
AWS Lambda

Manuel du développeur



AWS Lambda: Manuel du développeur

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Qu'est-ce qu'AWS Lambda ?	1
Quand dois-je utiliser AWS Lambda ?	1
Vous utilisez AWS Lambda pour la première fois ?	2
Démarrez	3
Blocs de construction d'une application Lambda	3
Outils permettant de créer et de tester les fonctions Lambda	3
Configuration d'un compte AWS	4
Configuration d'un compte AWS et création d'un utilisateur administrateur	4
Créer une fonction	6
Appeler la fonction Lambda et vérifier les résultats, les journaux et les métriques	9
Configuration de l'AWS CLI	12
Installation de l'interface de ligne de commande SAM	12
Fonctions Lambda	14
Création de fonctions Lambda	14
Création du code pour votre fonction Lambda	14
Déploiement du code et création d'une fonction Lambda	16
Surveillance et dépannage	17
Exemples d'applications basées sur AWS Lambda	17
Éditeur de code	17
Utilisation de fichiers et de dossiers	18
Utilisation du code	19
Utilisation de la barre de menus	22
Utilisation du mode plein écran	23
Utilisation des préférences	23
Utilisation des commandes	23
Modèle de programmation	24
Package de déploiement	25
Stratégies d'autorisation sur les packages de déploiement Lambda	26
Accès aux ressources AWS	27
Pour accéder aux services AWS	27
Accès aux services non AWS	27
Accès aux services privés ou aux ressources	27
Configuration des fonctions	29
Configuration de base	29
Variables d'environnement	30
Configuration	31
Règles relatives à l'attribution de noms pour les variables d'environnement	33
Variables d'environnement et gestion des versions des fonctions	33
Chiffrement des variables d'environnement	33
Création d'une fonction Lambda à l'aide des variables d'environnement	35
Création d'une fonction Lambda à l'aide de variables d'environnement pour stocker les informations sensibles	37
Gestion des versions	38
Gestion des versions	40
Alias	43
Stratégies de ressources	51
Gestion des versions	52
Déplacement du trafic à l'aide des alias	54
Couches	56
Configuration d'une fonction pour utiliser les couches	57
Gestion des couches	58
Inclusion de dépendances de bibliothèques dans une couche	59
Autorisations d'utilisation des couches	60
Paramètres VPC	61

Configuration d'une fonction Lambda pour l'accès à un Amazon VPC	61
Accès Internet pour les fonctions Lambda	62
Consignes pour la configuration de fonctions Lambda basées sur un VPC	63
Didacticiel : Amazon ElastiCache	63
Didacticiel : Amazon RDS	66
Appel de fonctions	70
Exemple 1	71
Exemple 2	72
Exemple 2	73
Types d'appel	74
Mappage de source d'événement	74
Mappage de source d'événement pour les services AWS	75
Mappage de source d'événement pour les services basés sur les interrogations	76
Mappage de source d'événement pour les applications personnalisées	77
Comportement de nouvelle tentative	78
Mise à l'échelle	79
Débit de demandes	80
Dimensionnement automatique	80
Files d'attente Lettre Morte	81
Sources d'événements prises en charge	82
Amazon S3	84
Amazon DynamoDB	84
Amazon Kinesis Data Streams	84
Amazon Simple Notification Service	85
Amazon Simple Email Service	85
Amazon Simple Queue Service	86
Amazon Cognito	86
AWS CloudFormation	86
Amazon CloudWatch Logs	87
Amazon CloudWatch Events	87
AWS CodeCommit	88
Événements planifiés (via Amazon CloudWatch Events)	88
AWS Config	88
Amazon Alexa	89
Amazon Lex	89
Amazon API Gateway	89
Bouton AWS IoT	90
Amazon CloudFront	90
Amazon Kinesis Data Firehose	90
Autres sources d'événements : appel d'une fonction Lambda à la demande	91
Echantillon de données d'événement	91
Runtimes Lambda	103
Environnement d'exécution	104
Variables d'environnement disponibles pour les fonctions Lambda	104
Contexte d'exécution	105
stratégie de prise en charge de l'exécution	106
Runtimes personnalisés	107
Utilisation d'un runtime personnalisé	107
Création d'un runtime personnalisé	107
Interface de runtime	109
Appel suivant	109
Réponse d'appel	110
Erreur d'appel	110
Erreur d'initialisation	110
Didacticiel – Runtime personnalisé	111
Prérequis	111
Créer une fonction	111

Créer une couche	113
Mise à jour de la fonction	114
Mise à jour du runtime	115
Partage de la couche	115
Nettoyage	116
Applications Lambda	117
Gérer des applications	117
Surveillance des applications	118
Tableaux de bord de surveillance personnalisés	118
AWS SAM	120
Création d'un pipeline pour votre application sans serveur	120
Étape 1 : Configuration de votre référentiel	122
Étape 2 : Créer votre pipeline	124
Étape 3 : Mettre à jour la stratégie de service générée	124
Étape 4 : Exécuter votre étape de déploiement bêta	125
Cas d'utilisation	126
Amazon S3	126
Didacticiel	128
Exemple de code	135
Exemple de modèle	141
Kinesis	142
Configuration de votre fonction et de votre flux de données	143
Création d'un mappage de source d'événement	143
API de mappage de la source d'événement	144
Autorisations du rôle d'exécution	145
Indicateurs Amazon CloudWatch	146
Didacticiel	146
Exemple de code	149
Exemple de modèle	152
Amazon SQS	153
Configuration d'une file d'attente à utiliser avec Lambda	154
Configuration d'une file d'attente en tant que source d'événement	154
Autorisations du rôle d'exécution	155
Didacticiel	155
Exemple de code	158
Exemple de modèle	161
Amazon DynamoDB	162
Création d'un mappage de source d'événement	163
Autorisations du rôle d'exécution	164
Didacticiel	164
Exemple de code	169
Exemple de modèle	172
AWS CloudTrail	173
Didacticiel	174
Exemple de code	180
Amazon SNS	181
Didacticiel	182
Exemple de code	184
Amazon API Gateway	186
Didacticiel	187
Exemple de code	196
Microservice Blueprint	198
Exemple de modèle	200
Application Android	200
Didacticiel	202
Exemple de code	208
CloudWatch Events	209

Didacticiel	210
Exemple de modèle	213
Expressions de planification	214
Equilibreur de charge d'application	216
AWS CLI	217
Prérequis	217
Créer le rôle d'exécution	218
Créer la fonction	218
Répertorier les fonctions Lambda dans votre compte	219
Nettoyage	221
Lambda@Edge	221
Surveillance	223
Utiliser Amazon CloudWatch	223
Scénarios de résolution des problèmes AWS Lambda	223
Accès aux métriques CloudWatch	225
Accédez aux journaux CloudWatch	227
Métriques	228
Utilisation d'AWS X-Ray	231
Suivi des applications basées sur Lambda avec AWS X-Ray	231
Configuration de AWS X-Ray avec Lambda	233
Emission des segments de suivi depuis une fonction Lambda	234
Démon AWS X-Ray dans l'environnement Lambda	235
Utilisation des variables d'environnement pour communiquer avec AWS X-Ray	235
Suivis Lambda dans la console AWS X-Ray : exemples	235
Administration	238
Autorisations	238
Authentification	238
Contrôle d'accès	240
Présentation de la gestion de l'accès	240
Utilisation des stratégies basées sur une identité (stratégies IAM)	244
Utilisation de stratégies basées sur une ressource (stratégies de fonction Lambda)	257
Modèle d'autorisations	260
Référence des autorisations d'API Lambda	263
Modèles de stratégie	265
CloudTrail	272
Informations AWS Lambda dans CloudTrail	273
Présentation des entrées des fichiers journaux AWS Lambda	274
Utilisation de CloudTrail pour suivre les appels de fonctions	275
Balisage	275
Balisage des fonctions Lambda pour la facturation	276
Application des balises aux fonctions Lambda	276
Filtrage sur les fonctions Lambda balisées	278
Restrictions liées aux balises	279
Gestion de la simultanéité	279
Limite des exécutions simultanées au niveau du compte	280
Limite des exécutions simultanées au niveau de la fonction	280
Comportement de limitation	282
Surveillance de votre utilisation de la simultanéité	283
Bonnes pratiques	283
Code de fonction	284
Configuration de fonctions	285
Alarmes et métriques	285
Appels d'événements de flux	286
Appels asynchrones	286
VPC Lambda	286
Limites	288
Utilisation de Node.js	289

