValue>();
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
```

```
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Écrit un lot d'éléments dans la table des films.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
```

```
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Supprime un seul élément d'une table.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

Interroge la table des films sortis au cours d'une année donnée.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);
```

```
            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

Recherche dans la table les films sortis au cours d'une plage d'années spécifique.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
```

```
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Supprime la table des films.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
```

```
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Interrogation
- Analyser
- UpdateItem

Actions

`BatchWriteItem`

L'exemple de code suivant montre comment utiliser`BatchWriteItem`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Écrit un lot d'éléments dans la table des films.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
```

```
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();
```

```
            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous BatchWriteItemà la section Référence des AWS SDK pour .NET API.

`CreateTable`

L'exemple de code suivant montre comment utiliser`CreateTable`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
```

```
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateTableà la section Référence des AWS SDK pour .NET API.

`DeleteItem`

L'exemple de code suivant montre comment utiliser`DeleteItem`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
```

```
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteItemà la section Référence des AWS SDK pour .NET API.

`DeleteTable`

L'exemple de code suivant montre comment utiliser`DeleteTable`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteTableà la section Référence des AWS SDK pour .NET API.

`GetItem`

L'exemple de code suivant montre comment utiliser`GetItem`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
```

```
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetItemà la section Référence des AWS SDK pour .NET API.

`PutItem`

L'exemple de code suivant montre comment utiliser`PutItem`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous PutItemà la section Référence des AWS SDK pour .NET API.

`Query`

L'exemple de code suivant montre comment utiliser`Query`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
```

```
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
```

```
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus d'informations sur l'API, consultez Requête dans la référence d'API AWS SDK pour .NET .

`Scan`

L'exemple de code suivant montre comment utiliser`Scan`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
```

```
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, consultez Scan dans la Référence des API du kit AWS SDK pour .NET .

`UpdateItem`

L'exemple de code suivant montre comment utiliser`UpdateItem`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
```

```
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateItemà la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon EC2 utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon EC2.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route

Mise en route

Hello Amazon EC2

L'exemple de code suivant montre comment commencer à utiliser Amazon EC2.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
 EC2).
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                .AddTransient<EC2Wrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);
```

```
            await foreach (var securityGroup in
 paginatorForSecurityGroups.SecurityGroups)
            {
                securityGroups.Add(securityGroup);
            }

            // Now print the security groups returned by the call to
            // DescribeSecurityGroupsAsync.
            Console.WriteLine("Welcome to the EC2 Hello Service example. " +
                              "\nLet's list your Security Groups:");
            securityGroups.ForEach(group =>
            {
                Console.WriteLine(
                    $"Security group: {group.GroupName} ID: {group.GroupId}");
            });
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"An Amazon EC2 service error occurred while listing
 security groups. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing security groups.
 {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DescribeSecurityGroupsà la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon ECS utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon ECS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route

Mise en route

Bonjour Amazon ECS

L'exemple de code suivant montre comment faire ses premiers pas avec Amazon ECS.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon ECS client.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonECS>()
            )
            .Build();

        var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your account");
        Console.WriteLine();

        var clusters = new List<string>();

        var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
 ListClustersRequest());

        await foreach (var response in clustersPaginator.Responses)
        {
            clusters.AddRange(response.ClusterArns);
        }

        if (clusters.Count > 0)
        {
            clusters.ForEach(cluster =>
            {
                Console.WriteLine($"\tARN: {cluster}");
                Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
                Console.WriteLine();
            });
        }
        else
        {
            Console.WriteLine("No clusters were found.");
        }

    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListClustersà la section Référence des AWS SDK pour .NET API.

AWS IoT exemples utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec AWS IoT.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour AWS IoT

L'exemple de code suivant montre comment démarrer avec AWS IoT.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
```

```
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
```

```
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
 type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, consultez listThings dans la Référence des API du kit AWS SDK pour .NET .

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un AWS IoT objet.
- générer un certificat d'appareil ;
- Mettez à jour un AWS IoT objet avec des attributs.
- renvoyer un point de terminaison unique ;
- Listez vos AWS IoT certificats.
- Mettez à jour une AWS IoT ombre.
- rédiger les informations d'état ;
- créer une règle ;
- répertorier vos règles ;
- rechercher des objets en utilisant le nom de l'objet ;
- Supprimer un AWS IoT objet.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif illustrant AWS IoT les fonctionnalités.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
```

```
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();
```

```
        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;
```

```
        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
 AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
```

```
            Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
```

```
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
                    {
                        Console.WriteLine("Certificate attached to Thing
successfully.");
```

```
                }
                else
                {
                    Console.WriteLine("Failed to attach certificate to Thing.");
                }

                Console.WriteLine("Thing Details:");
                Console.WriteLine($"Thing Name: {thingName}");
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var attributes = new Dictionary<string, string>
        {
            { "Location", "Seattle" },
            { "DeviceType", "Sensor" },
            { "Firmware", "1.2.3" }
        };

        await iotWrapper.UpdateThingAsync(thingName, attributes);
        Console.WriteLine("Thing attributes updated successfully.");
        Console.WriteLine(new string('-', 80));

        // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();
```

```
            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
```

```
                            ruleName = userRuleName;
                    }
                    else
                    {
                        Console.WriteLine($"Using default rule name:
{ruleName}");
                    }

                    // Now create the IoT rule with the CloudFormation outputs
                    var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                    if (ruleResult)
                    {
                        Console.WriteLine("IoT Rule created successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Failed to create IoT rule.");
                    }
                }
                else
                {
                    Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
            }
        }
        else
        {
            Console.WriteLine("Skipping CloudFormation stack deployment and rule
creation.");
        }
        Console.WriteLine(new string('-', 80));

        // Step 10: List your rules
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("10. List your rules.");
        if (IsInteractive)
        {
```

```
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
```

```
                {
                    Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
```

```
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
{_stackName}");
                }
                Console.WriteLine(new string('-', 80));
```

```
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
 cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
 cloudFormationClient, scenarioLogger);
```

```
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

                if (stackCreated)
```

```
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
```

```
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
```

```
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
```

```
   /// <returns>True if the stack was deleted successfully.</returns>
   private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       try
       {
           var request = new DeleteStackRequest
           {
               StackName = stackName
           };

           await cloudFormationClient.DeleteStackAsync(request);
           Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

           bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

           if (stackDeleted)
           {
               Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
               return true;
           }
           else
           {
               scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
               return false;
           }
       }
       catch (Exception ex)
       {
           scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
           return false;
       }
   }

   /// <summary>
   /// Waits for the stack to be deleted.
   /// </summary>
   /// <param name="stackName">The name of the CloudFormation stack.</param>
   /// <param name="cloudFormationClient">The CloudFormation client.</param>
```

```
   /// <param name="scenarioLogger">The logger.</param>
   /// <returns>True if the stack was deleted successfully.</returns>
   private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       int retryCount = 0;
       const int maxRetries = 30;
       const int retryDelay = 10000;

       while (retryCount < maxRetries)
       {
           var describeStacksRequest = new DescribeStacksRequest
           {
               StackName = stackName
           };

           try
           {
               var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

               if (describeStacksResponse.Stacks.Count == 0 ||
                   describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
               {
                   return true;
               }
           }
           catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
           {
               return true;
           }

           Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
           await Task.Delay(retryDelay);
           retryCount++;
       }

       scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
       return false;
   }
```

```
    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
 for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

Une classe wrapper pour les méthodes du AWS IoT SDK.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
```

```
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
```

```
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
```

```
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
```

```
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
 {response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
```

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
```

```
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                        RuleDisabled = false
                    }
                };

                await _amazonIoT.CreateTopicRuleAsync(request);
                _logger.LogInformation($"Created IoT rule {ruleName}");
                return true;
            }
            catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
            {
                _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
                return false;
            }
            catch (Exception ex)
            {
                _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
                return false;
            }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
```

```
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
```

```
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
```

```
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
```

```
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
```

```
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
```

```
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
```

```
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
 page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Actions

`AttachThingPrincipal`

L'exemple de code suivant montre comment utiliser`AttachThingPrincipal`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Attaches a certificate to an IoT Thing.
```

```
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.AttachThingPrincipalAsync(request);
           _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
           return false;
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous AttachThingPrincipalà la section Référence des AWS SDK pour .NET API.

`CreateKeysAndCertificate`

L'exemple de code suivant montre comment utiliser`CreateKeysAndCertificate`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Creates a device certificate for AWS IoT.
   /// </summary>
   /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
           var request = new CreateKeysAndCertificateRequest
           {
               SetAsActive = true
           };

           var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
           _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
           return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
       {
           _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
           return null;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
           return null;
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous CreateKeysAndCertificateà la section Référence des AWS SDK pour .NET API.

`CreateThing`

L'exemple de code suivant montre comment utiliser`CreateThing`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateThingà la section Référence des AWS SDK pour .NET API.

`CreateTopicRule`

L'exemple de code suivant montre comment utiliser`CreateTopicRule`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
```

```
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                        RuleDisabled = false
                    }
                };

                await _amazonIoT.CreateTopicRuleAsync(request);
                _logger.LogInformation($"Created IoT rule {ruleName}");
                return true;
            }
            catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
            {
                _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
                return false;
            }
            catch (Exception ex)
            {
                _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
                return false;
            }
        }
```

- Pour plus de détails sur l'API, reportez-vous CreateTopicRuleà la section Référence des AWS SDK pour .NET API.

`DeleteCertificate`

L'exemple de code suivant montre comment utiliser`DeleteCertificate`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteCertificateà la section Référence des AWS SDK pour .NET API.

`DeleteThing`

L'exemple de code suivant montre comment utiliser`DeleteThing`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
```

```
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteThingà la section Référence des AWS SDK pour .NET API.

`DescribeEndpoint`

L'exemple de code suivant montre comment utiliser`DescribeEndpoint`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };
```

```
            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeEndpointà la section Référence des AWS SDK pour .NET API.

`DetachThingPrincipal`

L'exemple de code suivant montre comment utiliser`DetachThingPrincipal`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
```

```
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new DetachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.DetachThingPrincipalAsync(request);
           _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
           return false;
       }
   }
```

- Pour plus de détails sur l'API, reportez-vous DetachThingPrincipalà la section Référence des AWS SDK pour .NET API.

`ListCertificates`

L'exemple de code suivant montre comment utiliser`ListCertificates`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListCertificatesà la section Référence des AWS SDK pour .NET API.

`ListThings`

L'exemple de code suivant montre comment utiliser`ListThings`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
```

```
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListThingsà la section Référence des AWS SDK pour .NET API.

`SearchIndex`

L'exemple de code suivant montre comment utiliser`SearchIndex`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
```

```
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
```

```
   /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
   /// </summary>
   /// <param name="queryString">The search query string.</param>
   /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
   private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
   {
       try
       {
           // Update indexing configuration to REGISTRY mode
           _logger.LogInformation("Setting up IoT search indexing
configuration...");
           await _amazonIoT.UpdateIndexingConfigurationAsync(
               new UpdateIndexingConfigurationRequest()
               {
                   ThingIndexingConfiguration = new ThingIndexingConfiguration()
                   {
                       ThingIndexingMode = ThingIndexingMode.REGISTRY
                   }
               });

           _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

           // Wait for the index to be set up - this can take some time
           const int maxRetries = 10;
           const int retryDelaySeconds = 10;

           for (int attempt = 1; attempt <= maxRetries; attempt++)
           {
               try
               {
                   _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                   await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                   // Try to get the current indexing configuration to see if it's
ready
                   var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                   if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
```

```
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous SearchIndexà la section Référence des AWS SDK pour .NET API.

`UpdateThing`

L'exemple de code suivant montre comment utiliser`UpdateThing`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Updates an IoT Thing with attributes.
   /// </summary>
   /// <param name="thingName">The name of the Thing to update.</param>
   /// <param name="attributes">Dictionary of attributes to add.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
   {
       try
       {
           var request = new UpdateThingRequest
           {
               ThingName = thingName,
               AttributePayload = new AttributePayload
               {
                   Attributes = attributes,
                   Merge = true
               }
           };

           await _amazonIoT.UpdateThingAsync(request);
           _logger.LogInformation($"Updated Thing {thingName} with attributes");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
```

```
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateThingà la section Référence des AWS SDK pour .NET API.

AWS IoT data exemples utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec AWS IoT data.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Actions

Actions

`GetThingShadow`

L'exemple de code suivant montre comment utiliser`GetThingShadow`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetThingShadowà la section Référence des AWS SDK pour .NET API.

`UpdateThingShadow`

L'exemple de code suivant montre comment utiliser`UpdateThingShadow`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Updates the Thing's shadow with new state information.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="shadowPayload">The shadow payload in JSON format.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
   {
       try
       {
           var request = new UpdateThingShadowRequest
           {
               ThingName = thingName,
               Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
           };

           await _amazonIotData.UpdateThingShadowAsync(request);
           _logger.LogInformation($"Updated shadow for Thing {thingName}");
           return true;
       }
       catch (Amazon.IotData.Model.ResourceNotFoundException ex)
       {
```

```
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous UpdateThingShadowà la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation d'Amazon Redshift SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon Redshift.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions

Mise en route

Bonjour Amazon Redshift

L'exemple de code suivant montre comment commencer à utiliser Amazon Redshift.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
```

```
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeClustersà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un cluster Redshift.
- Répertoriez les bases de données du cluster.
- Créez une table intitulée Films.
- Renseignez la table Films.
- Interrogez la table Films par année.
- Modifiez le cluster Redshift.
- Supprimez le cluster Amazon Redshift.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une classe d'enveloppe Redshift pour gérer les opérations.

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
 IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
 databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
 "ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
```

```
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
```

```
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
 {cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };
```

```
            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
    {
        try
        {
            var sqlStatement = @"
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
```

```
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
```

```
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

```
    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);
```

```
        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
 cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

Exécutez un scénario interactif illustrant les principes de base de Redshift.