Package de déploiement	289
Gestionnaire	291
Utilisation du paramètre de rappel	291
exemple	292
Contexte	293
Journalisation	294
Recherche des journaux	296
Erreurs	296
Gestion des erreurs de fonction	298
Suivi	299
Travail avec Python	301
Package de déploiement	301
Sans dépendances supplémentaires	302
Avec des dépendances supplémentaires	302
Avec un environnement virtuel	303
Gestionnaire	305
Contexte	306
Journalisation	307
Recherche des journaux	308
Erreurs	309
Gestion des erreurs de fonction	310
Suivi	311
Travail avec Java	314
Package de déploiement	315
Création d'un package de déploiement .jar via Maven sans IDE (Java)	315
Création d'un package de déploiement .jar via Maven et l'IDE Eclipse (Java)	317
Création d'un package de déploiement .zip (Java)	320
Création de fonctions Lambda à l'aide de l'IDE Eclipse et du plug-in AWS SDK (Java)	322
Gestionnaire	322
Résolution de la surcharge du gestionnaire	324
Informations supplémentaires	324
Types d'entrée et de sortie du gestionnaire (Java)	324
Utilisation des interfaces prédéfinies pour la création d'un gestionnaire (Java)	329
Contexte	333
Journalisation	334
Appender personnalisé pour Log4j™ 2	334
LambdaLogger.log()	334
Comment trouver les journaux	335
Exemples de journalisation (Java)	335
Erreurs	338
Gestion des erreurs de fonction	339
Suivi	340
Exemple de fonction	342
Utilisation de Go	344
Package de déploiement	344
Création d'un package de déploiement sous Windows	345
Gestionnaire	345
Gestionnaire	347
Utilisation de l'état global	348
Contexte	349
Accès aux informations du contexte d'appel	349
Journalisation	351
Recherche des journaux	352
Erreurs	352
Gestion des erreurs de fonction	310
Gestion des erreurs inattendues	353
Suivi	354

Installation du kit SDK X-Ray pour Go	354
Configuration du kit SDK X-Ray pour Go	354
Création d'un sous-segment	355
Capture	355
Suivi des demandes HTTP	355
Variables d'environnement	355
Utilisation de C#	357
Package de déploiement	358
Interface de ligne de commande .NET Core	358
AWS Toolkit for Visual Studio	365
Gestionnaire	367
Gestion des flux	367
Gestion des types de données standard	368
Signatures de gestionnaire	369
Restrictions liées au gestionnaire de la fonction Lambda	370
Utilisation des fonctions asynchrones dans C# avec AWS Lambda	370
Contexte	371
Journalisation	372
Comment trouver les journaux	373
Erreurs	373
Gestion des erreurs de fonction	375
Utilisation de PowerShell	377
Package de déploiement	377
Configuration d'un environnement de développement PowerShell	378
Utilisation du module AWSLambdaPSCore	378
Gestionnaire	380
Renvoi de données	381
Contexte	382
Journalisation	382
Comment trouver les journaux	382
Erreurs	383
Gestion des erreurs de fonction	384
Utilisation de Ruby	385
Gestionnaire	386
Package de déploiement	387
Mise à jour d'une fonction sans dépendances	388
Mise à jour d'une fonction avec dépendances supplémentaires	388
Contexte	389
Journalisation	390
Erreurs	390
Référence API	392
Actions	392
AddLayerVersionPermission	394
AddPermission	398
CreateAlias	402
CreateEventSourceMapping	406
CreateFunction	410
DeleteAlias	418
DeleteEventSourceMapping	420
DeleteFunction	423
DeleteFunctionConcurrency	425
DeleteLayerVersion	427
GetAccountSettings	429
GetAlias	431
GetEventSourceMapping	434
GetFunction	437
GetFunctionConfiguration	440

GetLayerVersion	445
GetLayerVersionPolicy	448
GetPolicy	450
Invoke	452
InvokeAsync	457
ListAliases	459
ListEventSourceMappings	462
ListFunctions	465
ListLayers	468
ListLayerVersions	470
ListTags	473
ListVersionsByFunction	475
PublishLayerVersion	478
PublishVersion	482
PutFunctionConcurrency	488
RemoveLayerVersionPermission	491
RemovePermission	493
TagResource	496
UntagResource	498
UpdateAlias	500
UpdateEventSourceMapping	504
UpdateFunctionCode	508
UpdateFunctionConfiguration	514
Data Types	521
AccountLimit	522
AccountUsage	523
AliasConfiguration	524
AliasRoutingConfiguration	526
Concurrency	527
DeadLetterConfig	528
Environment	529
EnvironmentError	530
EnvironmentResponse	531
EventSourceMappingConfiguration	532
FunctionCode	534
FunctionCodeLocation	535
FunctionConfiguration	536
Layer	540
LayersListItem	541
LayerVersionContentInput	542
LayerVersionContentOutput	543
LayerVersionsListItem	544
TracingConfig	546
TracingConfigResponse	547
VpcConfig	548
VpcConfigResponse	549
Erreurs de certificat lors de l'utilisation d'un kit SDK	549
Versions	551
Mises à jour antérieures	553
Glossaire AWS	561

Qu'est-ce qu'AWS Lambda ?

AWS Lambda est un service informatique qui vous permet d'exécuter du code sans nécessiter la mise en service ni la gestion de serveurs. AWS Lambda exécute le code uniquement lorsque cela est nécessaire et s'adapte automatiquement, qu'il s'agisse de quelques demandes par jour ou de milliers par seconde. Vous payez uniquement le temps de calcul utilisé et ne déboursez rien quand votre code ne s'exécute pas. Avec AWS Lambda, vous pouvez exécuter le code pour quasiment n'importe quel type d'application ou service backend, sans avoir à vous préoccuper de leur administration. AWS Lambda exécute le code sur une infrastructure de calcul à haute disponibilité et effectue toute l'administration des ressources de calcul, y compris la maintenance des serveurs et du système d'exploitation, le dimensionnement des capacités et la mise à l'échelle automatique, ainsi que la surveillance et la journalisation du code. Il vous suffit de fournir le code dans l'un des langages compatibles avec AWS Lambda (actuellement Node.js, Java, C# , Go et Python).

Vous pouvez utiliser AWS Lambda pour exécuter votre code en réponse à des événements, tels que des modifications apportées aux données d'un compartiment Amazon S3 ou d'une table Amazon DynamoDB, afin d'exécuter votre code en réponse à des requêtes HTTP à l'aide d'Amazon API Gateway, ou d'appeler votre code à l'aide des appels de l'API via les kits SDK AWS. Avec ces fonctionnalités, vous pouvez utiliser Lambda pour créer facilement des déclencheurs de traitement des données pour les services AWS tels qu'Amazon S3 et Amazon DynamoDB, pour traiter les données de streaming stockées dans Kinesis ou pour créer vos propres services dorsaux, tout en bénéficiant du dimensionnement, des performances et de la sécurité d'AWS.

Vous pouvez également créer des applications [sans serveur](#) composées de fonctions déclenchées par des événements et les déployer automatiquement à l'aide d'AWS CodePipeline et d'AWS CodeBuild. Pour plus d'informations, consultez [Applications AWS Lambda \(p. 117\)](#).

Pour plus d'informations sur l'environnement d'exécution AWS Lambda, consultez [Environnement d'exécution Lambda et bibliothèques disponibles \(p. 104\)](#). Pour découvrir comment AWS Lambda détermine les ressources de calcul requises pour exécuter le code, consultez [Configuration de base d'une fonction AWS Lambda \(p. 29\)](#).

Quand dois-je utiliser AWS Lambda ?

AWS Lambda est une plateforme de calcul qui répond aux besoins de nombreux scénarios d'application, à condition que vous soyez en mesure d'écrire le code de votre application dans des langages pris en charge par AWS Lambda (c'est-à-dire, Node.js, Java, Go, C# et Python) et que vous utilisiez l'environnement d'exécution standard AWS Lambda et les ressources fournies par Lambda.

Lorsque vous utilisez AWS Lambda, vous êtes uniquement responsable du code. AWS Lambda gère le parc d'instances de calcul qui assure l'équilibre des ressources de mémoire, d'UC, réseau et autres. C'est le prix de la flexibilité. Autrement dit, vous ne pouvez pas vous connecter aux instances de calcul ni personnaliser le système d'exploitation ou l'environnement Common Language Runtime. Ces contraintes permettent à AWS Lambda d'effectuer des activités opérationnelles et administratives en votre nom, y compris l'adaptation de la capacité, la vérification de l'état des instances, l'application des correctifs de sécurité, le déploiement du code, ainsi que la surveillance et la journalisation des fonctions Lambda.

Si vous avez besoin de gérer vos propres ressources de calcul, Amazon Web Services propose également d'autres services à ces fins.

- Le service Amazon Elastic Compute Cloud (Amazon EC2) combine la flexibilité avec un large éventail de types d'instance EC2 disponibles. Il vous offre la possibilité de personnaliser les systèmes d'exploitation, les paramètres de sécurité et de réseau, ainsi que l'intégralité de la pile de logiciels. Toutefois, vous

êtes chargé de l'allocation de la capacité, de la vérification de l'état des instances, de la surveillance des performances et de l'utilisation de zones de disponibilité pour gérer la tolérance aux pannes.

- Elastic Beanstalk offre un service simple d'utilisation pour déployer et dimensionner dans Amazon EC2 les applications dans lesquelles vous conservez la propriété et le contrôle total des instances EC2 sous-jacentes.

Lambda est un service à haute disponibilité. Pour plus d'informations, consultez le [contrat de niveau de service \(SLA\) AWS Lambda](#).

Vous utilisez AWS Lambda pour la première fois ?

Si vous utilisez AWS Lambda pour la première fois, nous vous recommandons de lire les sections suivantes dans l'ordre :

1. Lisez la présentation du produit et regardez la vidéo d'introduction pour comprendre les exemples de cas d'utilisation. Ces ressources sont disponibles sur la [page web d'AWS Lambda](#).
2. Passez en revue la section [Fonctions Lambda \(p. 14\)](#) de ce guide. Pour comprendre le modèle de programmation et les options de déploiement d'une fonction Lambda, vous devez vous familiariser avec un certain nombre de concepts. Cette section décrit ces concepts et explique comment ils fonctionnent dans les différents langages que vous pouvez utiliser pour créer le code de votre fonction Lambda.
3. Effectuez l'exercice de mise en route basée sur la console. Cet exercice fournit des instructions pour créer et tester votre première fonction Lambda à l'aide de la console. Vous y découvrez également les plans fournis par la console pour créer rapidement des fonctions Lambda. Pour plus d'informations, consultez [Mise en route \(p. 3\)](#).
4. Lisez la section [Déploiement d'applications à l'aide d'AWS Lambda \(p. 117\)](#) de ce guide. Cette section présente les différents composants AWS Lambda que vous utilisez pour créer un environnement complet.

Au-delà de l'exercice de mise en route, vous pouvez explorer les différents cas d'utilisation. Chaque cas d'utilisation est fourni avec un didacticiel qui présente un exemple de scénario. En fonction des besoins de votre application (par exemple, si vous voulez appeler les fonctions Lambda à la demande ou sur la base d'événements), vous pouvez suivre les didacticiels spécifiques. Pour plus d'informations, consultez [Cas d'utilisation \(p. 126\)](#).

Les rubriques suivantes fournissent des informations supplémentaires sur AWS Lambda :

- [Versions et alias des fonctions AWS Lambda \(p. 38\)](#)
- [Utiliser Amazon CloudWatch \(p. 223\)](#)
- [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 283\)](#)
- [Limites AWS Lambda \(p. 288\)](#)

Mise en route

Vous trouverez dans cette section une présentation des concepts de base d'une application Lambda type, ainsi que les options disponibles pour créer et tester des applications. Vous y trouverez en outre des instructions sur l'installation des outils nécessaires pour exécuter les didacticiels inclus dans ce guide et pour créer votre première fonction Lambda.

Blocs de construction d'une application Lambda

- Fonction – Un script ou programme qui s'exécute dans AWS Lambda. Lambda transmet les événements d'appels à votre fonction. La fonction traite un événement et renvoie une réponse. Pour plus d'informations, consultez [Utilisation des fonctions Lambda \(p. 14\)](#).
- Runtimes – Les runtimes Lambda permettent l'exécution de fonctions dans différents langages au sein d'un même environnement d'exécution de base. Vous configurez votre fonction de sorte à utiliser un runtime qui correspond à votre langage de programmation. Le runtime se situe entre le service Lambda et le code de votre fonction. Il relaie les événements d'appels, les informations de contexte et les réponses entre les deux. Vous pouvez utiliser les runtimes fournis par Lambda, ou créer vos propres runtimes. Pour plus d'informations, consultez [Runtimes AWS Lambda \(p. 103\)](#).
- Couches – Les couches Lambda sont un mécanisme de distribution pour les bibliothèques, les runtimes personnalisés, ainsi que d'autres dépendances de la fonction. Les couches vous permettent de gérer votre code de fonction en développement indépendamment du code immuable et des ressources qu'il utilise. Vous pouvez configurer votre fonction de sorte qu'elle utilise les couches que vous créez, les couches fournies par AWS ou des couches d'autres clients AWS. Pour plus d'informations, consultez [Couches AWS Lambda \(p. 56\)](#).
- Source d'événements – Un service AWS, tel que Amazon SNS, ou un service personnalisé, qui déclenche la fonction et exécute sa logique. Pour plus d'informations, consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#).
- Ressources en aval – Un service AWS, tel que les tables DynamoDB ou les compartiments Amazon S3, que votre fonction Lambda appelle une fois qu'elle est déclenchée.
- Flux de journaux – Lambda surveille automatiquement les appels de la fonction et consigne les métriques dans CloudWatch. Toutefois, vous pouvez annoter votre code de fonction à l'aide d'instructions de journalisation personnalisées qui vous permettront d'analyser le flux d'exécution et les performances de votre fonction Lambda afin de vous assurer qu'elle fonctionne correctement.
- AWS SAM – un modèle permettant de définir des [applications sans serveur](#). AWS SAM est pris en charge en mode natif par AWS CloudFormation et il définit la syntaxe simplifiée pour exprimer des ressources sans serveur. Pour plus d'informations, consultez [Qu'est-ce qu'AWS SAM ?](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Outils permettant de créer et de tester les fonctions Lambda

Trois principaux outils permettent d'interagir avec le service AWS Lambda ; ils sont décrits ci-dessous. Les outils permettant de développer des applications basées sur AWS Lambda seront abordés dans d'autres sections.

- Lambda Console – Permet de concevoir graphiquement une application basée sur Lambda, de créer ou de mettre à jour le code de fonction Lambda et de configurer l'événement, les ressources en aval et les autorisations IAM requis par la fonction.

- AWS CLI – Interface de ligne de commande que vous pouvez utiliser pour tirer parti des opérations d'API de Lambda, telles que la création de fonctions et le mappage de sources d'événements. Pour obtenir la liste complète des opérations de l'API Lambda, consultez [Actions \(p. 392\)](#).
- Interface de ligne de commande AWS – Interface de ligne de commande que vous pouvez utiliser pour développer, tester et analyser localement vos applications sans serveur avant de les charger dans le runtime Lambda. Pour plus d'informations, consultez [Interface de ligne de commande AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Configuration d'un compte AWS

Si ce n'est pas déjà fait, vous devez créer un compte AWS et un administrateur de compte. Vous devez également configurer l'AWS Command Line Interface (AWS CLI). Un grand nombre de didacticiels utilisent l'AWS CLI.

Pour effectuer la configuration, suivez les instructions des rubriques ci-après :

Configuration d'un compte AWS et création d'un utilisateur administrateur

Inscrivez-vous à AWS

Lors de votre inscription à Amazon Web Services (AWS), votre compte AWS est automatiquement inscrit à tous les services AWS, y compris AWS Lambda. Seuls les services que vous utilisez vous sont facturés.

Avec AWS Lambda, vous ne payez que les ressources que vous utilisez. Pour plus d'informations sur les tarifs d'utilisation relatifs à AWS Lambda, consultez la [page de présentation du produit AWS Lambda](#). Si vous êtes un nouveau client AWS, vous pouvez démarrer avec AWS Lambda gratuitement. Pour plus d'informations, consultez la page [Niveau d'offre gratuite d'AWS](#).

Si vous possédez déjà un compte AWS, passez à l'étape suivante. Si tel n'est pas le cas, observez la procédure suivante pour en créer un.

Pour créer un compte AWS

1. Ouvrez <https://aws.amazon.com/>, puis choisissez Créez un compte AWS.

Note

Si vous êtes déjà connecté à la AWS Management Console à l'aide d'informations d'identification Utilisateur racine d'un compte AWS, choisissez Sign in to a different account (Se connecter à un autre compte). Si vous êtes déjà connecté à la console à l'aide d'informations d'identification IAM, choisissez Sign in using root account credentials (Se connecter à l'aide d'informations d'identification de compte racine). Ensuite, choisissez Créez un nouveau compte AWS.

2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisierez un code de vérification en utilisant le clavier numérique du téléphone.

Notez votre ID de compte AWS, car vous en aurez besoin lors de la prochaine tâche.

Créer un utilisateur IAM

Pour accéder à un service d'AWS, tel qu'AWS Lambda, vous devez fournir vos informations d'identification afin que le service puisse déterminer si vous êtes autorisé à accéder à ses ressources. La console

exige votre mot de passe. Vous pouvez créer des clés d'accès pour votre compte AWS afin d'accéder à l'AWS CLI ou à l'API. Cependant, nous vous déconseillons d'accéder à AWS à l'aide des informations d'identification de votre compte AWS. À la place, nous vous recommandons d'utiliser AWS Identity and Access Management (IAM). Créez un utilisateur IAM, ajoutez-le à un groupe IAM disposant d'autorisations d'administration, puis attribuez ces autorisations à cet utilisateur IAM que vous avez créé. Vous pouvez alors accéder à AWS à l'aide d'une URL spéciale et des informations d'identification de cet utilisateur IAM.