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
```

```
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
```

```
                    userName = userInput;


 Console.WriteLine("=========================================================================
                Console.WriteLine("Please enter a user password for the cluster
 (default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


 Console.WriteLine("=========================================================================

                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
 clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
 userName, userPassword);

 Console.WriteLine("=========================================================================

            // Step 4: Wait for cluster to become available

 Console.WriteLine("=========================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

 Console.WriteLine("=========================================================================

            // Step 5: List databases

 Console.WriteLine("=========================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
 database is created by default and used in this scenario.");
```

```
            Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

Console.WriteLine("=========================================================================



Console.WriteLine("=========================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("=========================================================================

            // Step 6: Create Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("=========================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
                    recordCount = inputCount;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("================================================================================

            // Step 8 & 9: Query movies by year

Console.WriteLine("================================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)
```

```
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
 {year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
 userName, year);

 Console.WriteLine("=========================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=========================================================================

            // Step 11 & 12: Delete cluster confirmation

 Console.WriteLine("=========================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
 cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
```

```
                    await
Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

Console.WriteLine("=====================================================================


Console.WriteLine("=====================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=====================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
        }
```

```
        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
 file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement

- GetStatementResult
- ListDatabasesPaginator
- ModifyCluster

Actions

`CreateCluster`

L'exemple de code suivant montre comment utiliser`CreateCluster`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Create a new Amazon Redshift cluster.
   /// </summary>
   /// <param name="clusterIdentifier">The identifier for the cluster.</param>
   /// <param name="databaseName">The name of the database.</param>
   /// <param name="masterUsername">The master username.</param>
   /// <param name="masterUserPassword">The master user password.</param>
   /// <param name="nodeType">The node type for the cluster.</param>
   /// <returns>The cluster that was created.</returns>
   public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
       string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
   {
       try
       {
           var request = new CreateClusterRequest
           {
               ClusterIdentifier = clusterIdentifier,
               DBName = databaseName,
               MasterUsername = masterUsername,
               MasterUserPassword = masterUserPassword,
```

```
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateClusterà la section Référence des AWS SDK pour .NET API.

`DeleteCluster`

L'exemple de code suivant montre comment utiliser`DeleteCluster`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
```

```
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteClusterà la section Référence des AWS SDK pour .NET API.

`DescribeClusters`

L'exemple de code suivant montre comment utiliser`DescribeClusters`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Describe Amazon Redshift clusters.
   /// </summary>
   /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
   /// <returns>A list of clusters.</returns>
   public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
   {
       try
       {
           var clusters = new List<Cluster>();
           var request = new DescribeClustersRequest();
           if (!string.IsNullOrEmpty(clusterIdentifier))
           {
               request.ClusterIdentifier = clusterIdentifier;
           }

           var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
           await foreach (var response in clustersPaginator.Responses)
           {
               if (response.Clusters != null)
                   clusters.AddRange(response.Clusters);
           }

           Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
           foreach (var cluster in clusters)
           {
               Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
           }

           return clusters;
```

```
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeClustersà la section Référence des AWS SDK pour .NET API.

`DescribeStatement`

L'exemple de code suivant montre comment utiliser`DescribeStatement`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
```

```
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DescribeStatementà la section Référence des AWS SDK pour .NET API.

`GetStatementResult`

L'exemple de code suivant montre comment utiliser`GetStatementResult`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
```

```
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetStatementResultà la section Référence des AWS SDK pour .NET API.

`ListDatabases`

L'exemple de code suivant montre comment utiliser`ListDatabases`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ListDatabasesà la section Référence des AWS SDK pour .NET API.

`ModifyCluster`

L'exemple de code suivant montre comment utiliser`ModifyCluster`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };
```

```
            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous ModifyClusterà la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon S3 utilisant SDK pour .NET (v4)

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants en utilisant le AWS SDK pour .NET (v4) avec Amazon S3.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- Mise en route
- Principes de base
- Actions
- Scénarios

Mise en route

Hello Amazon S3

L'exemple de code suivant montre comment commencer à utiliser Amazon S3.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Hello Amazon Simple Storage Service
// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var s3Client = new AmazonS3Client();

        try
        {
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));
```

```
            // Use the built-in paginator to list buckets
            var request = new ListBucketsRequest();
            var paginator = s3Client.Paginators.ListBuckets(request);

            var buckets = new List<S3Bucket>();

            await foreach (var response in paginator.Responses)
            {
                buckets.AddRange(response.Buckets);
            }

            if (buckets.Any())
            {
                Console.WriteLine($"Found {buckets.Count} S3 buckets:");
                Console.WriteLine();

                foreach (var bucket in buckets)
                {
                    Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
                    Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-
MM-dd HH:mm:ss UTC}");
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No S3 buckets found in your account.");
            }

            Console.WriteLine("Hello S3 completed successfully.");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"S3 service error occurred: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list S3 buckets. Here's why:
 {ex.Message}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListBucketsà la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer un compartiment et y charger un fichier ;
- télécharger un objet à partir d'un compartiment ;
- copier un objet dans le sous-dossier d'un compartiment ;
- répertorier les objets d'un compartiment ;
- supprimer le compartiment et tous les objets qui y figurent.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif illustrant les fonctionnalités d'Amazon S3.

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
```

```
                    .AddTransient<S3Wrapper>()
                    .AddLogging(builder => builder.AddConsole()))
            .Build();

        _logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<S3_Basics>();

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

        var sepBar = new string('-', 45);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
        Console.WriteLine("\n\t5. Delete all the items in the bucket");
        Console.WriteLine("\n\t6. Delete the bucket");
        Console.WriteLine(sepBar);

        await RunScenario(_s3Wrapper, _logger);

        Console.WriteLine(sepBar);
        Console.WriteLine("The Amazon S3 scenario has successfully completed.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Run the S3 Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="s3Wrapper">The S3 wrapper instance.</param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
    {
        string bucketName = BucketName;
        string filePath = TempFilePath;
        string keyName = string.Empty;

        var sepBar = new string('-', 45);
```

```
        try
        {
            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                Console.Write("Please enter a name for the new bucket: ");
                bucketName = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Using bucket name: {bucketName}");
            }

            var success = await s3Wrapper.CreateBucketAsync(bucketName);
            if (success)
            {
                Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not create bucket: {bucketName}.\n");
            }

            Console.WriteLine(sepBar);
            Console.WriteLine("Upload a file to the new bucket.");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                // Get the local path and filename for the file to upload.
                while (string.IsNullOrEmpty(filePath))
                {
                    Console.Write("Please enter the path and filename of the file to
upload: ");
                    filePath = Console.ReadLine();

                    // Confirm that the file exists on the local computer.
                    if (!File.Exists(filePath))
                    {
```

```
                        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
                        filePath = string.Empty;
                    }
                }
            }
            else
            {
                // Use the public variable if set, otherwise create a temp file
                if (!string.IsNullOrEmpty(TempFilePath))
                {
                    filePath = TempFilePath;
                    Console.WriteLine($"Using provided test file: {filePath}");
                }
                else
                {
                    // Create a temporary test file for non-interactive mode
                    filePath = Path.GetTempFileName();
                    var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
                    await File.WriteAllTextAsync(filePath, testContent);
                    Console.WriteLine($"Created temporary test file: {filePath}");
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
 filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
 to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set up download path
            string downloadPath = string.Empty;
```

```
            if (IsInteractive)
            {
                // Now get a new location where we can save the file.
                while (string.IsNullOrEmpty(downloadPath))
                {
                    // First get the path to which the file will be downloaded.
                    Console.Write("Please enter the path where the file will be
downloaded: ");
                    downloadPath = Console.ReadLine();

                    // Confirm that the file doesn't already exist on the local
computer.
                    if (File.Exists($"{downloadPath}\\{keyName}"))
                    {
                        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                        downloadPath = string.Empty;
                    }
                }
            }
            else
            {
                downloadPath = Path.GetTempPath();
                var downloadFile = Path.Combine(downloadPath, keyName);
                if (File.Exists(downloadFile))
                {
                    File.Delete(downloadFile);
                }

                Console.WriteLine($"Using download path: {downloadPath}");
            }

            // Download an object from a bucket.
            success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);

            if (success)
            {
                Console.WriteLine($"Successfully downloaded {keyName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not download {keyName}.\n");
            }
```

```
            // Copy the object to a different folder in the bucket.
            string folderName = string.Empty;

            if (IsInteractive)
            {
                while (string.IsNullOrEmpty(folderName))
                {
                    Console.Write("Please enter the name of the folder to copy your
 object to: ");
                    folderName = Console.ReadLine();
                }
            }
            else
            {
                folderName = "test-folder";
                Console.WriteLine($"Using folder name: {folderName}");
            }

            await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
 folderName);

            // List the objects in the bucket.
            await s3Wrapper.ListBucketContentsAsync(bucketName);

            // Delete the contents of the bucket.
            if (IsInteractive)
            {
                Console.WriteLine("Press <Enter> when you are ready to delete the
 bucket contents.");
                _ = Console.ReadLine();
            }

            var deleteContentsSuccess = await
 s3Wrapper.DeleteBucketContentsAsync(bucketName);
            if (deleteContentsSuccess)
            {
                Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete contents of
 {bucketName}.\n");
```

```
            }

            if (IsInteractive)
            {
                // Deleting the bucket too quickly after separately deleting its
contents can
                // cause an error that the bucket isn't empty. To delete contents
and bucket in one
                // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
                Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
                _ = Console.ReadLine();
            }
            else
            {
                // Add a small delay for non-interactive mode to ensure objects are
fully deleted.
                Console.WriteLine("Waiting a moment for objects to be fully
deleted...");
                await Task.Delay(2000);
            }

            // Delete the bucket.
            var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
            if (deleteSuccess)
            {
                Console.WriteLine($"Successfully deleted {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (File.Exists(filePath))
                    {
                        File.Delete(filePath);
                        Console.WriteLine("Cleaned up temporary test file.");
                    }
```

```
                    var downloadFile = Path.Combine(downloadPath, keyName);
                    if (File.Exists(downloadFile))
                    {
                        File.Delete(downloadFile);
                        Console.WriteLine("Cleaned up downloaded test file.");
                    }
                }
                catch (Exception ex)
                {
                    scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
                }
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

            // Clean up on error - delete bucket if it exists
            try
            {
                if (!string.IsNullOrEmpty(bucketName))
                {
                    await s3Wrapper.DeleteBucketContentsAsync(bucketName);
                    await s3Wrapper.DeleteBucketAsync(bucketName);
                }
            }
            catch (Exception cleanupEx)
            {
                scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
                    {
                        File.Delete(filePath);
                    }
                }
                catch (Exception fileCleanupEx)
```

```
                {
                    scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
 temporary files during error handling.");
                }
            }

            throw;
        }
    }
}
```

Une classe d'encapsuleur pour les méthodes du kit SDK Amazon S3.

```
using Amazon.S3;
using Amazon.S3.Model;

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
    }


    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
```

```
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
```

```
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
```

```
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Copies an object in an Amazon S3 bucket to a folder within the
    /// same bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object to copy is located.</param>
    /// <param name="objectName">The object to be copied.</param>
    /// <param name="folderName">The folder to which the object will
    /// be copied.</param>
    /// <returns>A boolean value that indicates the success or failure of
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
```

```
            return false;
        }
    }



    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }
```

```
            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }



    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }
```

```
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest { BucketName = bucketName, };

            await _amazonS3.DeleteBucketAsync(request);
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting bucket: {ex.Message}");
            return false;
        }
    }

}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
  - CopyObject
  - CreateBucket

- DeleteBucket
- DeleteObjects
- GetObject
- ListObjectsV2
- PutObject

Actions

`CopyObject`

L'exemple de code suivant montre comment utiliser`CopyObject`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
```

```
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CopyObjectà la section Référence des AWS SDK pour .NET API.

`CreateBucket`

L'exemple de code suivant montre comment utiliser`CreateBucket`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
```

```
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateBucketà la section Référence des AWS SDK pour .NET API.

`CreatePresignedPost`

L'exemple de code suivant montre comment utiliser`CreatePresignedPost`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une URL POST présignée.

```
    /// <summary>
    /// Create a presigned POST URL with conditions.
    /// </summary>
    /// <param name="s3Client">The Amazon S3 client.</param>
```

```
    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
 be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
            }
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }
```

- Pour plus de détails sur l'API, reportez-vous CreatePresignedPostà la section Référence des AWS SDK pour .NET API.

`DeleteBucket`

L'exemple de code suivant montre comment utiliser`DeleteBucket`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteBucketà la section Référence des AWS SDK pour .NET API.

`DeleteObjects`

L'exemple de code suivant montre comment utiliser`DeleteObjects`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }

            return true;
        }
```

```
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous DeleteObjectsà la section Référence des AWS SDK pour .NET API.

`GetObject`

L'exemple de code suivant montre comment utiliser`GetObject`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
```

```
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous GetObjectà la section Référence des AWS SDK pour .NET API.

`ListObjectsV2`

L'exemple de code suivant montre comment utiliser`ListObjectsV2`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
```

```
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
```

```
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }
```

- Pour plus de détails sur l'API, voir ListObjectsV2 dans le manuel de référence des AWS SDK pour .NET API.

`PutObject`

L'exemple de code suivant montre comment utiliser`PutObject`.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
```

```
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous PutObjectà la section Référence des AWS SDK pour .NET API.

Scénarios

Créer une URL présignée

L'exemple de code suivant montre comment créer une URL présignée pour Amazon S3 et charger un objet.

SDK pour .NET (v4)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez et utilisez un POST présigné URLs pour les téléchargements directs par navigateur.