Si vous êtes inscrit à AWS, mais que vous n'avez pas créé d'utilisateur IAM pour vous-même, vous pouvez le faire avec la console IAM.

Les exercices et les didacticiels de mise en route de ce guide présument que l'utilisateur (`adminuser`) dispose de privilèges d'administrateur. Lorsque vous suivez la procédure, créez un utilisateur avec le nom `adminuser`.

Pour créer un utilisateur IAM pour vous-même et l'ajouter au groupe Administrateurs.

1. Utilisez l'adresse e-mail et le mot de passe de votre compte AWS pour vous connecter en tant qu'[Utilisateur racine d'un compte AWS](#) à la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.

Note

Nous vous recommandons vivement de respecter la bonne pratique qui consiste à avoir recours à l'utilisateur IAM **Administrateur** ci-dessous et à mettre en sécurité les informations d'identification de l'utilisateur racine. Connectez-vous à l'utilisateur racine pour effectuer certaines [tâches de gestion des comptes et des services](#).

2. Dans le volet de navigation, choisissez Utilisateurs, puis Ajouter un utilisateur.
3. Pour Nom d'utilisateur, entrez **Administrateur**.
4. Activez la case à cocher en regard de AWS Management Console access, sélectionnez Custom password, puis tapez le nouveau mot de passe utilisateur dans la zone de texte. Vous pouvez également activer la case à cocher Require password reset (Exiger une réinitialisation du mot de passe) pour obliger l'utilisateur à créer un nouveau mot de passe lors de sa prochaine connexion.
5. Choisissez Next: Permissions.
6. Sur la page Set permissions , choisissez Add user to group.
7. Choisissez Create group.
8. Dans la boîte de dialogue Create group (Créer un groupe), tapez **Administrateurs** dans Group name (Nom du groupe).
9. Pour Filter policies (Stratégies de filtre), cochez la case correspondant à AWS managed - job function (Fonction professionnelle gérée par AWS).
10. Dans la liste des stratégies, activez la case à cocher AdministratorAccess. Choisissez ensuite Create group.
11. De retour dans la liste des groupes, activez la case à cocher du nouveau groupe. Choisissez Refresh si nécessaire pour afficher le groupe dans la liste.
12. Choisissez Next: Review pour afficher la liste des membres du groupe à ajouter au nouvel utilisateur. Une fois que vous êtes prêt à continuer, choisissez Create user.

Vous pouvez utiliser ce même processus pour créer d'autres groupes et utilisateurs et pour accorder à vos utilisateurs l'accès aux ressources de votre compte AWS. Pour en savoir plus sur l'utilisation des stratégies afin de limiter les autorisations d'accès des utilisateurs à certaines ressources AWS, consultez [Gestion des accès](#) et [Exemples de stratégies](#).

Pour vous connecter en tant que nouvel utilisateur IAM

1. Déconnectez-vous d'AWS Management Console.
2. Utilisez le format d'URL suivant pour vous connecter à la console :

`https://aws_account_number.signin.aws.amazon.com/console/`

La valeur `aws_account_number` correspond à votre ID de compte AWS, sans trait d'union. Par exemple, si votre ID de compte AWS est 1234-5678-9012, votre numéro de compte AWS est 123456789012. Pour découvrir comment déterminer votre numéro de compte, consultez [Votre ID de compte AWS et son alias](#) dans le IAM Guide de l'utilisateur.

3. Saisissez le nom utilisateur et le mot de passe IAM que vous venez de créer. Lorsque vous êtes connecté, la barre de navigation affiche `votre_nom_utilisateur @ votre_id_de_compte_aws`.

Si vous ne voulez pas que l'URL de votre page de connexion contienne votre ID de compte AWS, vous pouvez créer un alias de compte.

Pour créer ou supprimer un alias de compte

1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Dans le volet de navigation, sélectionnez Dashboard (Tableau de bord).
3. Localisez le lien de connexion des utilisateurs IAM.
4. Pour créer l'alias, cliquez sur Customize (Personnaliser), saisissez le nom que vous voulez utiliser pour l'alias, puis sélectionnez Yes, Create (Oui, créer).
5. Pour supprimer l'alias, cliquez sur Customize (Personnaliser), puis sur Yes, Delete (Oui, Supprimer). L'URL de connexion utilise de nouveau votre ID de compte AWS.

Pour vous connecter après avoir créé un alias de compte, utilisez l'URL suivante :

`https://your_account_alias.signin.aws.amazon.com/console/`

Pour contrôler le lien de connexion des utilisateurs IAM de votre compte, ouvrez la console IAM et vérifiez le lien sous IAM users sign-in link (Lien de connexion des utilisateurs IAM) sur le tableau de bord.

Pour plus d'informations sur IAM, consultez les ressources suivantes :

- [AWS Identity and Access Management \(IAM\)](#)
- [Mise en route](#)
- [IAM Guide de l'utilisateur](#)

Créer une fonction Lambda avec la console

Dans cet exercice de mise en route, vous commencez par créer une fonction Lambda à l'aide de la console AWS Lambda. Ensuite, vous appelez manuellement la fonction Lambda à l'aide d'un exemple de données d'événement. AWS Lambda exécute la fonction Lambda et renvoie les résultats. Vous vérifiez ensuite les résultats de l'exécution, y compris les journaux que la fonction Lambda a créés, ainsi que diverses métriques CloudWatch.

Lorsque vous suivez les différentes étapes, vous vous familiariserez également avec la console AWS Lambda, notamment en apprenant à effectuer les actions ci-dessous :

- Explorez les plans. Chaque plan fournit des exemples de code et de configuration qui vous permettent de créer des fonction Lambda en seulement quelques clics.
- Affichez et mettez à jour les informations de configuration de la fonction Lambda.

- Appelez une fonction Lambda manuellement et explorez les résultats dans la section Résultat de l'exécution.
- Contrôlez les métriques CloudWatch dans la console.

Suivez les étapes de cette section pour créer une fonction Lambda simple.

Pour créer une fonction Lambda

1. Connectez-vous à AWS Management Console et ouvrez la console AWS Lambda.
2. Notez qu'AWS Lambda propose une fonction `Hello World` simple au démarrage, sous l'intitulé Fonctionnement, et comprend une option Exécuter qui permet d'invoquer la fonction sous la forme d'une introduction générale. Ce didacticiel présente des options supplémentaires à votre disposition pour créer, tester et mettre à jour vos fonctions Lambda, ainsi que d'autres fonctions de la console Lambda, et il fournit des liens vers chacune de ces options pour que vous puissiez les explorer en détail.

Pour continuer, sélectionnez Create a function dans la section Get Started.

Note

La console affiche la page Mise en route seulement si vous n'avez encore créé aucune fonction Lambda. Si vous avez déjà créé des fonctions, vous voyez la page Lambda > Fonctions. Sur la page de la liste, sélectionnez Create a function pour accéder à la page Create function.

3. Sur la page Créer une fonction, trois options se présentent à vous :
 - Crée à partir de zéro
 - Plans
 - Serverless Application Repository

Pour plus d'informations sur l'utilisation de Serverless Application Repository, consultez [Qu'est-ce qu'AWS Serverless Application Repository ?](#)

- a. Pour revoir les plans, cliquez sur le bouton Blueprints afin d'afficher les plans disponibles. Vous pouvez également utiliser le filtre pour rechercher des plans spécifiques. Par exemple :
 - Entrez `s3` dans Filtrer pour obtenir uniquement la liste des plans disponibles pour traiter les événements Amazon S3.
 - Entrez `dynamodb` dans Filtrer pour obtenir la liste des plans disponibles pour traiter les événements Amazon DynamoDB.
- b. Dans le cadre de cet exercice de mise en route, choisissez le bouton **Author from scratch**.
4. Dans Crée à partir de zéro, procédez comme suit :
 - Dans Nom*, spécifiez le nom de votre fonction Lambda.
 - Dans Exécution*, choisissez `Python 3.7`.
 - Dans Rôle*, sélectionnez Crée un rôle à partir de modèles :
 - Dans le champ Role Name*, entrez un nom pour votre rôle.
 - Laissez vide le champ Modèles de stratégie. Dans le cadre de cette présentation, votre fonction Lambda aura les autorisations d'exécution nécessaires.

Note

Pour obtenir des informations détaillées sur les stratégies de sécurité d'AWS Lambda, consultez [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

- Sélectionnez Create Function.

5. Sous votre nouvelle page **function-name**, notez ce qui suit :

Dans le panneau Ajouter des déclencheurs, vous pouvez, si vous le souhaitez, choisir un service qui déclenche automatiquement votre fonction Lambda en sélectionnant l'une des options de service répertoriées.

Selon le service que vous sélectionnez, vous êtes invité à fournir des informations spécifiques. Par exemple, si vous sélectionnez DynamoDB, vous devez indiquer les éléments suivants :

- Nom de la table DynamoDB
- Taille de lot
- Position de départ

Pour cet exemple, ne configurez pas de déclencheur.

- Dans Function code (Code de fonction), notez que le code est fourni. Il renvoie un simple « Hello from Lambda ».
- Gestionnaire affiche la valeur **lambda_function.lambda_handler**. Il s'agit de **filename.handler-function**. La console enregistre l'exemple de code dans le fichier `lambda_function.py` et, dans le code, `lambda_handler` correspond au nom de la fonction qui reçoit l'événement comme paramètre lorsque la fonction Lambda est invoquée. Pour plus d'informations, consultez [Gestionnaire de fonctions AWS Lambda dans Python \(p. 305\)](#).
- Notez l'IDE intégré (Environnement de développement intégré). Pour en savoir plus, consultez la section [Création de fonctions à l'aide de l'éditeur de la console AWS Lambda \(p. 17\)](#).

6. Les autres options de configuration sur cette page sont les suivantes :

- Variables d'environnement – Pour les fonctions Lambda, elles vous permettent de transférer dynamiquement les paramètres vers le code de votre fonction et les bibliothèques, sans modifier votre code. Pour plus d'informations, consultez [Variables d'environnement AWS Lambda \(p. 30\)](#).
- Balises – Paires clé-valeur que vous attachez aux ressources AWS afin de mieux les organiser. Pour plus d'informations, consultez [Balisage des fonctions Lambda \(p. 275\)](#).
- Rôle d'exécution – Permet d'administrer la sécurité sur votre fonction, à l'aide de rôles et de stratégies définis ou à l'aide de nouveaux rôle et de nouvelles stratégies. Pour plus d'informations, consultez [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

- Paramètres de base – Permet d'imposer l'allocation de mémoire et le délai d'attente pour votre fonction Lambda. Pour plus d'informations, consultez [Limites AWS Lambda \(p. 288\)](#).
- Réseau – Permet de sélectionner un VPC auquel votre fonction accèdera. Pour plus d'informations, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#).
- Débogage et gestion des erreurs – Permet de sélectionner une ressource [Files d'attente Lettre Morte \(p. 81\)](#) pour analyser les nouvelles tentatives d'appel de fonction ayant échoué. Cette option vous permet également d'activer le suivi actif. Pour plus d'informations, consultez [Utilisation d'AWS X-Ray \(p. 231\)](#).
- Simultanéité – Permet d'allouer une limite spécifique d'exécutions simultanées à cette fonction. Pour plus d'informations, consultez [Limite des exécutions simultanées au niveau de la fonction \(p. 280\)](#).
- Audit et conformité – Permet de journaliser les appels de fonction à des fins d'audit opérationnel et des risques, d'administration et de conformité. Pour plus d'informations, consultez [Utilisation de AWS Lambda avec AWS CloudTrail \(p. 173\)](#).

Appeler la fonction Lambda et vérifier les résultats, les journaux et les métriques

Suivez les étapes indiquées pour appeler la fonction Lambda à l'aide de l'échantillon de données d'événement fourni dans la console.

1. Sur la page **yourfunction**, sélectionnez Test.
2. Sur la page Configure test event, sélectionnez Create new test event puis, sous Event template, conservez l'option par défaut Hello World. Saisissez un nom d'événement et notez l'exemple de modèle d'événement suivant :

```
{  
    "key3": "value3",  
    "key2": "value2",  
    "key1": "value1"  
}
```

Vous pouvez modifier les clés et les valeurs de l'exemple de code JSON. Toutefois, ne modifiez pas la structure de l'événement. Si vous modifiez les clés et les valeurs, vous devez mettre à jour l'exemple de code en conséquence.

Note

Si vous choisissez de supprimer l'événement de test, accédez à la page Configurer des événements de test, puis choisissez Supprimer.

3. Choisissez Créer, puis Test. Chaque utilisateur peut créer jusqu'à 10 événements de test par fonction. Ces événements de test ne sont pas disponibles pour les autres utilisateurs.
4. AWS Lambda exécute la fonction en votre nom. Le `handler` de la fonction Lambda reçoit, puis traite l'exemple d'événement.
5. Lorsque l'exécution aboutit, vérifiez les résultats dans la console.

The screenshot shows the AWS Lambda execution results page. At the top, a green checkmark indicates "Execution result: succeeded (logs)". Below this, a "Details" section shows the returned value: "Hello from Lambda". A "Summary" table provides execution statistics:

Code SHA-256	EK2s5r/qvcinS/tlz7w9OWqkT6XkRzQ6RtC2JFG3K8o=	Request ID	5e2630d3-d476-11e7-8121-ff2df5001d1f
Duration	9.04 ms	Billed duration	100 ms
Resources configured	128 MB	Max memory used	19 MB

The "Log output" section displays CloudWatch log entries:

```
START RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f Version: $LATEST
END RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f
REPORT RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f Duration: 9.04 ms Billed Duration: 100 ms Memory Size: 128 MB Max Mem
```

Remarques :

- La section Execution result indique l'état d'exécution succeeded, ainsi que les résultats de l'exécution de la fonction renvoyés par l'instruction `return`.

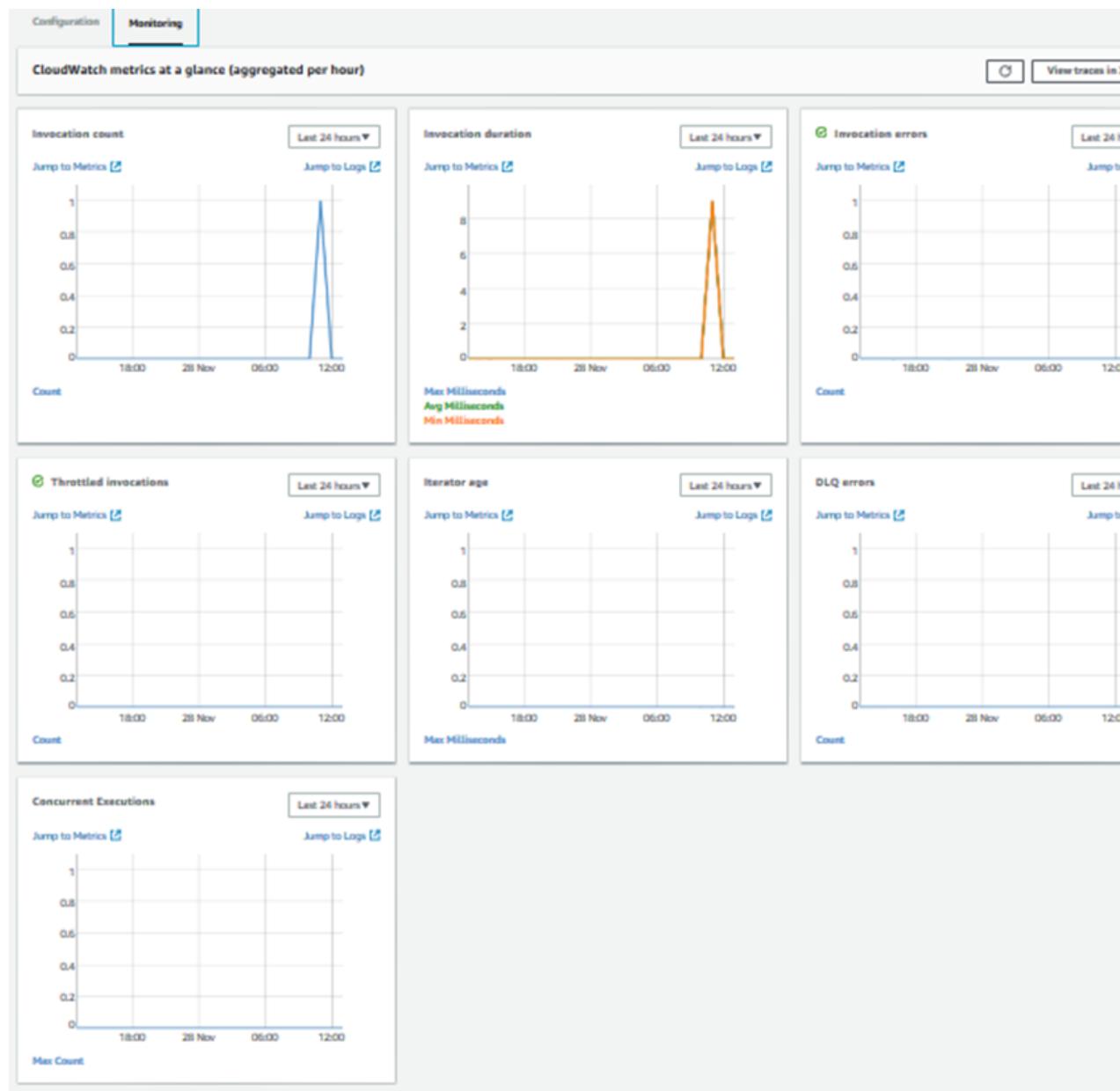
Note

La console utilise toujours le type d'appel `RequestResponse` (appel synchrone) lors de l'appel d'une fonction Lambda. AWS Lambda renvoie donc une réponse immédiatement. Pour plus d'informations, consultez [Types d'appel \(p. 74\)](#).