```
/// <summary>
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }
```

```
    /// <summary>
    /// Perform the actions defined for the Amazon S3 Presigned POST scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        _isInteractive = !args.Contains("--non-interactive");

        // Set up dependency injection for Amazon S3
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddHttpClient()
            )
            .Build();

        SetUpServices(host);

        try
        {
            // Display overview
            _uiMethods.DisplayOverview();
            _uiMethods.PressEnter(_isInteractive);

            // Step 1: Create bucket
            await CreateBucketAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 2: Create presigned URL
            _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
            var response = await CreatePresignedPostAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 3: Display URL and fields
            _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
            DisplayPresignedPostFields(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 4: Upload file
```

```
            _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
 URL");
            await UploadFileAsync(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 5: Verify file exists
            await VerifyFileExistsAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 6: Cleanup
            _uiMethods.DisplayTitle("Step 6: Clean up resources");
            await CleanupAsync();

            _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
 successfully!");
            _uiMethods.PressEnter(_isInteractive);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error in scenario");
            Console.WriteLine($"Error: {ex.Message}");

            // Attempt cleanup if there was an error
            if (!string.IsNullOrEmpty(_bucketName))
            {
                _uiMethods.DisplayTitle("Cleaning up resources after error");
                await _s3Wrapper.DeleteBucketAsync(_bucketName);
                Console.WriteLine($"Cleaned up bucket: {_bucketName}");
            }
        }
    }

    /// <summary>
    /// Create an S3 bucket for the scenario.
    /// </summary>
    private static async Task CreateBucketAsync()
    {
        _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

        // Generate a default bucket name for the scenario
        var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

        // Prompt user for bucket name or use default in non-interactive mode
```

```
        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}'): ",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrWhiteSpace(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");

        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }


    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
    private static async Task<CreatePresignedPostResponse>
 CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
 demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);

        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.
```

```
    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");

        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
{uploadResult.StatusCode}");
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
{uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {
```

```
        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
 URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);

            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
 formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
 httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
 responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>
```

```
    private static async Task VerifyFileExistsAsync()
    {
        _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

        Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

        try
        {
            var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
 _objectKey!);

            Console.WriteLine($"File verification successful! File exists in the
 bucket.");
            Console.WriteLine($"File size: {metadata.ContentLength} bytes");
            Console.WriteLine($"File type: {metadata.Headers.ContentType}");
            Console.WriteLine($"Last modified: {metadata.LastModified}");
        }
        catch (AmazonS3Exception ex) when (ex.StatusCode ==
 System.Net.HttpStatusCode.NotFound)
        {
            Console.WriteLine($"Error: File was not found in the bucket.");
            throw;
        }
    }

    private static void DisplayPresignedPostFields(CreatePresignedPostResponse
 response)
    {
        Console.WriteLine($"Presigned POST URL: {response.Url}");
        Console.WriteLine("Form fields to include:");

        foreach (var field in response.Fields)
        {
            Console.WriteLine($"  {field.Key}: {field.Value}");
        }
    }

    /// <summary>
    /// Clean up resources created by the scenario.
    /// </summary>
    private static async Task CleanupAsync()
    {
        if (!string.IsNullOrEmpty(_bucketName))
```

```
        {
            Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");
            bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);

            if (result)
            {
                Console.WriteLine("Bucket deleted successfully");
            }
            else
            {
                Console.WriteLine("Failed to delete bucket - it may have been
 already deleted");
            }
        }
    }
}
```

Migration de votre projet pour AWS SDK pour .NET

Cette section fournit des informations sur les tâches de migration pour la version 4 du AWS SDK pour .NET, ainsi que des instructions sur la manière d'effectuer ces tâches. Pour plus d'informations sur la migration vers des versions antérieures (par exemple, la migration de la V2 vers la V3, ou d'une sous-version V3 vers une autre), voir Migrer votre projet dans le Guide du AWS SDK pour .NET développeur (version 3).

Voir également Nouveautés etPlateformes prises en charge.

Rubriques

- Migration vers la version 4 du AWS SDK pour .NET
- Migration à partir de .NET Standard 1.3

Migration vers la version 4 du AWS SDK pour .NET

La AWS SDK pour .NET version 4 (V4) comporte un nombre important de modifications majeures par rapport à la version 3 (V3) du SDK. Cette rubrique décrit les modifications majeures apportées à la version 4 et les tâches que vous pourriez avoir à effectuer pour migrer votre environnement ou votre code depuis la V3. Pour plus d'informations sur les autres modifications importantes apportées au SDK, consultez les ressources suivantes :

- Le problème du logiciel de suivi du développement sur GitHub :. https://github.com/aws/aws-sdk-net/issues/3362
- Le billet de blog Preview 1 of AWS SDK pour .NET V4.
- Le billet de blog Preview 4 of AWS SDK pour .NET V4.
- Le billet de blog Disponibilité générale de la AWS SDK pour .NET version 4.0.

.NET Framework

La cible .NET Framework 3.5 a été supprimée de la version 4 du AWS SDK pour .NET. Par conséquent, le SDK ne prend plus en charge .NET Framework 3.5. Cette version du SDK est compilée avec .NET Framework 4.7.2 et s'exécute dans le runtime .NET 4.0. Pour plus d'informations, consultez la section Plateformes prises en charge.

Types de valeur

Les propriétés qui utilisent des types de valeur dans les classes utilisées pour effectuer des demandes et des réponses ont été modifiées pour utiliser des types de valeurs nullables. Les propriétés des types suivants ont été modifiées :

- `bool`a été changé en `bool?`
- `double`a été changé en `double?`
- `int`a été changé en `int?`
- `float`a été changé en `float?`
- `long`a été changé en `long?`
- `DateTime`a été changé en `DateTime?`

Pour plus d'informations sur cette modification, consultez le billet de blog Preview 1 of AWS SDK pour .NET V4.

Collections

Les propriétés qui utilisent des collections dans des classes utilisées pour effectuer des demandes et des réponses sont désormais définies par défaut sur`null`. Par conséquent, votre code doit vérifier qu'une collection n'est pas nulle avant d'essayer de l'utiliser. Par exemple :

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

Le comportement de la V3 lors de l'initialisation des collections peut être restauré en `Amazon.AWSConfigs.InitializeCollections` réglant sur. `true` Cette propriété existe également dans la version 3 pour les utilisateurs qui souhaitent essayer ce changement de comportement avant de passer à la version 4.

Pour plus d'informations sur cette modification, consultez le billet de blog Preview 1 of AWS SDK pour .NET V4.

AWS Security Token Service (STS)

- Le point final régional

  Lorsque vous utilisez des fournisseurs d'informations d'identification qui s'appuient sur AWS STS, les appels utilisent toujours le point de terminaison régional. Cela diffère de la version 3 du SDK, qui utilisait la `us-east-1` région par défaut lors de l'exécution sur la partition publique, quelle que soit la région configurée.

- L'**`StsRegionalEndpointsValue`**énumération

  L'`StsRegionalEndpointsValue`énumération a été supprimée de l'espace de noms Amazon.Runtime. Tout code utilisant cette énumération doit être supprimé.

- La classe **`STSAssumeRoleAWSCredentials`**

  Le fournisseur d'informations d'identification STS assume le rôle obsolète `STSAssumeRoleAWSCredentials` a été supprimé d'Amazon. SecurityTokenespace de noms. Utilisez plutôt AssumeRoleAWSCredentialsdepuis Amazon.Runtime.

Changements liés à **`ClientConfig`**

L'Amazon.Runtime. ClientConfigclass est la classe de base des classes de configuration des clients de service telles que AmazonS3Config. Les modifications suivantes ont été apportées à cette classe de base.

- Mode de nouvelle tentative par défaut

  La valeur par défaut de la `RetryMode` propriété est au `Standard` lieu de. `Legacy` Par conséquent, la `Legacy` valeur a été supprimée d'Amazon.Runtime. RequestRetryModeénum.

- Mode de configuration par défaut

  La valeur par défaut de la `DefaultConfigurationMode` propriété est au `Standard` lieu de. `Legacy` Par conséquent, la `Legacy` valeur a été supprimée d'Amazon.Runtime. DefaultConfigurationModeénum.

- La **`ReadWriteTimeout`** propriété

La `ReadWriteTimeout` propriété obsolète a été supprimée de toutes les cibles à l'exception de .NET Framework 4.7.2.

Les AWSSDK .Extensions. NETCore.Package d'installation NuGet

Les AWSSDK.Extensions. NETCoreLe NuGet package .Setup a été mis à jour pour résoudre les problèmes présents dans la version 3 du SDK et pour sécuriser le package pour Native AOT. Ces modifications sont résumées ci-dessous. Pour des informations détaillées, voir le PR 3353 dans le aws-sdk-netréférentiel sur GitHub.

- La classe **`DefaultClientConfig`**

  La `DefaultClientConfig` classe n'est plus héritée de la classe de base de configuration du client de service Amazon.Runtime. ClientConfig. Les propriétés pertinentes de `ClientConfig` ont été répliquées à `DefaultClientConfig` l'aide de types de valeurs nullables. Cette modification nous permet de détecter le moment où une valeur a été définie `DefaultClientConfig` lors de la copie des valeurs dans la configuration en cours de création pour le client de service.

  L'un des résultats particuliers de ce changement est qu'il n'`DefaultClientConfig.HttpClientFactory`est plus disponible dans la version 4. Utilisez `AWSConfigs.HttpClientFactory` à la place. Pour plus d'informations, consultez le GitHub numéro 3790.

  Cette modification a également entraîné la suppression des surcharges génériques de la méthode d'`IConfiguration.GetAWSOptions`extension qui acceptait un objet de configuration de service. La surcharge non générique doit être utilisée à la place, et le SDK se chargera automatiquement de renseigner les paramètres spécifiques au service. Pour plus d'informations, consultez le GitHub numéro 3866.

- AOT natif

  Un nouveau mécanisme de création de clients de service utilisant les méthodes d'interface statique C# 11 a été ajouté au package. Cette modification élimine le besoin d'effectuer des chargements de type Assembly pour créer des instances de clients de service, y compris la manipulation de chaînes du nom de l'interface de service pour calculer le type de client de service, ce qui est incompatible avec l'AOT natif. Cette modification n'est disponible que pour .NET 8 et versions ultérieures ; les anciennes versions utilisent toujours le mécanisme d'origine.

Pour plus d'informations sur ce package dans ce guide, consultezAWSSDK.Prolongations. NETCore.Configuration et IConfiguration. Le code source de ce package se trouve GitHub surhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

`CookieSigner` et `UrlSigner`

Les `UrlSigner` extensions `CookieSigner` et pour Amazon CloudFront ont été déplacées vers un package d'extension distinct appelé AWSSDK.Extensions. CloudFront.Signataires. Ce changement vise à prendre en charge OpenSSL 3 et à devenir dépendant de .Cryptography. BouncyCastle

Le code source de ce package se trouve GitHub surhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers.

DateTime contre UTC DateTime

Certaines classes V3 ont une DateTime propriété marquée comme « obsolète » ou « obsolète », ainsi qu'une propriété UTC alternative. DateTime Dans ces classes, la DateTime propriété obsolète a été supprimée et le nom de la DateTime propriété UTC a été remplacé par le nom d'origine de la DateTime propriété.

Voici quelques exemples de classes pour lesquelles cette modification a été mise en œuvre.

- DescribeSpotPriceHistoryRequest:
  - La `StartTime` propriété obsolète a été supprimée et le nom de la `StartTimeUtc` propriété a été changé en « StartTime ».
  - La `EndTime` propriété obsolète a été supprimée et le nom de la `EndTimeUtc` propriété a été changé en « EndTime ».
- CreateFleetRequest
  - La `ValidFrom` propriété obsolète a été supprimée et le nom de la `ValidFromUtc` propriété a été changé en « ValidFrom ».
  - La `ValidUntil` propriété obsolète a été supprimée et le nom de la `ValidUntilUtc` propriété a été changé en « ValidUntil ».

Cette modification peut entraîner des temps de décalage si une application utilise la DateTime propriété obsolète d'origine. Une erreur de compilation se produira pour le code utilisant la DateTime propriété UTC.

DateTime analyse syntaxique

La DateTimeUnmarshaller classe a été mise à jour. Cette classe analysait et renvoyait des DateTime chaînes en heure locale. Dans certains cas, ces valeurs ont été reconverties en UTC en raison d'une mise à jour antérieure, mais pas toujours. Désormais, les DateTime chaînes qui ne sont pas rassemblées sont supposées être en UTC et seront spécifiées et non rassemblées en UTC. Cette mise à jour inclut les modifications de comportement suivantes.

Certaines propriétés d'horodatage basées sur la DateTime classe étaient analysées en heure locale. Il s'agissait notamment de démarshallers de réponse pour les horodatages et de listes d'horodatages pour les formats et. `TimestampFormat.ISO8601` `TimestampFormat.RFC822` DateTime l'analyse a été mise à jour pour renvoyer les heures UTC à la place.

`ConvertFromUnixEpochSeconds` et **`ConvertFromUnixEpochMilliseconds`**

Les ConvertFromUnixEpochMillisecondsméthodes ConvertFromUnixEpochSecondset, qui convertissent les secondes d'époque Unix en une DateTime structure, renvoyaient l'heure d'époque Unix sous forme d'heure locale au lieu d'heure UTC. Ces méthodes renvoient désormais l'heure UTC.

Logging

La façon dont vous activez la connexion au SDK a été mise à jour pour la version 4. La connexion à la console et aux diagnostics du système fonctionne de la même manière que dans la version 3, c'est-à-dire en définissant la `LoggingConfig.LogTo` propriété de la AWSConfigsclasse sur `LoggingOptions.Console` ou`LoggingOptions.SystemDiagnostics`. L'LoggingOptionsoption for `log4net` a été supprimée, de même que la logique interne du SDK permettant d'utiliser la réflexion pour s'attacher à une instance en mémoire de. `log4net`

Pour inclure la connexion du SDK dans une structure de journalisation, un module adaptateur distinct est utilisé pour connecter le SDK à la structure de journalisation. Utilisez le package AWSSDK.Extensions.Logging.Log4 pour et le package NetAdaptor .Extensions.Logging. `log4net` AWSSDK ILoggerEnsemble d'adaptateurs pour. `Microsoft.Extensions.Logging` Les exemples de code suivants vous montrent comment configurer la journalisation dans ces deux cas.

Exemple de configuration pour log4net

Ajoutez le `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet package et appelez la `ConfigureAWSSDKLogging` méthode statique à partir de`Log4NetAWSExtensions`.

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Exemple de configuration pour Microsoft.Extensions.Logging

Ajoutez le `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet package et appelez la méthode d'`ConfigureAWSSDKLogging`extension depuis l'`ILoggerFactory`interface.

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support pour HTTP 2

Support du protocole HTTP 2 a été ajouté pour permettre le streaming bidirectionnel. Pour de plus amples informations, veuillez consulter Support pour HTTP 2.

Authentification unique

La valeur par défaut de la `SupportsGettingNewToken` propriété de la classe SSOAWSCredentialsOptions est passée de `true` à`false`. Si certaines applications utilisent la SSOAWSCredentialsclasse pour obtenir des informations d'identification SSO, vous devrez peut-être définir la `Options.SupportsGettingNewToken` propriété sur. `true` Pour des exemples de cette configuration, consultez les exemples de code dansTutoriel pour l'authentification unique utilisant uniquement des applications .NET. Pour plus d'informations, voir PR 3737 dans le aws-sdk-net GitHub référentiel.

Modifications spécifiques à DynamoDB

Les modifications suivantes sont spécifiques à Amazon DynamoDB. Nombre d'entre eux sont en train de changer radicalement.

Pour plus d'informations sur les modifications apportées à DynamoDB dans la version 4 du, consultez AWS SDK pour .NET le billet de blog Preview 4 of V4. AWS SDK pour .NET

Ouvert pour voir les articles

Les modifications apportées à la version 4 du SDK pour DynamoDB résolvent certains problèmes liés à la testabilité, mais concernent principalement les bibliothèques de haut niveau :

- Le modèle de document .NET, nommé DocumentModeldans le code.
- Le modèle de persistance des objets .NET, nommé DataModeldans le code.

Pour obtenir des informations détaillées sur ces modes de programmation, consultez DynamoDB ce guide.

Modèle de document : exception mise à jour pour l'interface simulée **`IAmazonDynamoDB`**

Dans le modèle de document antérieur à la version V4 du SDK, si une table était initialisée avec une interface IAmazonDynamoDB simulée, elle retournait. `NullReferenceException` La version V4 du SDK revient à la `InvalidOperationException` place. `Table`Les méthodes asynchrones devraient fonctionner avec un client simulé, mais il se peut que vous rencontriez des exceptions lorsque vous appelez des méthodes synchrones depuis. `.NET/Core/Standard`

Pour plus d'informations sur cette modification, consultez le PR 3388 sur GitHub.

Modèle de document : **`FromJson`** et **`ToJson`** méthodes

Les `ToJson` méthodes `FromJson` et de la classe Document sont désormais utilisées à la `System.Text.Json` place LitJson de la sérialisation et ont LitJson été supprimées de la version 4 du SDK. L'un des avantages de son utilisation `System.Text.Json` est que cet analyseur prend en charge l'utilisation du `Decimal` type .NET, qui permet une plus grande précision pour les propriétés numériques à virgule flottante.

Modèle de persistance des objets : la **`DynamoDBOperationConfig`** classe

Dans le modèle de persistance des objets, les modifications suivantes ont été apportées à la classe Dynamo DBOperation Config partagée :

- La classe a été séparée en nouvelles classes spécifiques à l'opération SaveConfig, telles que LoadConfig, et. QueryConfig Les méthodes qui `DynamoDBOperationConfig` ont été marquées comme obsolètes sont susceptibles d'être supprimées à l'avenir.

Pour plus d'informations sur cette modification, consultez le PR 3421 sur GitHub.

- Les `DisableFetchingTableMetadata` propriétés `MetadataCachingMode` et ont été supprimées de la classe. Ces propriétés n'étaient pas incluses dans les nouvelles classes spécifiques aux opérations mentionnées précédemment. Les propriétés supprimées sont des paramètres au niveau de la table qui doivent être spécifiés sur la `Context` propriété globale de la classe AWSConfigsDynamoDB ou sur la classe Dynamo Config. DBContext

  Pour plus d'informations sur cette modification, consultez le PR 3422 sur GitHub.

- La classe n'hérite plus de la classe Dynamo DBContext Config. Cela vous empêche de transmettre un `DynamoDBOperationConfig` objet au constructeur de Dynamo DBContext, où certaines propriétés de la configuration spécifique à l'opération (telles que`OverrideTableName`) ne s'appliquent pas.

  Pour plus d'informations sur cette modification, consultez le PR 3422 sur GitHub.

Modèle de persistance des objets : polymorphisme

La DBPolymorphic TypeAttribute classe Dynamo a été ajoutée au modèle de persistance des objets. Cette classe permet de prendre en charge la sérialisation et la désérialisation des types polymorphes. Pour plus d'informations, consultez le PR 3643 sur GitHub.

Modèle de document et modèle de persistance des objets : opérations simulables

De nouvelles interfaces spécifiques aux opérations ont été ajoutées pour permettre aux clients de simuler des opérations DynamoDB. Les méthodes d'usine de l'IDynamoDBContextinterface ont été mises à jour pour renvoyer les nouvelles interfaces.

Pour plus d'informations sur cette modification, consultez le PR 3450 sur GitHub.

- Modèle de persistance des objets
  - `BatchGet`Opérations simulées via les `IMultiTableBatchGet` interfaces `IBatchGet` et.
  - `BatchWrite`Opérations simulées via les `IMultiTableBatchWrite` interfaces `IBatchWrite` et.
  - `TransactGet`Opérations simulées via les `IMultiTableTransactGet` interfaces `ITransactGet` et.
  - `TransactWrite`Opérations simulées via les `IMultiTableTransactWrite` interfaces `ITransactWrite` et.

  - `Scan`Maquette et `Query` opérations via l'`IAsyncSearch`interface.
- Modèle de document
  - `Table`Opérations simulées via l'`ITable`interface.
  - `Scan`Maquette et `Query` opérations via l'`ISearch`interface.
  - `TransactWrite`Opérations simulées via les `IMultiTableDocumentTransactWrite` interfaces `IDocumentTransactWrite` et.
  - `TransactGet`Opérations simulées via les `IMultiTableDocumentTransactGet` interfaces `IDocumentTransactGet` et.
  - `BatchWrite`Opérations simulées via les `IMultiTableDocumentBatchWrite` interfaces `IDocumentBatchWrite` et.
  - `BatchGet`Opérations simulées via les `IMultiTableDocumentBatchGet` interfaces `IDocumentBatchGet` et.

Modèle de document et modèle de persistance des objets : Support de l'AOT natif

L'une des limites de l'AOT natif est la prise en charge des types .NET imbriqués. Dans certains cas, ces types imbriqués peuvent passer inaperçus par le composant de découpage du compilateur .NET. Dans ce cas, il se peut que vous receviez une exception telle que : « `System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.` »

Vous pouvez contourner cette limitation en ajoutant un élément dans `DynamicDependency` le chemin du code qui informe le trimmer de la dépendance à l'égard du sous-type. Le constructeur du type .NET de niveau supérieur en cours d'enregistrement est un endroit probable. L'exemple de code suivant montre comment utiliser l'`DynamicDependency`attribut :

```
[DynamoDBTable("TestTable")]
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
```

```
    public string Name { get; set; }

    public SubType SubType { get; set; }
}

class SubType
{
    public string SubName { get; set; }
}
```

Dynamo DBStreams

Dynamo DBStreams a été supprimé du package AWSSDK.DynamoDB et est disponible dans son propre NuGet package, AWSSDK.Dynamo, et possède son propre espace de noms DBStreams,. `Amazon.DynamoDBStreams`

Autoriser la suppression de la **`TableNamePrefix`** valeur

Vous pouvez désormais supprimer la valeur de la `TableNamePrefix` propriété dans la classe Dynamo DBContext Config au niveau de l'opération individuelle. Pour plus d'informations sur cette modification, consultez le PR 3476 sur GitHub.

propriété **`RetrieveDateTimeInUtc`**

Pour la classe Dynamo DBContext Config, la valeur par défaut de la `RetrieveDateTimeInUtc` propriété a été modifiée en`true`.

propriété **`DynamoDBContextTableNamePrefix`**

La `DynamoDBContextTableNamePrefix` propriété a été supprimée de la AWSConfigsclasse DynamoDB. Les utilisateurs doivent appeler `AWSConfigsDynamoDB.Context.TableNamePrefix` au lieu de

Modifications spécifiques à EC2

Les modifications suivantes sont spécifiques à Amazon EC2. La plupart ou la totalité d'entre eux sont des changements radicaux.

Ouvert pour voir les articles

`GetDecryptedPassword`

L'`GetDecryptedPassword`extension pour Amazon EC2 a été déplacée vers un package d'extension distinct appelé AWSSDK .extensions.EC2. DecryptPassword. Ce changement vise à prendre en charge OpenSSL 3 et à devenir dépendant de .Cryptography. BouncyCastle

Le code source de ce package se trouve GitHub surhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword.

Support pour Amazon EC2 IMDSv1

Support pour le service de métadonnées d'instance version 1 (IMDSv1) a été supprimé. La version 4 du SDK utilise toujours le service de métadonnées d'instance version 2 (IMDSv2) lors de la récupération des informations d'identification et d'autres métadonnées à partir de l'IMDS. Pour plus d'informations sur l'IMDS, consultez la section Utiliser l'IMDS dans le guide de l'utilisateur Amazon EC2.

Éléments de programmation qui ont été modifiés ou supprimés

- L'espace de `Amazon.EC2.Import` noms et le code dans leur intégralité ont été supprimés.
- L'espace de `Amazon.EC2.Util` noms et le code complets ont été supprimés, y compris les utilitaires AMI utilisés pour rechercher EC2 AMIs pour Windows.
- La `IpRanges` propriété obsolète a été supprimée de la IpPermissionclasse. Utilisez plutôt les `Ipv6Ranges` propriétés `Ipv4Ranges` or.
- Les champs obsolètes suivants ont été supprimés de la EC2InstanceMetadataclasse : `EC2_METADATA_SVCEC2_METADATA_ROOT`,`EC2_USERDATA_ROOT`,`EC2_DYNAMICDATA_ROOT`, et`EC2_APITOKEN_URL`.

Modifications spécifiques à S3

Les modifications suivantes sont spécifiques à Amazon S3. La plupart ou la totalité d'entre eux sont des changements radicaux.

Ouvert pour voir les articles

Région AWS us-east-1

Les clients du service Amazon S3 configurés pour la `us-east-1` région ne peuvent plus accéder aux compartiments des autres régions. Les compartiments doivent être accessibles avec des clients du service S3 configurés pour la région dans laquelle se trouve le compartiment.

Pour plus d'informations sur cette modification, consultez le billet de blog Preview 4 of AWS SDK pour .NET V4.

Client de chiffrement S3

Le client de chiffrement Amazon S3, défini dans l'espace de `Amazon.S3.Encryption` noms, a été supprimé du package AWSSDK.S3. Ce client a été déplacé vers son propre package appelé Amazon.Extensions.S3.Encryption, et sa documentation se trouve à l'adresse. https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html Pour plus d'informations sur la migration, consultezthe section called “Migration du client de chiffrement S3 (V1 vers V2)”. Pour plus d'informations sur le chiffrement S3, consultez la section Algorithmes de chiffrement pris en charge dans le manuel Amazon S3 Encryption Client Developer Guide.

Directive de balisage S3 pour **`CopyObject`**

La `TaggingDirective` propriété a été exposée en tant que propriété publique de la CopyObjectRequestclasse, qui est utilisée par `AmazonS3Client.CopyObject` les méthodes. Cette propriété correspond au `x-amz-tagging-directive` paramètre Amazon S3, tel que défini dans l'CopyObjectaction.

La directive de balisage n'est plus automatiquement définie sur COPY. Si un développeur ne spécifie pas de directive de balisage, le backend S3 suppose automatiquement qu'il s'agit de COPY, mais si un développeur définit explicitement la propriété sur null, la valeur n'est pas définie du tout.

La **`UseArnRegion`** propriété pour la configuration S3

La `UseArnRegion` propriété de la classe Amazon.S3.AmazonS3Config a été mise à jour de telle sorte que la variable d'`AWS_S3_USE_ARN_REGIO`Nenvironnement a priorité sur le paramètre du fichier partagé. `s3_use_arn_region` AWS `config` Pour plus d'informations sur ces variables et paramètres, consultez la section Référence des paramètres dans le Guide de référence des outils AWS SDKs et des outils.

Principales barres obliques pour les méthodes et **`CopyObject` `CopyPart`**

Les barres obliques en tête ne seront plus supprimées pour Amazon S3 `CopyObject` et ses méthodes. `CopyPart` La `DisableTrimmingLeadingSlash` propriété a été supprimée des CopyPartRequestclasses CopyObjectRequestet.

Les **`DoesS3BucketExist...`** méthodes

Les `DoesS3BucketExistAsync` méthodes `DoesS3BucketExist` et méthodes obsolètes ont été supprimées de la classe AmazonS3Util, qui implémente l'interface AmazonS3. ICore Ces méthodes ont été supprimées car elles utilisent toujours le protocole HTTP. Utilisez plutôt DoESS3 BucketExist V2 et DoESS3 V2Async BucketExist.

Le SDK utilise toujours SigV4

La version 4 du utilise AWS SDK pour .NET toujours la version 4 de AWS Signature (SigV4) pour les demandes de signature. Cette modification entraîne les modifications connexes suivantes :

- La `UseSignatureVersion4` propriété de la classe AWSConfigsS3 a été supprimée.
- `SignatureVersion`Propriété de l'Amazon.Runtime. ClientConfigla classe a été supprimée. Cette propriété a été utilisée uniquement par Amazon S3 à des fins de rétrocompatibilité.
- La `RegionEndpoint.Endpoint` classe a été supprimée. Cela inclut la `SignatureVersionOverride` propriété, qui a été utilisée pour remplacer les versions de signature pour Amazon S3. Utilisez plutôt la `client.DetermineServiceOperationEndPoint()` méthode spécifique au service.
- Méthodes mises à jour AmazonS3Util. PostUploadet S3PostUploadSignedPolicy. GetSignedPolicypour utiliser SigV4. En conséquence, la `S3PostUploadSignedPolicy.GetSignedPolicyV4` méthode a été supprimée car elle exécute `GetSignedPolicy` désormais la même fonction. En outre, un troisième paramètre `GetSignedPolicy` a été attribué pour le point de terminaison de la région.

Les **`PutACL`** méthodes **`GetACL`** et

Les `PutACL` méthodes `GetACL` et de la classe AmazonS3Client ont été marquées comme obsolètes. Pour accéder aux fonctionnalités de ces méthodes, utilisez plutôt les nouvelles méthodes suivantes : `GetBucketACL`PutBucketACL,`GetObjectACL`, et`PutObjectACL`.

Éléments de programmation obsolètes supprimés

Un certain nombre d'éléments de programmation de l'implémentation d'Amazon S3 ont été supprimés de la version V4 du SDK, notamment les valeurs d'énumération, les types, les méthodes, les espaces de noms, etc. Ils sont énumérés ci-dessous, s'ils ne l'ont pas déjà été, ainsi que les mesures que vous pouvez prendre pour faciliter leur suppression.

- La `DisableMD5Stream` propriété a été supprimée de la TransferUtilityUploadRequestclasse. Utilisez plutôt la propriété `DisableDefaultChecksumValidation`.

  De plus, la `CalculateContentMD5Header` propriété a été supprimée de la `TransferUtilityUploadRequest` classe. Cette propriété n'est plus nécessaire car le SDK calcule une somme de contrôle par défaut.
- Les `ServerSideEncryptionKeyManagementServiceKeyId` propriétés `ServerSideEncryptionMethod` et ont été supprimées de la CopyPartRequestclasse. Utilisez plutôt les propriétés portant le même nom dans la InitiateMultipartUploadRequestclasse, qui est utilisée dans certaines `InitiateMultipartUpload...` méthodes de la classe AmazonS3Client.
- La `Expires` propriété a été supprimée de la GetObjectResponseclasse. Utilisez plutôt la propriété `ExpiresString`. La chaîne n'est peut-être pas dans un format d'horodatage valide. Votre code doit donc utiliser `TryParse` cette méthode lors de la conversion en un. `DateTime`
- Région AWS Les identifiants obsolètes ont été supprimés de l'énumération S3Region.
- La `Prefix` propriété a été supprimée de la LifecycleRuleclasse. Utilisez plutôt la propriété `Filter`.

  De plus, les `Transition` propriétés `NoncurrentVersionTransition` et ont été supprimées de la `LifecycleRule` classe. Utilisez plutôt `NoncurrentVersionTransitions` les `Transitions` collections et.
- La `Event` propriété a été supprimée de la TopicConfigurationclasse. Utilisez plutôt la `Events` collection.
- CalculateContentMD5Propriété d'en-tête. Il n'est plus nécessaire de définir cette propriété car le SDK calcule une somme de contrôle par défaut.
- La `Bucket` propriété a été supprimée de la SelectObjectContentRequestclasse. Utilisez plutôt la propriété `BucketName`.
- La `NumberOfUploadThreads` propriété a été supprimée de la TransferUtilityConfigclasse. Utilisez plutôt `ConcurrentServiceRequests` la propriété.

Éléments de programmation qui ont été supprimés

Un certain nombre d'éléments de programmation ont été supprimés de la version V4 du SDK, notamment les valeurs d'énumération, les types, les méthodes, les espaces de noms, etc. Ils sont énumérés ci-dessous, s'ils ne l'ont pas déjà été, ainsi que les mesures que vous pouvez prendre pour faciliter leur suppression.

Ouvert pour voir les articles

L'espace de noms **`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`**

L'`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`espace de noms a été supprimé. Cela inclut les identificateurs d'action IAM, qui ont été définis dans la `IdentityandAccessManagementActionIdentifiers` classe. Le code qui utilise ces identificateurs d'action doit être modifié pour utiliser les valeurs de chaîne du nom de l'action.

Pour plus d'informations, consultez la section Présentation the section called “Création de politiques gérées à partir de JSON” des politiques JSON dans le guide de l'utilisateur IAM.

La classe **`ClientConfig`**

L'Amazon.Runtime. ClientConfigclass est la classe de base des classes de configuration des clients de service telles que AmazonS3Config. Les éléments de programmation suivants ont été supprimés de cette classe.

- Les `DetermineDnsSuffix` méthodes `DetermineServiceURL` et ont été supprimées. Utilisez plutôt la `DetermineServiceOperationEndpoint` méthode du client de service ; par exemple, AmazonS3Client. DetermineServiceOperationEndpoint.
- La `ReadEntireResponse` propriété a été supprimée. Utilisez plutôt l'une des options suivantes :
  - La `LogResponses` propriété du AWSConfigs. LoggingConfigclasse.
  - `LogResponse`Propriété de la configuration du client ; par exemple, AmazonS3Config.

L'espace de noms **`Amazon.Runtime`**

L'espace de noms Amazon.Runtime a été mis à jour comme suit :

- La `ECSTaskCredentials` classe obsolète a été supprimée de l'espace de noms. Utilisez plutôt le GenericContainerCredentialsfournisseur, qui prend également en charge Amazon EKS Pod Identities.

- Les `StoredProfileCredentials` classes obsolètes `StoredProfileAWSCredentials` ont été supprimées de l'espace de noms. Utilisez le SDKCredentialsfichier réseau ou la SharedCredentialsFileclasse d'Amazon.Runtime. CredentialManagementespace de noms à la place.
- La `HasCachedAccessTokenAvailable` méthode obsolète de la SSOAWSCredentialsclasse a été supprimée de l'espace de noms.
- La `EnvironmentAWSCredentials` classe obsolète a été supprimée de l'espace de noms. Utilisez plutôt la classe AppConfigAWSCredentials.
- La `StoredProfileFederatedCredentials` classe obsolète a été supprimée de l'espace de noms. Utilisez plutôt la AWSCredentials classe Federated.
- Les classes obsolètes suivantes ont été supprimées de l'espace de noms :`EnvironmentVariableAWSEndpointDiscoveryEnabled`,`ProfileAWSEndpointDiscovery` et`FallbackEndpointDiscoveryEnabledFactory`.
- La `UseSigV4` propriété obsolète a été supprimée de la AmazonWebServiceRequestclasse. Utilisez plutôt la propriété `SignatureVersion`.
- La `ProfileIniFile` classe de l'espace de `Amazon.Runtime.Internal.Util` noms possède une méthode surchargée appelée. `TryGetSection` Les versions de la méthode qui ne prennent pas en charge le `out` paramètre for `nestedProperties` ont été supprimées de la classe.
- La `EventBridgeSigner` classe obsolète de l'espace de `Amazon.Runtime.Internal.Auth` noms a été supprimée.
- La propriété de `Parameters` dictionnaire obsolète a été supprimée de la WebServiceRequestEventArgsclasse. Utilisez plutôt la propriété `ParameteCollection`.

BouncyCastle

La copie source de BouncyCastle a été supprimée de la version V4 du SDK.

La classe **`StoredProfileSAMLCredentials`**

La `StoredProfileSAMLCredentials` classe obsolète d'Amazon. SecurityTokenL'espace de noms .SAML a été supprimé. Utilisez plutôt la AWSCredentials classe Federated dans l'espace de noms Amazon.Runtime.

La classe **`AWSSDKUtils`**

Les méthodes suivantes ont été supprimées de la AWSSDKUtilsclasse : `ResolveResourcePathProtectEncodedSlashUrlEncode`, et`ConvertToUnixEpochMilliSeconds`.

La classe **`ProfileManager`**

La `ProfileManager` classe obsolète a été supprimée de l'espace de noms Amazon.Util. Utilisez le SDKCredentialsfichier réseau ou la SharedCredentialsFileclasse d'Amazon.Runtime. CredentialManagementespace de noms à la place.

La classe **`AWSConfigs`**

Les propriétés obsolètes suivantes ont été supprimées de la AWSConfigsclasse :`Logging`,`ResponseLogging`, et`LogMetrics`. Utilisez plutôt la propriété `LoggingConfig`.