- La section Summary affiche les informations de clé présentées dans la section Log output (ligne REPORT dans le journal d'exécution).
- La section Sortie de journal affiche le journal généré par AWS Lambda pour chaque exécution. Il s'agit des journaux écrits dans CloudWatch par la fonction Lambda. La console AWS Lambda présente ces journaux à titre indicatif.

Notez que le lien Cliquez ici permet de consulter les journaux dans la console CloudWatch. La fonction ajoute ensuite les journaux dans Amazon CloudWatch, dans le groupe de journaux correspondant à la fonction Lambda.

- Exécutez la fonction Lambda plusieurs fois pour collecter des métriques que vous examinerez à l'étape suivante.
- Choisissez l'onglet Surveillance pour afficher les métriques CloudWatch de votre fonction Lambda. Cette page affiche les métriques CloudWatch.



Notez bien ce qui suit :

- L'axe des X affiche les 24 dernières heures à compter de l'heure actuelle.
- La valeur Invocation count présente le nombre d'appels au cours de cet intervalle.
- La valeur Invocation duration indique combien de temps l'exécution de la fonction Lambda a duré. Elle présente la durée d'exécution minimale, maximale et moyenne.
- La valeur Invocation errors affiche le nombre de fois que la fonction Lambda a échoué. Vous pouvez comparer le nombre de fois où la fonction a abouti et le nombre de fois où elle a échoué (le cas échéant).
- La métrique d'appels limités indique si AWS Lambda a limité l'appel de votre fonction Lambda. Pour plus d'informations, consultez [Limites AWS Lambda \(p. 288\)](#).
- Les métriques d'exécution simultanée présentent le nombre d'appels simultanés de la fonction Lambda. Pour plus d'informations, consultez [Gestion de la simultanéité \(p. 279\)](#).

- La console AWS Lambda présente ces métriques CloudWatch à titre indicatif. Pour les afficher dans la console Amazon CloudWatch, cliquez sur une de ces métriques.

Pour en savoir plus sur ces métriques et sur leur signification, consultez [Métriques AWS Lambda CloudWatch \(p. 229\)](#).

Configuration de l'AWS CLI

Tous les exercices de ce guide supposent que vous utilisez les informations d'identification d'administrateur de votre compte pour effectuer les opérations. Pour plus d'informations sur la création d'un administrateur dans votre compte AWS, consultez [Configuration d'un compte AWS et création d'un utilisateur administrateur \(p. 4\)](#), puis suivez les étapes pour télécharger et configurer l'AWS Command Line Interface (AWS CLI).

Pour configurer l'AWS CLI

1. Téléchargez et configurez l'AWS CLI. Pour obtenir des instructions, consultez les rubriques suivantes dans le AWS Command Line Interface Guide de l'utilisateur.
 - [Préparation de l'installation de l'AWS Command Line Interface](#)
 - [Configuration de la AWS Command Line Interface](#)
2. Ajoutez un profil nommé pour l'utilisateur administrateur dans le [fichier de configuration de l'AWS CLI](#). Vous utiliserez ce profil lorsque vous exécuterez les commandes AWS CLI. Pour plus d'informations sur la création de ce profil, consultez [Profiles nommés](#)

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

Pour obtenir la liste des régions AWS disponibles, consultez [Régions et points de terminaison](#) dans la Référence générale d'Amazon Web Services.

3. Vérifiez la configuration en saisissant les commandes suivantes à l'invite de commande.
 - Essayez la commande d'aide pour vérifier que l'AWS CLI est installé sur votre ordinateur :

```
aws help
```

- Testez une commande Lambda pour vérifier si l'utilisateur peut accéder à AWS Lambda. Cette commande répertorie les fonctions Lambda du compte, le cas échéant. L'AWS CLI utilise les informations d'identification `adminuser` pour authentifier la demande.

```
aws lambda list-functions --profile adminuser
```

Installation de l'interface de ligne de commande SAM

L'interface de ligne de commande AWS SAM est un outil de ligne de commande qui fonctionne sur un modèle AWS SAM et un code d'application. Avec l'interface de ligne de commande AWS SAM,

vous pouvez invoquer localement des fonctions Lambda, créer un package de déploiement pour votre application sans serveur, déployer votre application sans serveur vers le cloud AWS, etc.

Pour plus d'informations sur l'installation de l'interface de ligne de commande AWS SAM, consultez [Installation de l'interface de ligne de commande AWS SAM](#) dans le [Manuel du développeur Modèle d'application sans serveur AWS](#).

Utilisation des fonctions Lambda

Si vous n'êtes pas encore familiarisé avec AWS Lambda, vous vous demandez peut-être comment AWS Lambda exécute votre code ? Comment AWS Lambda détermine-t-il la quantité de mémoire et les exigences d'UC requises pour exécuter le code Lambda ? Les sections suivantes présentent le fonctionnement d'une fonction Lambda.

Dans les sections suivantes, nous expliquons de quelle façon sont appelées les fonctions que vous créez et comment les déployer et les surveiller. Nous vous recommandons également de lire les sections Code de fonction et Configuration de fonctions sous [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 283\)](#).

Pour commencer, la première rubrique explique les principes de base de la création d'une fonction Lambda, [Création de fonctions Lambda \(p. 14\)](#).

Rubriques

- [Création de fonctions Lambda \(p. 14\)](#)
- [Création de fonctions à l'aide de l'éditeur de la console AWS Lambda \(p. 17\)](#)
- [Modèle de programmation \(p. 24\)](#)
- [Création d'un package de déploiement \(p. 25\)](#)
- [Accès aux ressources AWS à partir d'une fonction Lambda \(p. 27\)](#)

Création de fonctions Lambda

Vous importez le code d'application sous la forme d'une ou de plusieurs fonctions Lambda dans AWS Lambda, un service de calcul. En retour, AWS Lambda exécute le code pour vous. AWS Lambda s'occupe de la mise en service et de la gestion des serveurs permettant d'exécuter le code lors d'un appel.

Généralement, le cycle de vie d'une application basée sur AWS Lambda inclut la création du code, son déploiement dans AWS Lambda, puis la surveillance et le dépannage. Voici les questions générales qui interviennent dans chaque phase de ce cycle de vie :

- Crédit de code pour la fonction Lambda – Quels sont les langages pris en charge ? Faut-il suivre un modèle de programmation spécifique ? Comment compresser le code et les dépendances afin de les importer dans AWS Lambda ? Quels sont les outils disponibles ?
- Importation de code et création de fonctions Lambda – Comment charger le package de code dans AWS Lambda ? Comment indiquer à AWS Lambda où commencer l'exécution de mon code ? Comment spécifier les exigences de calcul telles que la mémoire et le délai d'expiration ?
- Surveillance et dépannage – Pour la fonction Lambda qui est en production, quelles sont les métriques disponibles ? En cas d'erreur, comment obtenir les journaux ou résoudre les problèmes ?

Les sections suivantes fournissent des informations d'introduction, tandis que la section finale vous permet d'explorer divers exemples d'utilisation.

Création du code pour votre fonction Lambda

Vous pouvez créer le code de votre fonction Lambda dans les langages pris en charge par AWS Lambda. Pour consulter une liste des langages pris en charge, reportez-vous à la section [Environnement d'exécution](#)

Lambda et bibliothèques disponibles (p. 104). Des outils, tels que la console AWS Lambda, l'IDE Eclipse ou l'IDE Visual Studio, sont à votre disposition pour créer le code. Toutefois, les outils et les options proposés varient en fonction des éléments suivants :

- Le langage que vous choisissez pour écrire le code de la fonction Lambda.
- Les bibliothèques que vous utilisez dans le code. Le runtime AWS Lambda fournit certaines bibliothèques et vous devez charger les autres bibliothèques dont vous avez éventuellement besoin.

Le tableau suivant répertorie les langages, ainsi que les outils et les options que vous pouvez utiliser.