La classe **`ConditionFactory`**

La méthode avec la signature suivante a été supprimée de la ConditionFactoryclasse :`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Utilisez plutôt la NewConditionUtcméthode.

CloudFront Utilitaires Amazon

L'espace de `Amazon.CloudFront.Util` noms et la `AmazonCloudFrontUtil` classe obsolètes ont été supprimés.

AWS IoT

Dans le ListPrincipalThingsResponsecours, une ancienne personnalisation pour une `NextToken` dérogation a été supprimée au profit de la pagination.

AWS Lambda

Les `Invoke...` méthodes suivantes de la AmazonLambdaClientclasse ont été supprimées car les noms prêtaient à confusion.

- La méthode V3 avec la signature suivante a été supprimée :`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. Il s'agit d'une méthode synchrone dans la version

3 du SDK. Utilisez `InvokeResponse Invoke(InvokeRequest)` plutôt (pour le traitement synchrone) ou `Task InvokeAsync(InvokeRequest, CancellationToken)` (pour le traitement asynchrone).

- La méthode V3 avec la signature suivante a été supprimée :`Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`. Il s'agit d'une méthode asynchrone dans la version 3 du SDK. Utilisez `Task InvokeAsync(InvokeRequest, CancellationToken)` à la place.

Amazon SageMaker Runtime

Les constructeurs obsolètes de la PayloadPartclasse ont été supprimés.

Migration à partir de .NET Standard 1.3

Le 27 juin 2019, Microsoft a mis fin à la prise en charge des versions .NET Core 1.0 et .NET Core 1.1. Suite à cette annonce, le support de .NET Standard 1.3 a AWS pris fin AWS SDK pour .NET le 31 décembre 2020.

AWS a continué à fournir des mises à jour de service et des correctifs de sécurité sur le .NET Standard 1.3 AWS SDK pour .NET cible jusqu'au 1er octobre 2020. Après cette date, la cible .NET Standard 1.3 est passée en mode maintenance, ce qui signifie qu'aucune nouvelle mise à jour n'a été publiée ; seuls des correctifs de bogues critiques et des correctifs de sécurité ont été AWS appliqués.

Le 31 décembre 2020, le support de .NET Standard 1.3 AWS SDK pour .NET a pris fin. Après cette date, aucun correctif de bogue ou correctif de sécurité n'a été appliqué. Les artefacts créés avec cette cible restent disponibles en téléchargement sur NuGet.

Ce que vous devez faire

- Si vous exécutez des applications qui utilisent .NET Framework, vous n'êtes pas concerné.

- Si vous exécutez des applications qui utilisent .NET Core 2.0 ou version ultérieure, vous n'êtes pas concerné.

- Si vous exécutez des applications qui utilisent .NET Core 1.0 ou .NET Core 1.1, migrez vos applications vers une version plus récente de .NET Core en suivant les instructions de migration Microsoft. Nous recommandons au minimum .NET Core 3.1.

- Si vous exécutez des applications métier stratégiques qui ne peuvent pas être mises à niveau pour le moment, vous pouvez continuer à utiliser votre version actuelle du kit AWS SDK pour .NET.

Si vous avez des questions ou des inquiétudes, contactez le support AWS.

Sécurité de ce AWS produit ou service

Chez Amazon Web Services (AWS), la sécurité dans le cloud est la priorité principale. En tant que client AWS , vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses sur la sécurité. La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cela comme la sécurité du cloud et la sécurité dans le cloud.

Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute tous les services proposés dans le AWS cloud et de vous fournir des services que vous pouvez utiliser en toute sécurité. Notre responsabilité en matière de sécurité est notre priorité absolue AWS, et l'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de AWS conformité.

Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez et par d'autres facteurs, notamment la sensibilité de vos données, les exigences de votre organisation et les lois et réglementations applicables.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Rubriques

- Protection des données dans ce AWS produit ou service
- Gestion de l’identité et des accès
- Validation de conformité pour ce AWS produit ou service
- Résilience pour ce AWS produit ou service
- Sécurité de l'infrastructure pour ce AWS produit ou service
- Appliquer une version minimale de TLS dans le AWS SDK pour .NET
- Migration du client de chiffrement S3 (V1 vers V2)
- Migration du client de chiffrement S3 (V2 vers V4)

Protection des données dans ce AWS produit ou service

Le modèle de responsabilité AWS partagée de s'applique à la protection des données dans ce AWS produit ou service. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée d'AWS et RGPD (Règlement général sur la protection des données) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section Utilisation des CloudTrail sentiers dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez Norme FIPS (Federal Information Processing Standard) 140-3.

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec ce AWS produit ou service ou un autre à

Services AWS l'aide de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Gestion de l'identité et des accès

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS les ressources. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l'accès à l'aide de politiques
- Comment Services AWS travailler avec IAM
- Résolution des problèmes AWS d'identité et d'accès

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction du travail que vous effectuez. AWS

Utilisateur du service : si vous avez l' Services AWS habitude de faire votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de nouvelles AWS fonctionnalités pour effectuer votre travail, vous aurez peut-être besoin d'autorisations supplémentaires. Si vous comprenez bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans AWS, consultez Résolution des problèmes AWS d'identité et d'accès le guide de l'utilisateur du Service AWS que vous utilisez.

Administrateur du service — Si vous êtes responsable des AWS ressources de votre entreprise, vous avez probablement un accès complet à AWS. C'est à vous de déterminer les AWS fonctionnalités et les ressources auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs

de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec AWS, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Administrateur IAM – Si vous êtes un administrateur IAM, vous souhaiterez peut-être en savoir plus sur la façon d'écrire des politiques pour gérer l'accès à AWS. Pour consulter des exemples de politiques AWS basées sur l'identité que vous pouvez utiliser dans IAM, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations

d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle utilisateur à un rôle IAM (console) ou en appelant une opération AWS CLI ou AWS API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Vous devez spécifier un principal dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Comment Services AWS travailler avec IAM

Pour obtenir une vue d'ensemble du Services AWS fonctionnement de la plupart des fonctionnalités IAM, consultez les AWS services compatibles avec IAM dans le guide de l'utilisateur IAM.

Pour savoir comment utiliser un service spécifique Service AWS avec IAM, consultez la section sécurité du guide de l'utilisateur du service concerné.

Résolution des problèmes AWS d'identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans AWS
- Je ne suis pas autorisé à effectuer iam : PassRole
- Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Je ne suis pas autorisé à effectuer une action dans AWS

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `awes:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `awes:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter `iam:PassRole` l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à AWS.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans AWS. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si ces fonctionnalités sont prises AWS en charge, consultezComment Services AWS travailler avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.

- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Validation de conformité pour ce AWS produit ou service

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Résilience pour ce AWS produit ou service

L'infrastructure AWS mondiale est construite autour Régions AWS de zones de disponibilité.

Régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant.

Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS mondiale.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Sécurité de l'infrastructure pour ce AWS produit ou service

Ce AWS produit ou service utilise des services gérés et est donc protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section Sécurité du AWS cloud. Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à ce AWS produit ou service via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Ce AWS produit ou service suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Appliquer une version minimale de TLS dans le AWS SDK pour .NET

Pour renforcer la sécurité lors de la communication avec AWS les services, vous devez configurer le AWS SDK pour .NET pour utiliser le protocole TLS 1.2 ou version ultérieure.

AWS SDK pour .NET Utilise le moteur d'exécution .NET sous-jacent pour déterminer le protocole de sécurité à utiliser. Par défaut, les versions actuelles de .NET utilisent le dernier protocole configuré pris en charge par le système d'exploitation. Votre application peut remplacer ce comportement du kit SDK, mais il n'est pas recommandé de le faire.

.NET Core

Par défaut, .NET Core utilise le dernier protocole configuré pris en charge par le système d'exploitation. Le AWS SDK pour .NET ne fournit pas de mécanisme permettant de remplacer cela.

Si vous utilisez une version .NET Core antérieure à 2.1, nous vous recommandons vivement de mettre à niveau votre version .NET Core.

Pour plus d'informations spécifiques à chaque système d'exploitation, reportez-vous à ce qui suit.

Windows

Dans les distributions modernes de Windows la prise en charge de TLS 1.2 est activée par défaut. Si vous utilisez Windows 7 SP1 ou Windows Server 2008 R2 SP1, vous devez vous assurer que le support TLS 1.2 est activé dans le registre, comme décrit dans https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12. Si vous exécutez une distribution antérieure, vous devez mettre à niveau votre système d'exploitation. Pour plus d'informations sur la prise en charge du protocole TLS 1.3 sous Windows, consultez la dernière documentation Microsoft pour connaître les versions minimales requises du client ou du serveur.

macOS

Si vous exécutez .NET Core 2.1 ou version ultérieure, TLS 1.2 est activé par défaut. Le protocole TLS 1.2 est pris en charge par OS X Mavericks 10.9 ou version ultérieure. .NET Core version 2.1 et versions ultérieures nécessitent des versions plus récentes de macOS, comme décrit à l'adresse https://learn.microsoft.com/en-us/dotnet/core/install/windows? tabs=net80&pivots=os-macos.

Si vous utilisez .NET Core 1.0, .NET Core utilise OpenSSL sur macOS, une dépendance qui doit être installée séparément. OpenSSL a ajouté le support pour TLS 1.2 dans la version 1.0.1, et le support pour TLS 1.3 dans la version 1.1.1.

Linux

.NET Core sous Linux nécessite OpenSSL, qui est fourni avec de nombreuses distributions Linux. Mais il peut également être installé séparément. OpenSSL a ajouté le support pour TLS 1.2 dans la

version 1.0.1, et le support pour TLS 1.3 dans la version 1.1.1. Si vous utilisez une version moderne de .NET Core (2.1 ou version ultérieure) et que vous avez installé un gestionnaire de paquets, il est probable qu'une version plus moderne d'OpenSSL a été installée pour vous.

Pour être sûr, vous pouvez exécuter **`openssl version`** dans un terminal et vérifier que la version est postérieure à 1.0.1.

.NET Framework

Si vous exécutez une version moderne de .NET Framework (4.7 ou version ultérieure) et une version moderne de Windows (au moins Windows 8 pour les clients, Windows Server 2012 ou version ultérieure pour les serveurs), TLS 1.2 est activé et utilisé par défaut.

Si vous utilisez un environnement d'exécution .NET Framework qui n'utilise pas les paramètres du système d'exploitation (.NET Framework 3.5 à 4.5.2), il AWS SDK pour .NET tentera d'ajouter la prise en charge de TLS 1.1 et TLS 1.2 aux protocoles pris en charge. Si vous utilisez .NET Framework 3.5, cela ne fonctionnera que si le correctif approprié est installé, comme suit :

- Windows 10 version 1511 et Windows Server 2016 — KB3156421
- Windows 8.1 et Windows Server 2012 R2 — KB3154520
- Windows Server 2012 — KB3154519
- Windows 7 SP1 et Server 2008 R2 SP1 — KB3154518

> **Warning**
>
> Le 15 août 2024, le support de .NET Framework 3.5 AWS SDK pour .NET a été arrêté et la version minimale de .NET Framework a été remplacée par la version 4.7.2. Pour plus d'informations, consultez le billet de blog Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET.

Si votre application s'exécute sur un .NET Framework plus récent sous Windows 7 SP1 ou Windows Server 2008 R2 SP1, vous devez vous assurer que le support TLS 1.2 est activé dans le registre, comme décrit dans https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12. Dans les versions plus récentes de Windows il est activé par défaut.

Pour connaître les meilleures pratiques détaillées relatives à l'utilisation du protocole TLS avec .NET Framework, consultez l'article Microsoft à https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tlsl'adresse.

Outils AWS pour PowerShell

Outils AWS pour PowerShellutilisez le AWS SDK pour .NET pour tous les appels aux AWS services. Le comportement de votre environnement dépend de la version de Windows que PowerShell vous utilisez, comme suit.

Windows PowerShell 2.0 à 5.x

Windows PowerShell 2.0 à 5.x s'exécute sur .NET Framework. Vous pouvez vérifier quel environnement d'exécution .NET (2.0 ou 4.0) est utilisé à PowerShell l'aide de la commande suivante.

```
$PSVersionTable.CLRVersion
```

- Lorsque vous utilisez .NET Runtime 2.0, suivez les instructions fournies précédemment concernant le AWS SDK pour .NET et .NET Framework 3.5.

> ⚠ Warning
>
> Le 15 août 2024, le support de .NET Framework 3.5 AWS SDK pour .NET a été arrêté et la version minimale de .NET Framework a été remplacée par la version 4.7.2. Pour plus d'informations, consultez le billet de blog Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET.

- Lorsque vous utilisez .NET Runtime 4.0, suivez les instructions fournies précédemment concernant AWS SDK pour .NET et .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 et les versions ultérieures s'exécutent sur .NET Core. Vous pouvez vérifier quelle version de .NET Core est utilisée en exécutant la commande suivante.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Suivez les instructions fournies précédemment concernant AWS SDK pour .NET la version appropriée de .NET Core.

Xamarin

Pour Xamarin, consultez les instructions sur -layer-security. https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport Pour résumer :

Pour Android

- Exige Android 5.0 ou version ultérieure.
- Propriétés du projet, options Android : l' HttpClient implémentation doit être définie sur Android et l' SSL/TLS implémentation définie sur Native TLS 1.2+.

Pour iOS

- Exige iOS 7 ou version ultérieure.
- Propriétés du projet, version iOS : l' HttpClientimplémentation doit être définie sur NSUrlSession.

Pour macOS

- Exige macOS 10.9 ou version ultérieure.
- Options du projet, build, build Mac : l' HttpClient implémentation doit être définie sur NSUrlSession.

Unity

Vous devez utiliser Unity 2018.2 ou version ultérieure et utiliser le runtime de script équivalent .NET 4.x. Vous pouvez le définir dans Paramètres du projet, Configuration, Lecteur, comme décrit dans https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html. L'environnement d'exécution de script équivalent .NET 4.x permet la prise en charge du protocole TLS 1.2 sur toutes les plateformes Unity exécutant Mono ou CPP. IL2

Navigateur (pour Blazor WebAssembly)

WebAssembly s'exécute dans le navigateur plutôt que sur le serveur et utilise le navigateur pour gérer le trafic HTTP. Par conséquent, la prise en charge de TLS est déterminée par la prise en charge du navigateur.

Blazor WebAssembly, en version préliminaire d'ASP.NET Core 3.1, n'est pris en charge que dans les navigateurs compatibles WebAssembly, comme décrit dans -platforms. https://learn.microsoft.com/en-us/ aspnet/core/blazor/supported Tous les principaux navigateurs étaient compatibles avec le protocole TLS 1.2 avant de le prendre en charge WebAssembly. Si tel est le cas pour votre navigateur, alors si votre application s'exécute, elle peut communiquer via TLS 1.2.

Consultez la documentation de votre navigateur pour plus d'informations et de vérifications.

Migration du client de chiffrement S3 (V1 vers V2)

> **Note**
>
> Si vous utilisez la version 2 et que vous souhaitez migrer vers la version 4, consultezMigration du client de chiffrement S3 (V2 vers V4).

Cette rubrique explique comment migrer vos applications de la version 1 (V1) du client de chiffrement Amazon Simple Storage Service (S3) vers la version 2 (V2), et comment garantir la disponibilité des applications tout au long du processus de migration.

Les objets chiffrés avec le client V2 ne peuvent pas être déchiffrés avec le client V1. Afin de faciliter la migration vers le nouveau client sans avoir à rechiffrer tous les objets en même temps, un client « V1-Transitional » a été fourni. Ce client peut déchiffrer les objets chiffrés en V1 et en V2, mais chiffre les objets uniquement dans un format compatible avec la version 1. Le client V2 peut déchiffrer les objets chiffrés en V1 et en V2 (lorsqu'il est activé pour les objets V1), mais chiffre les objets uniquement dans un format compatible avec la version v2.

Présentation de la migration

Cette migration s'effectue en trois phases. Ces phases sont présentées ici et décrites en détail ultérieurement. Chaque phase doit être terminée pour tous les clients qui utilisent des objets partagés avant le début de la phase suivante.

1. Mettez à jour les clients existants vers les clients de transition vers la version V1-Transitional afin de lire les nouveaux formats. Tout d'abord, mettez à jour vos applications pour qu'elles dépendent du client V1-Transitional plutôt que du client V1. Le client V1-Transitional permet à votre code existant de déchiffrer les objets écrits par les nouveaux clients V2 et les objets écrits dans un format compatible avec la version 1.

> **Note**
>
> Le client V1-Transitional est fourni à des fins de migration uniquement. Procédez à la mise à niveau vers le client V2 après avoir migré vers le client V1-Transitional.

2. Migrez les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats. Remplacez ensuite tous les clients de transition V1 de vos applications par des clients V2 et définissez le profil de sécurité sur. `V2AndLegacy` La définition de ce profil de sécurité sur les clients V2 permet à ces clients de déchiffrer des objets chiffrés dans un format compatible avec la version 1.
3. Mettez à jour les clients V2 pour qu'ils ne lisent plus les formats V1. Enfin, une fois que tous les clients ont été migrés vers la version V2 et que tous les objets ont été chiffrés ou rechiffrés dans un format compatible avec la version 2, définissez le profil de sécurité V2 sur au lieu de. `V2` `V2AndLegacy` Cela empêche le déchiffrement d'objets au format compatible avec la version 1.

Mettez à jour les clients existants vers des clients de transition vers la version 1 pour lire les nouveaux formats

Le client de chiffrement V2 utilise des algorithmes de chiffrement que les anciennes versions du client ne prennent pas en charge. La première étape de la migration consiste à mettre à jour vos clients de déchiffrement V1 afin qu'ils puissent lire le nouveau format.

Le client V1-Transitional permet à vos applications de déchiffrer les objets chiffrés en V1 et en V2. Ce client fait partie du package NuGet Amazon.Extensions.S3.Encryption. Effectuez les étapes suivantes sur chacune de vos applications pour utiliser le client V1-Transitional.

1. Prenez une nouvelle dépendance vis-à-vis du package Amazon.Extensions.S3.Encryption. Si votre projet dépend directement du AWSSDK.S3 ouAWSSDK. KeyManagementServicepackages, vous devez soit mettre à jour ces dépendances, soit les supprimer afin que leurs versions mises à jour soient intégrées à ce nouveau package.
2. Remplacez l'`using`instruction appropriée par « `Amazon.S3.Encryption` à »`Amazon.Extensions.S3.Encryption`, comme suit :

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. Reconstruisez et redéployez votre application.

Le client V1-Transitional est entièrement compatible avec l'API du client V1, aucune autre modification de code n'est donc requise.

Migrer les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats

Le client V2 fait partie du package NuGet Amazon.Extensions.S3.Encryption. Il permet à vos applications de déchiffrer les objets chiffrés en V1 et en V2 (si cela est configuré pour le faire), mais chiffre les objets uniquement dans un format compatible avec la version v2.

Après avoir mis à jour vos clients existants pour lire le nouveau format de chiffrement, vous pouvez procéder à la mise à jour en toute sécurité de vos applications vers les clients de chiffrement et de déchiffrement V2. Pour utiliser le client V2, effectuez les étapes suivantes sur chacune de vos applications :

1. Remplacez `EncryptionMaterials` par `EncryptionMaterialsV2`
    a. Lorsque vous utilisez KMS :
        i. Fournissez un ID de clé KMS.
        ii. Déclarez la méthode de cryptage que vous utilisez, c'est-à-dire`KmsType.KmsContext`.
        iii. Fournissez un contexte de chiffrement à KMS à associer à cette clé de données. Vous pouvez envoyer un dictionnaire vide (le contexte de chiffrement Amazon sera toujours intégré), mais il est recommandé de fournir un contexte supplémentaire.
    b. Lorsque vous utilisez des méthodes d'encapsulation de clé fournies par l'utilisateur (chiffrement symétrique ou asymétrique) :
        i. Fournissez une `AES` ou une `RSA` instance contenant le matériel de chiffrement.
        ii. Déclarez l'algorithme de chiffrement à utiliser, c'est-à-dire `SymmetricAlgorithmType.AesGcm` ou`AsymmetricAlgorithmType.RsaOaepSha1`.
2. Passez `AmazonS3CryptoConfiguration` à `AmazonS3CryptoConfigurationV2` avec la `SecurityProfile` propriété définie sur`SecurityProfile.V2AndLegacy`.

3. Remplacez `AmazonS3EncryptionClient` par `AmazonS3EncryptionClientV2` Ce client prend les objets récemment convertis `AmazonS3CryptoConfigurationV2` et `EncryptionMaterialsV2` les objets des étapes précédentes.

Exemple : KMS vers KMS+Context

Prémigration