Langage	Outils et options pour la création de code
Node.js	<ul style="list-style-type: none">• Console AWS Lambda• Visual Studio, avec plug-in IDE (voir la section Prise en charge d'AWS Lambda dans Visual Studio)• Votre propre environnement de création• Pour plus d'informations, consultez Déploiement du code et création d'une fonction Lambda (p. 16).
Java	<ul style="list-style-type: none">• Eclipse, avec AWS Toolkit pour Eclipse (voir Utilisation d'AWS Lambda avec AWS Toolkit pour Eclipse)• Votre propre environnement de création• Pour plus d'informations, consultez Déploiement du code et création d'une fonction Lambda (p. 16).
C#	<ul style="list-style-type: none">• Visual Studio, avec plug-in IDE (voir la section Prise en charge d'AWS Lambda dans Visual Studio)• .NET Core (voir Guide d'installation de .NET Core)• Votre propre environnement de création• Pour plus d'informations, consultez Déploiement du code et création d'une fonction Lambda (p. 16).
Python	<ul style="list-style-type: none">• Console AWS Lambda• Votre propre environnement de création• Pour plus d'informations, consultez Déploiement du code et création d'une fonction Lambda (p. 16).
Go	<ul style="list-style-type: none">• Votre propre environnement de création• Pour plus d'informations, consultez Déploiement du code et création d'une fonction Lambda (p. 16).
PowerShell	<ul style="list-style-type: none">• Votre propre environnement de création• PowerShell Core 6.0 (voir Installation de PowerShell Core)• Kit SDK .NET Core 2.1 (voir les téléchargements .NET)• Module AWSLambdaPSCore (voir PowerShell Gallery)

Par ailleurs, quel que soit le langage que vous choisissez, un modèle commun régit la création du code de la fonction Lambda. Il définit, par exemple, comment écrire la méthode de gestionnaire de la fonction Lambda (autrement dit, la méthode qu'AWS Lambda appelle en premier quand il commence l'exécution du code), comment transmettre les événements au gestionnaire, quelles instructions utiliser dans le code pour générer des journaux dans CloudWatch Logs, comment interagir avec le runtime AWS Lambda et obtenir des informations telles que le temps restant avant l'expiration, ainsi que comment gérer les exceptions. La

section [Modèle de programmation \(p. 24\)](#) fournit des informations pour chacun des langages pris en charge.

Note

Une fois que vous vous êtes familiarisé avec AWS Lambda, consultez les [Cas d'utilisation \(p. 126\)](#), qui fournissent des instructions pas à pas pour tirer pleinement parti de votre environnement.

Déploiement du code et création d'une fonction Lambda

Pour créer une fonction Lambda, commencez par compresser le code et les dépendances dans un package de déploiement. Chargez ensuite ce package de déploiement dans AWS Lambda pour créer votre fonction Lambda.

Rubriques

- [Création d'un package de déploiement – Organisation du code et des dépendances \(p. 16\)](#)
- [Chargement d'un package de déploiement – Crédit d'une fonction Lambda \(p. 16\)](#)
- [Test d'une fonction Lambda \(p. 16\)](#)

Création d'un package de déploiement – Organisation du code et des dépendances

Vous devez tout d'abord organiser le code et les dépendances d'une certaine façon et créer un package de déploiement. Les instructions relatives à la création d'un package de déploiement varient en fonction du langage que vous choisissez pour concevoir le code. Par exemple, vous pouvez utiliser des plug-ins de développement tels que Jenkins (pour Node.js et Python) et Maven (pour Java) pour créer les packages de déploiement. Pour plus d'informations, consultez [Création d'un package de déploiement \(p. 25\)](#).

Lorsque vous créez des fonctions Lambda à l'aide de la console, celle-ci crée le package de déploiement pour vous, puis le charge afin de créer la fonction Lambda.

Chargement d'un package de déploiement – Crédit d'une fonction Lambda

AWS Lambda fournit l'opération [CreateFunction \(p. 410\)](#), qui vous permet de créer une fonction Lambda. Vous pouvez utiliser la console AWS Lambda, l'AWS CLI et les kits SDK AWS pour créer une fonction Lambda. En interne, toutes ces interfaces appellent l'opération [CreateFunction](#).

En plus de fournir le package de déploiement, vous pouvez spécifier des informations de configuration lorsque vous créez votre fonction Lambda, y compris les exigences de calcul de cette dernière, le nom de la méthode de gestionnaire dans la fonction Lambda et le runtime, qui dépend du langage que vous avez choisi pour concevoir le code. Pour plus d'informations, consultez [Utilisation des fonctions Lambda \(p. 14\)](#).

Test d'une fonction Lambda

Si votre fonction Lambda a pour but de traiter un type d'événement spécifique, vous pouvez utiliser un échantillon de données d'événement pour tester la fonction Lambda via l'une des méthodes suivantes :

- Testez la fonction Lambda dans la console.
- Testez votre fonction Lambda à l'aide de l'AWS CLI. Vous pouvez utiliser la méthode `Invoke` pour appeler la fonction Lambda et lui transmettre l'échantillon de données d'événement.

- Testez votre fonction Lambda localement à l'aide de l'[interface de ligne de commande AWS SAM](#).

La console offre également un échantillon de données d'événement. Ces données sont également fournies dans la rubrique [Exemples d'événements publiés en fonction des sources \(p. 91\)](#), que vous pouvez utiliser dans l'AWS CLI pour appeler la fonction Lambda.

Surveillance et dépannage

Une fois que la fonction Lambda est en production, AWS Lambda la surveille automatiquement en votre nom et présente les métriques associées via Amazon CloudWatch. Pour plus d'informations, consultez [Accès aux métriques Amazon CloudWatch pour AWS Lambda \(p. 225\)](#).

Pour vous aider à résoudre les problèmes d'une fonction, Lambda consigne toutes les demandes gérées par cette dernière et stocke automatiquement les journaux que votre code génère dans Amazon CloudWatch Logs. Pour plus d'informations, consultez [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 227\)](#).

Exemples d'applications basées sur AWS Lambda

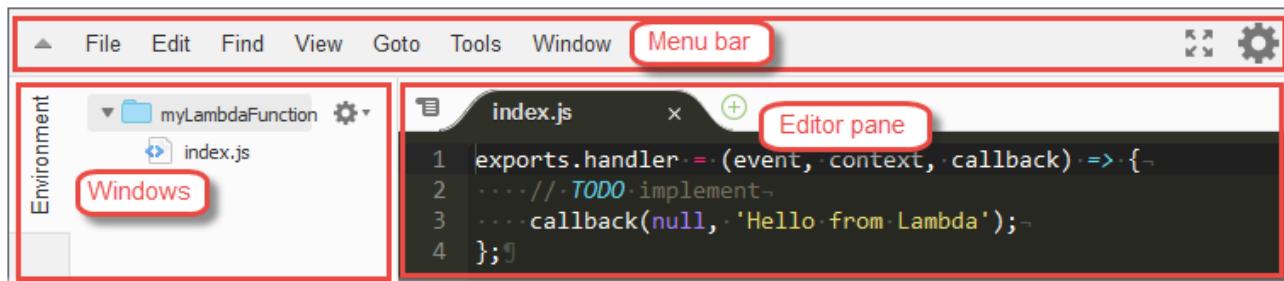
Ce guide fournit plusieurs exemples avec des instructions détaillées. Si vous utilisez AWS Lambda pour la première fois, nous vous recommandons d'essayer les exercices suivants :

- [Mise en route \(p. 3\)](#) – L'exercice de mise en route fournit un environnement basé sur la console. Un exemple de code est fourni pour vos exécutions préférées. Vous pouvez également écrire le code dans la console à l'aide de l'[éditeur de code](#), puis le charger dans AWS Lambda et le tester à l'aide de l'échantillon de données d'événement fourni dans la console.
- [Cas d'utilisation \(p. 126\)](#) – Si vous ne pouvez pas créer le code à l'aide de la console, vous devez créer vos propres packages de déploiement et utiliser l'AWS CLI (ou les kits SDK) pour créer votre fonction Lambda. Pour plus d'informations, consultez [Création du code pour votre fonction Lambda \(p. 14\)](#). La plupart des exemples de la section des cas d'utilisation s'appuient sur l'AWS CLI. Si vous utilisez AWS Lambda pour la première fois, nous vous recommandons d'essayer l'un de ces exercices.

Création de fonctions à l'aide de l'éditeur de la console AWS Lambda

L'éditeur de code dans la console AWS Lambda permet d'écrire, de tester et d'afficher les résultats de l'exécution du code de votre fonction Lambda.

L'éditeur de code se compose d'une barre de menus, de fenêtres et du volet de l'éditeur.



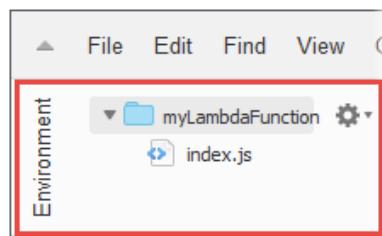
Vous utilisez la barre de menus pour exécuter les commandes courantes. Pour plus d'informations, consultez [Utilisation de la barre de menus \(p. 22\)](#).

Vous utilisez les fenêtres pour travailler avec des fichiers, des dossiers et d'autres commandes. Pour plus d'informations, consultez [Utilisation de fichiers et de dossiers \(p. 18\)](#) et [Utilisation des commandes \(p. 23\)](#).