```
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);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new
 EncryptionMaterialsV2("1234abcd-12ab-34cd-56ef-1234567890ab", KmsType.KmsContext,
 encryptionContext);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Exemple : algorithme symétrique (enroulement de clé AES-CBC vers AES-GCM)

`StorageMode` peut avoir la valeur `ObjectMetadata` ou `InstructionFile`.

Prémigration

```
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);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
 SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

> **Note**
>
> Lorsque vous déchiffrez avec AES-GCM, lisez l'objet dans son intégralité avant de commencer à utiliser les données déchiffrées. Cela permet de vérifier que l'objet n'a pas été modifié depuis qu'il a été chiffré.

Exemple : algorithme asymétrique (RSA to RSA-OAEP-SHA 1 Key Wrap)

`StorageMode` peut avoir la valeur `ObjectMetadata` ou `InstructionFile`.

Prémigration

```
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);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterialsV2(asymmetricAlgorithm,
 AsymmetricAlgorithmType.RsaOaepSha1);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Mettre à jour les clients V2 pour ne plus lire les formats V1

À terme, tous les objets auront été chiffrés ou rechiffrés à l'aide d'un client V2. Une fois cette conversion terminée, vous pouvez désactiver la compatibilité V1 dans les clients V2 en définissant la `SecurityProfile` propriété sur`SecurityProfile.V2`, comme indiqué dans l'extrait suivant.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migration du client de chiffrement S3 (V2 vers V4)

> **Note**
>
> Si vous utilisez la V1 et que vous souhaitez migrer vers la V4, vous devez d'abord migrer vers la V2. Consultez Migration du client de chiffrement S3 (V1 vers V2).

Cette rubrique explique comment migrer vos applications de la version 2 (V2) du client de chiffrement Amazon Simple Storage Service (S3) vers la version 4 (V4) et comment garantir la disponibilité des applications tout au long du processus de migration. La V4 utilise l'AES-GCM avec un engagement clé pour le chiffrement du contenu et introduit des politiques d'engagement pour améliorer la sécurité contre les attaques par substitution de clés.

Le client V4 est disponible dans le package NuGet Amazon.Extensions.S3.Encryption.

> **Important**
>
> Changements majeurs : les clients V4 configurés avec une `RequireEncryptRequireDecrypt` politique ne peuvent pas déchiffrer les objets chiffrés avec les clients V1 ou V2. Seuls les derniers clients V2 peuvent déchiffrer les objets chiffrés V4 avec un engagement clé. Avant de passer à cette `RequireEncryptRequireDecrypt` politique, vous devez rechiffrer toutes vos données existantes à l'aide de clients V4 avec activation de la clé d'engagement.

Comprendre les concepts de la V4

La V4 utilise l'AES-GCM avec un engagement clé en matière de chiffrement de contenu et introduit des concepts de sécurité clés qui améliorent la protection de vos données chiffrées :

Politique d'engagement

La politique d'engagement contrôle la manière dont le client de chiffrement gère l'engagement des clés lors des opérations de chiffrement et de déchiffrement. V4 soutient trois politiques d'engagement :

`ForbidEncryptAllowDecrypt`

Chiffrement : sans engagement

Déchiffrement : autorise les objets sans engagement

Sécurité : n'impose pas d'engagement et peut permettre la falsification

Compatibilité : toutes les implémentations V2 et V4 peuvent lire les objets chiffrés avec cette politique

`RequireEncryptAllowDecrypt`

Chiffrement : avec engagement clé

Déchiffrement : autorise à la fois les objets validants et non validants

Sécurité : les nouveaux objets sont protégés contre les attaques de substitution de clés, les anciens objets restent lisibles

Compatibilité : seule la V4 prend en charge cette politique

`RequireEncryptRequireDecrypt`(Par défaut pour la V4)

Chiffrement : avec engagement clé

Déchiffrement : validation d'objets uniquement

Sécurité : respect total des engagements pour une sécurité maximale

Compatibilité : seule la V4 prend en charge cette politique

AES GCM avec un engagement clé

La V4 utilise l'AES-GCM avec un engagement clé pour le chiffrement du contenu, ce qui améliore la sécurité :

- Protection contre la falsification : elle protège contre les attaques de substitution de clés en liant cryptographiquement la clé aux données cryptées.
- Compatibilité des versions : les objets chiffrés avec un engagement clé ne peuvent être déchiffrés que par les clients V4 et les versions plus récentes.

> **Warning**
>
> Avant d'activer le chiffrement par engagement par clé en production, assurez-vous que toutes les applications qui ont besoin de déchiffrer vos objets ont été mises à niveau vers la version V4 ou une version ultérieure, car le client V2 est obsolète.

Mettre à jour les clients existants pour lire les formats V4

Le client de chiffrement V4 utilise des algorithmes de chiffrement que les anciennes versions du client ne prennent pas en charge. La première étape de la migration consiste à mettre à jour vos clients V2 afin qu'ils puissent lire le nouveau format V4.

Mettre à jour les dépendances NuGet du package

Mettez à jour vos applications pour utiliser la dernière version du package Amazon.Extensions.S3.Encryption NuGet qui inclut le support V4. Effectuez les étapes suivantes sur chacune de vos applications :

1. Effectuez la mise à jour vers le dernier package Amazon.Extensions.S3.Encryption. Si votre projet dépend directement du AWSSDK.S3 ouAWSSDK. KeyManagementServicepackages, vous devez soit mettre à jour ces dépendances, soit les supprimer afin que leurs versions mises à jour soient intégrées à ce nouveau package.
2. Assurez-vous que vos `using` instructions font référence au bon espace de noms :

   ```
   using Amazon.Extensions.S3.Encryption;
   using Amazon.Extensions.S3.Encryption.Primitives;
   ```

3. Reconstruisez et redéployez votre application.

Vos clients V2 existants continueront de fonctionner avec le package mis à jour et pourront déchiffrer les objets chiffrés par les clients V4 (en fonction de la politique d'engagement utilisée).

Création et déploiement d'applications

Après avoir mis à jour les dépendances de votre NuGet package :

1. Créez votre application pour vous assurer que toutes les dépendances sont correctement résolues.

2. Testez votre application dans un environnement de développement pour vérifier que les fonctionnalités existantes continuent de fonctionner.
3. Déployez l'application mise à jour dans votre environnement de production.

Cette mise à jour permet à vos clients V2 existants de déchiffrer les objets qui seront chiffrés par les clients V4, garantissant ainsi la compatibilité pendant le processus de migration.

Migrer vers des clients V4

Après avoir mis à jour vos clients existants pour lire le nouveau format de chiffrement, vous pouvez procéder à la mise à jour sécurisée de vos applications pour utiliser les clients de chiffrement et de déchiffrement V4. Le client V4 fournit une sécurité renforcée grâce à un engagement clé tout en maintenant la compatibilité avec les objets chiffrés existants.

Processus de migration en 4 étapes

La migration de la V2 vers la V4 suit un processus structuré en 4 étapes pour garantir la compatibilité et la sécurité. Chaque étape représente une configuration spécifique qui doit être déployée dans toutes vos applications avant de passer à l'étape suivante.

1. Étape 0 : Client V2 (point de départ) - Votre implémentation V2 existante
2. Étape 1 : compatibilité V4 avec V2 - Migrer vers le client V4 tout en conservant un comportement de chiffrement compatible avec la version V2
3. Étape 2 : version V4 avec écriture par clé - Commencez à chiffrer avec un engagement par clé tout en autorisant le déchiffrement des objets existants
4. Étape 3 : V4 avec mise en œuvre complète - Nécessite un engagement clé pour le chiffrement et le déchiffrement

Étape 0 : Client V2 (point de départ)

Cela représente votre configuration client V2 existante. Cette étape montre l'état de départ avant la migration.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
```

```
// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Étape 1 : compatibilité V4 avec V2

Migrez vers le client V4 tout en conservant un comportement identique à celui de la V2. Cette étape utilise une `ForbidEncryptAllowDecrypt` politique pour chiffrer sans engagement et permettre le déchiffrement de tous les objets.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);
```

```
// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportement : chiffre sans engagement, peut déchiffrer à la fois les objets validants et non validants. Comportement identique à celui de la V2.

Étape 2 : V4 avec des écritures d'engagement clés

Commencez à chiffrer avec un engagement clé tout en maintenant la rétrocompatibilité pour le déchiffrement. Cette étape utilise `RequireEncryptAllowDecrypt` la politique.

> **Warning**
>
> Avant de déployer l'étape 2, assurez-vous que tous les lecteurs ont été mis à jour à l'étape 1 ou à une version ultérieure pour gérer le chiffrement par engagement par clé.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
```

```
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportement : chiffre avec engagement, permet de déchiffrer à la fois les objets validants et non validants. Les nouveaux objets sont protégés contre les attaques de substitution de clés.

Étape 3 : V4 avec mise en œuvre complète

Exigez un engagement clé pour le chiffrement et le déchiffrement. Cette étape utilise une `RequireEncryptRequireDecrypt` politique pour une sécurité maximale.

> **Warning**
>
> Avant de déployer l'étape 3, assurez-vous que tous les objets de votre système ont été rechiffrés avec clé d'engagement (étape 2). Cette étape ne permettra pas de déchiffrer les objets chiffrés sans engagement.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
```

```
    Key = objectKey,
    ContentBody = content
});
```

Comportement : chiffre avec engagement, déchiffre uniquement les objets chiffrés avec engagement. Sécurité maximale contre les attaques par substitution de clés.

Exemples de configuration supplémentaires

Cette section fournit des exemples supplémentaires pour configurer les clients V4 avec différentes options lors de la migration.

Permettre le support existant

Pour permettre aux clients V4 de lire les objets chiffrés par les clients V1 et V2, configurez le client avec une politique d'engagement qui autorise le déchiffrement traditionnel :

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Utilisez cette configuration lors de la migration lorsque vous devez déchiffrer des objets chiffrés par des clients plus anciens tout en vous assurant que les nouveaux objets sont chiffrés avec une sécurité renforcée.

Configuration de la méthode de stockage

La V4 prend en charge deux méthodes de stockage pour les métadonnées de chiffrement. Choisissez la méthode qui convient le mieux à votre cas d'utilisation :

Métadonnées d'objet (par défaut)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Fichier d'instructions

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

À utiliser `InstructionFile` lorsque vous devez conserver les métadonnées d'objets à d'autres fins ou lorsque vous travaillez avec des objets dont la taille des métadonnées est limitée.

Configuration de la politique d'engagement

Choisissez la politique d'engagement appropriée en fonction de vos exigences de sécurité et de la phase de migration :

Phase de migration (compatibilité V2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

Phase de transition (recommandée)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

Phase de sécurité complète

```
// For maximum security: require commitment for both encryption and decryption
```

```
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

Commencez par `ForbidEncryptAllowDecrypt` la migration initiale, passez à la phase `RequireEncryptAllowDecrypt` de transition, puis utilisez-la `RequireEncryptRequireDecrypt` lorsque tous les clients ont été mis à niveau et que tous les objets ont été rechiffrés avec engagement.

Considérations spéciales relatives au AWS SDK pour .NET

Cette section contient des considérations relatives aux cas particuliers dans lesquels les configurations ou procédures normales ne sont pas appropriées ou suffisantes.

Rubriques

- Obtention d'assemblages pour AWS SDK pour .NET
- Accès aux informations d'identification et aux profils dans une application
- Considérations spéciales relatives au support Unity
- Considérations spéciales pour le support Xamarin

Obtention d'assemblages pour AWS SDK pour .NET

Cette rubrique décrit comment vous pouvez obtenir les AWSSDK assemblages et les stocker localement (ou sur site) pour les utiliser dans vos projets. Cette méthode n'est pas recommandée pour gérer les références au SDK, mais elle est requise dans certains environnements.

> **Note**
>
> La méthode recommandée pour gérer les références aux SDK consiste à télécharger et à installer uniquement les NuGet packages dont chaque projet a besoin. Cette méthode est décrite dansInstallez AWSSDK des packages avec NuGet.

Si vous ne pouvez pas ou n'êtes pas autorisé à télécharger et à installer des NuGet packages par projet, les options suivantes sont à votre disposition.

Téléchargez et extrayez des fichiers ZIP

(N'oubliez pas que ce n'est pas la méthode recommandée pour gérer les références à AWS SDK pour .NET.)

1. Téléchargez l'un des fichiers ZIP suivants :

   - aws-sdk-net8.0.zip - Assemblages compatibles avec .NET 8 et versions ultérieures.
   - aws-sdk-netcoreapp3.1.zip - Assemblages compatibles avec .NET Core 3.1 et versions ultérieures.

- aws-sdk-netstandard2.0.zip - Assemblages compatibles avec .NET Standard 2.0 et 2.1.
- aws-sdk-net472.zip : assemblages compatibles avec .NET Framework 4.5 et versions ultérieures.

2. Extrayez les assemblages dans un dossier de « téléchargement » de votre système de fichiers, peu importe où. Prenez note de ce dossier.
3. Lorsque vous configurez votre projet, vous obtenez les assemblages requis dans ce dossier, comme décrit dansInstallez AWSSDK des assemblages sans NuGet.

Accès aux informations d'identification et aux profils dans une application

La méthode préférée pour utiliser les informations d'identification est de les autoriser AWS SDK pour .NET à les rechercher et à les récupérer pour vous, comme décrit dansRésolution des informations d'identification et des profils.

Toutefois, vous pouvez également configurer votre application pour récupérer activement les profils et les informations d'identification, puis utiliser explicitement ces informations d'identification lors de la création d'un client de AWS service.

Pour récupérer activement les profils et les informations d'identification, utilisez les classes d'Amazon.Runtime. CredentialManagementespace de noms.

- Pour rechercher un profil dans un fichier utilisant le format de fichier AWS d'informations d'identification (soit le fichier AWS d'informations d'identification partagé dans son emplacement par défaut, soit un fichier d'informations d'identification personnalisé), utilisez la SharedCredentialsFileclasse. Les fichiers de ce format sont parfois simplement appelés fichiers d'identification dans ce texte par souci de concision.
- Pour rechercher un profil dans le SDK Store, utilisez la classe Net SDKCredentials File.
- Pour effectuer une recherche à la fois dans un fichier d'informations d'identification et dans le SDK Store, en fonction de la configuration d'une propriété de classe, utilisez la CredentialProfileStoreChainclasse.

  Vous pouvez utiliser cette classe pour rechercher des profils. Vous pouvez également utiliser cette classe pour demander des AWS informations d'identification directement au lieu d'utiliser la `AWSCredentialsFactory` classe (décrite ci-dessous).

- Pour récupérer ou créer différents types d'informations d'identification à partir d'un profil, utilisez la classe AWSCredentialsFactory.

Les sections suivantes fournissent des exemples pour ces classes.

Exemples de cours CredentialProfileStoreChain

Vous pouvez obtenir des informations d'identification ou des profils auprès de la CredentialProfileStoreChainclasse à l'aide TryGetProfiledes méthodes TryGetAWSCredentialsor. La `ProfilesLocation` propriété de la classe détermine le comportement des méthodes, comme suit :

- Si la valeur `ProfilesLocation` est nulle ou vide, recherchez le SDK Store si la plateforme le prend en charge, puis recherchez le fichier AWS d'informations d'identification partagé à l'emplacement par défaut.
- Si la `ProfilesLocation` propriété contient une valeur, recherchez le fichier d'informations d'identification spécifié dans la propriété.

Obtenez des informations d'identification depuis le SDK Store ou le fichier AWS d'informations d'identification partagé

Cet exemple vous montre comment obtenir des informations d'identification à l'aide de la `CredentialProfileStoreChain` classe, puis les utiliser pour créer un objet AmazonS3Client. Les informations d'identification peuvent provenir du SDK Store ou du fichier d' AWS informations d'identification partagé à l'emplacement par défaut.

Cet exemple utilise également le logiciel Amazon.Runtime. AWSCredentialsclasse.

```
var chain = new CredentialProfileStoreChain();
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("some_profile", out awsCredentials))
{
    // Use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Obtenir un profil depuis le SDK Store ou le fichier d' AWS informations d'identification partagé

Cet exemple montre comment obtenir un profil à l'aide de la CredentialProfileStoreChain classe. Les informations d'identification peuvent provenir du SDK Store ou du fichier d' AWS informations d'identification partagé à l'emplacement par défaut.

Cet exemple utilise également la CredentialProfileclasse.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Obtenir des informations d'identification à partir d'un fichier d'informations d'identification personnalisé

Cet exemple montre comment obtenir des informations d'identification à l'aide de la CredentialProfileStoreChain classe. Les informations d'identification proviennent d'un fichier qui utilise le format de fichier AWS d'informations d'identification mais qui se trouve à un autre emplacement.

Cet exemple utilise également le logiciel Amazon.Runtime. AWSCredentialsclasse.

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials to create an AWS service client
}
```

Exemples pour les classes SharedCredentialsFile et AWSCredentials Factory

Créez un client Amazon S3 à l'aide de la classe SharedCredentialsFile

Cet exemple vous montre comment rechercher un profil dans le fichier d' AWS informations d'identification partagé, créer des informations AWS d'identification à partir du profil, puis

utiliser les informations d'identification pour créer un objet AmazonS3Client. L'exemple utilise la SharedCredentialsFileclasse.

Cet exemple utilise également la CredentialProfileclasse et le fichier Amazon.Runtime.AWSCredentialsclasse.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
 awsCredentials))
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

> **Note**
>
> La classe Net SDKCredentials File peut être utilisée exactement de la même manière, sauf que vous instanciez un nouvel objet Net SDKCredentials File au lieu d'un SharedCredentialsFile objet.

Considérations spéciales relatives au support Unity

Lorsque vous utilisez .NET Standard 2.0 pour votre application Unity, celle-ci doit référencer les AWS SDK pour .NET assemblys (fichiers DLL) directement au lieu d'utiliser NuGet. AWS SDK pour .NET Compte tenu de cette exigence, vous devrez effectuer les actions importantes suivantes.

- Vous devez obtenir les AWS SDK pour .NET assemblages et les appliquer à votre projet. Pour plus d'informations sur la procédure à suivre, reportez-vous Téléchargez et extrayez des fichiers ZIP à la rubriqueObtention d' AWSSDK assemblages.

- Vous devez inclure les éléments suivants DLLs dans votre projet Unity en plus de DLLs for AWSSDK.Core et des autres AWS services que vous utilisez. À partir de la version 3.5.109 du AWS SDK pour .NET, le fichier ZIP standard .NET contient ces informations supplémentaires. DLLs
  - Microsoft Bcl. AsyncInterfaces.dll
  - Système. Runtime. CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- Si vous utilisez IL2CPP pour créer votre projet Unity, vous devez ajouter un `link.xml` fichier dans votre dossier Asset pour empêcher le découpage du code. Le `link.xml` fichier doit répertorier tous les AWSSDK assemblages que vous utilisez, et chacun doit inclure l'`preserve="all"`attribut. L'extrait suivant montre un exemple de ce fichier.

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> Pour lire des informations générales intéressantes relatives à cette exigence, consultez l'article sur https://aws.amazon.com/blogs/developer/ referencing-the-aws-sdk - for-net-standard -2-0- -uwp/. from-unity-xamarin-or

Considérations spéciales pour le support Xamarin

Les projets Xamarin (nouveaux et existants) doivent cibler .NET Standard 2.0. Consulter Prise en charge de .NET Standard 2.0 dans Xamarin.Forms et Prise en charge de l'implémentation .NET.

Consultez également les informations sur Portable Class Library et Xamarin.

Référence d'API pour AWS SDK pour .NET

AWS SDK pour .NET fournit une API que vous pouvez utiliser pour accéder aux AWS services. Pour connaître les classes et les méthodes disponibles dans l'API, consultez la référence de l'AWS SDK pour .NET API.

Outre la référence générale donnée ci-dessus, chacun des exemples de Exemples de code guidés cette section contient des références aux méthodes et classes spécifiques utilisées dans cet exemple.

À propos des versions de référence de l'API

La référence d'API décrite précédemment concerne les versions 3.0 et ultérieures du AWS SDK pour .NET.

Pour plus d'informations sur la migration à partir d'anciennes versions du SDK, voir Migration de votre projet

Pour trouver le contenu obsolète des versions antérieures de la référence d'API du SDK, consultez les éléments suivants :

- AWS SDK pour .NET API Reference V1 (obsolète)
- AWS SDK pour .NET API Reference V2 (obsolète)

Pour afficher une référence d' AWS SDK pour .NET API obsolète, vous devez l'extraire et configurer un navigateur Web. Les instructions ci-dessous sont des exemples illustrant la procédure à suivre. Ils sont basés sur l'utilisation du navigateur Web Google Chrome sur un système Windows. Adaptez-les à votre navigateur Web et à votre système d'exploitation spécifiques.

Afficher la référence d'API obsolète V1

1. Téléchargez le fichier ZIP de la version obsolète de l' AWS SDK pour .NET API Reference V1. Par défaut, le nom du fichier ZIP est`sdkfornet-api-ref_v1_deprecated.zip`. Ce nom sera utilisé tout au long de ces instructions.
2. Placez le fichier ZIP dans le dossier de votre choix. Pour ces instructions, le nom du dossier est supposé être`C:\work\temp\api-refs\V1`.

3. Cliquez avec le bouton droit sur le fichier ZIP et choisissez Extraire tout. Acceptez l'emplacement par défaut, qui correspond `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` à ces instructions.
4. Créez un raccourci pour Google Chrome dans le `C:\work\temp\api-refs\V1` dossier. Veillez à ne pas déplacer l'application d'origine.
5. Dans les propriétés du nouveau raccourci, définissez les champs suivants :

   - Cible : `"<path to the Google Chrome application>" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"`

     Pour l'`--user-data-dir`argument, utilisez un nom de dossier adapté à votre environnement. Il n'est pas nécessaire que le dossier existe.
   - Commencez dans : `C:\work\temp\api-refs\V1`
6. Donnez un nom approprié au raccourci.
7. Ouvrez le raccourci pour afficher l'ancienne référence d'API.

Afficher la référence d'API obsolète V2

1. Téléchargez le fichier ZIP de la version obsolète de l' AWS SDK pour .NET API Reference V2. Par défaut, le nom du fichier ZIP est`sdkfornet-api-ref_v2_deprecated.zip`. Ce nom sera utilisé tout au long de ces instructions.
2. Placez le fichier ZIP dans le dossier de votre choix. Pour ces instructions, le nom du dossier est supposé être`C:\work\temp\api-refs\V2`.
3. Cliquez avec le bouton droit sur le fichier ZIP et choisissez Extraire tout. Acceptez l'emplacement par défaut, qui correspond `C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2` à ces instructions.
4. Créez un raccourci pour Google Chrome dans le `C:\work\temp\api-refs\V2` dossier. Veillez à ne pas déplacer l'application d'origine.
5. Dans les propriétés du nouveau raccourci, définissez les champs suivants :

   - Cible : `"<path to the Google Chrome application>" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"`

Pour l'`--user-data-dir`argument, utilisez un nom de dossier adapté à votre environnement. Il n'est pas nécessaire que le dossier existe.

- Commencez dans : `C:\work\temp\api-refs\V2`

6. Donnez un nom approprié au raccourci.
7. Ouvrez le raccourci pour afficher l'ancienne référence d'API.

Historique du document

Le tableau suivant décrit les modifications importantes apportées depuis la dernière version du Guide du AWS SDK pour .NET développeur. Pour recevoir les notifications concernant les mises à jour de cette documentation, abonnez-vous à un flux RSS.

Modification	Description	Date
Quoi de neuf	Ajout d'une note concernant la version 2.0 de l'outil de AWS déploiement pour .NET CLI.	17 octobre 2025
AWS OpsWorks	AWS OpsWorks a atteint la fin de sa vie. Il a été retiré de la documentation.	16 septembre 2025
Quoi de neuf	End-of-support a été annoncé pour la V3 du AWS SDK pour .NET.	21 août 2025
Quoi de neuf	Disponibilité générale pour le fournisseur de cache distribué	2 juillet 2025
Cadre de traitement des messages	Le cadre de traitement des AWS messages pour .NET est désormais disponible pour le grand public !	5 mai 2025
Quoi de neuf	La version 4 du AWS SDK pour .NET est généralem ent disponible ! Pour plus d'informations sur la migration de vos applications vers la version 4, consultez la section Migration vers la version 4.	28 avril 2025
Migration vers la version 4 du AWS SDK pour .NET	Informations d'aperçu incluses sur la migration des applicati	3 mars 2025

	ons créées avec AWS SDK pour .NET la version 4.	
Intégration AWS à .NET Aspire dans AWS SDK pour .NET	Vous trouverez des informations sur la façon dont l'intégration à .NET Aspire AWS a été intégrée dans le AWS SDK pour .NET.	15 février 2025
Observabilité	Annonce de la sortie GA pour l'observabilité	10 février 2025
Quoi de neuf	Ajout d'informations sur le nouveau comportement par défaut pour la protection de l'intégrité.	15 janvier 2025
Migration du client de chiffrement S3 V4	Ajout d'informations sur la migration de la V2 vers la V4.	9 décembre 2024
Quoi de neuf	Ajout d'informations sur la quatrième version préliminaire de la AWS SDK pour .NET version 4.	15 novembre 2024