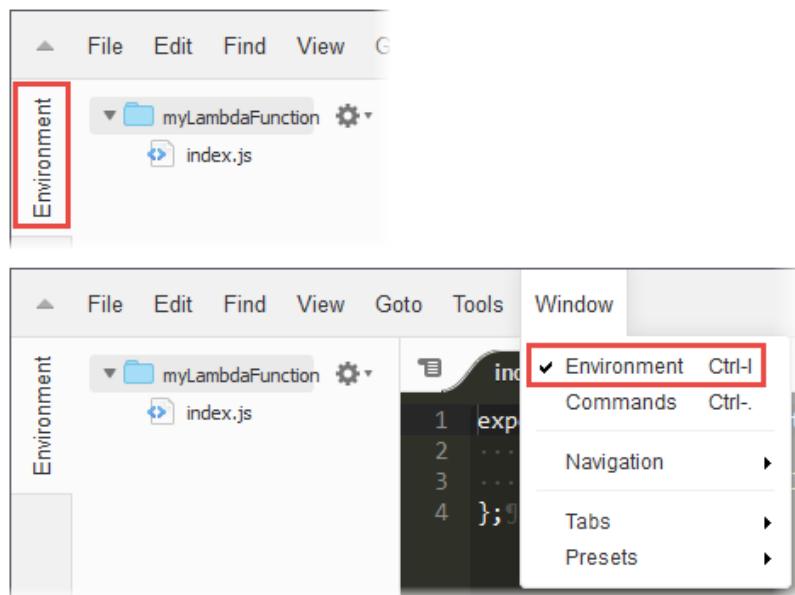
Vous utilisez le volet de l'éditeur pour écrire le code. Pour plus d'informations, consultez [Utilisation du code \(p. 19\)](#).

Utilisation de fichiers et de dossiers

Vous pouvez utiliser la fenêtre Environment dans l'éditeur de code pour créer, ouvrir et gérer des fichiers liés à votre fonction.



Pour afficher ou masquer la fenêtre Environment, sélectionnez le bouton Environment. Si le bouton Environment n'est pas visible, dans la barre de menus, sélectionnez Window, Environment.



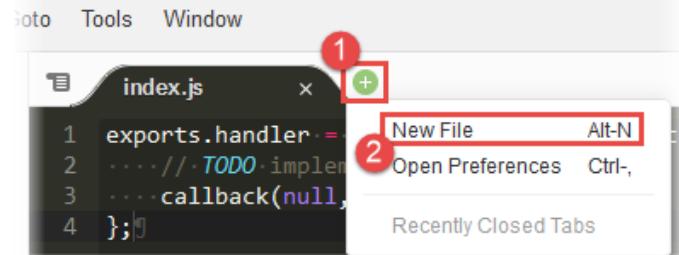
Pour ouvrir un seul fichier et afficher son contenu dans le volet de l'éditeur, double-cliquez sur le fichier dans la fenêtre Environment.

Pour ouvrir plusieurs fichiers et afficher leur contenu dans le volet de l'éditeur, sélectionnez les fichiers dans la fenêtre Environment. Cliquez avec le bouton droit de la souris sur la sélection, puis sélectionnez Open.

Pour créer un fichier, procédez comme suit :

- Dans la fenêtre Environment, cliquez avec le bouton droit de la souris sur le dossier où placer le nouveau fichier, puis sélectionnez New File. Tapez le nom et l'extension du fichier, puis appuyez sur Enter.
- Dans la barre de menus, sélectionnez File, New File. Quand vous êtes prêt à enregistrer le fichier, sélectionnez File, Save ou File, Save As dans la barre de menus. Ensuite, dans la boîte de dialogue Save As qui s'affiche, nommez le fichier et indiquez où l'enregistrer.

- Dans la barre de boutons de l'onglet du volet de l'éditeur, cliquez sur le bouton +, puis sélectionnez New File. Quand vous êtes prêt à enregistrer le fichier, sélectionnez File, Save ou File, Save As dans la barre de menus. Ensuite, dans la boîte de dialogue Save As qui s'affiche, nommez le fichier et indiquez où l'enregistrer.



Pour créer un dossier, dans la fenêtre Environment, cliquez avec le bouton droit de la souris sur le dossier où placer le nouveau dossier, puis sélectionnez New Folder. Tapez le nom du dossier, puis appuyez sur Enter.

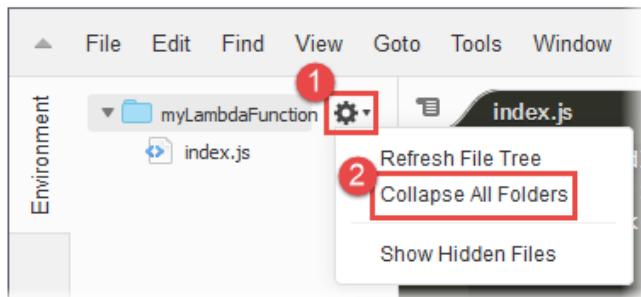
Pour enregistrer un fichier, avec le fichier ouvert et son contenu visible dans le volet de l'éditeur, sélectionnez File, Save dans la barre de menus.

Pour renommer un fichier ou un dossier, cliquez avec le bouton droit de la souris sur le fichier ou le dossier dans la fenêtre Environment. Tapez le nouveau nom, puis appuyez sur Enter.

Pour supprimer des fichiers ou des dossiers, sélectionnez les fichiers ou les dossiers dans la fenêtre Environment. Cliquez avec le bouton droit de la souris sur la sélection, puis sélectionnez Delete. Ensuite, confirmez la suppression en sélectionnant Yes (pour une seule sélection) ou Yes to All.

Pour couper, copier, coller ou dupliquer des fichiers ou des dossiers, sélectionnez les fichiers ou les dossiers dans la fenêtre Environment. Cliquez avec le bouton droit de la souris sur la sélection, puis sélectionnez Cut, Copy, Paste ou Duplicate, respectivement.

Pour réduire les dossiers, sélectionnez l'icône représentant un engrenage dans la fenêtre Environment, puis Collapse All Folders.

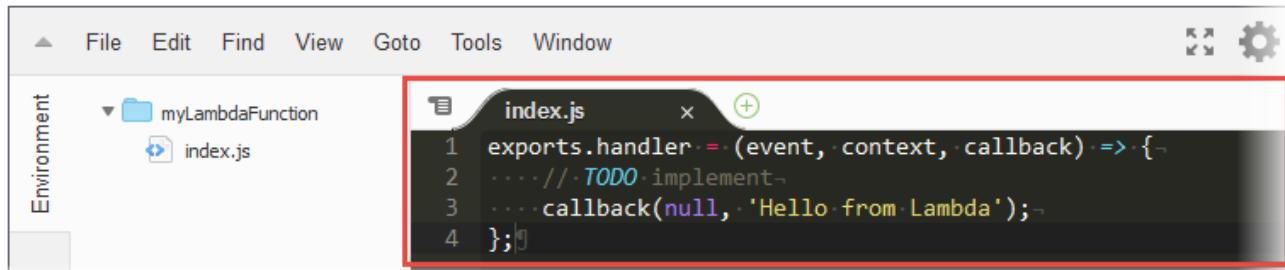


Pour afficher les fichiers masqués ou les masquer s'ils sont affichés, dans la fenêtre Environment, sélectionnez l'icône d'engrenage, puis Show Hidden Files.

Vous pouvez également créer, ouvrir et gérer les fichiers à l'aide de la fenêtre Commands. Pour plus d'informations, consultez [Utilisation des commandes \(p. 23\)](#).

Utilisation du code

Utilisez le volet de l'éditeur de code pour afficher et écrire le code.

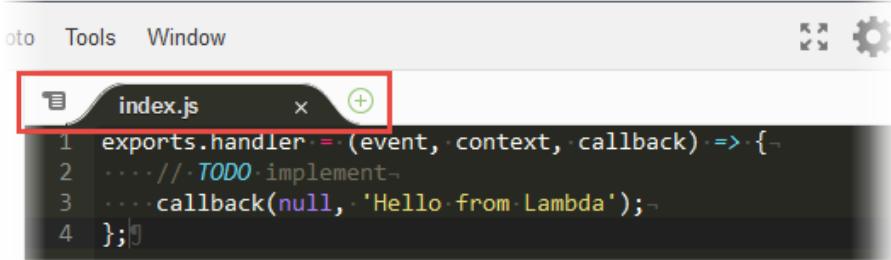


The screenshot shows the AWS Lambda developer interface. On the left, there's a sidebar labeled "Environment" with a folder icon "myLambdaFunction" and a file icon "index.js". The main area is a code editor with a dark theme. A red box highlights the tab for "index.js", which contains the following code:

```
index.js
1 exports.handler = (event, context, callback) => {
2     ... // TODO implement
3     callback(null, 'Hello from Lambda')
4 };
```

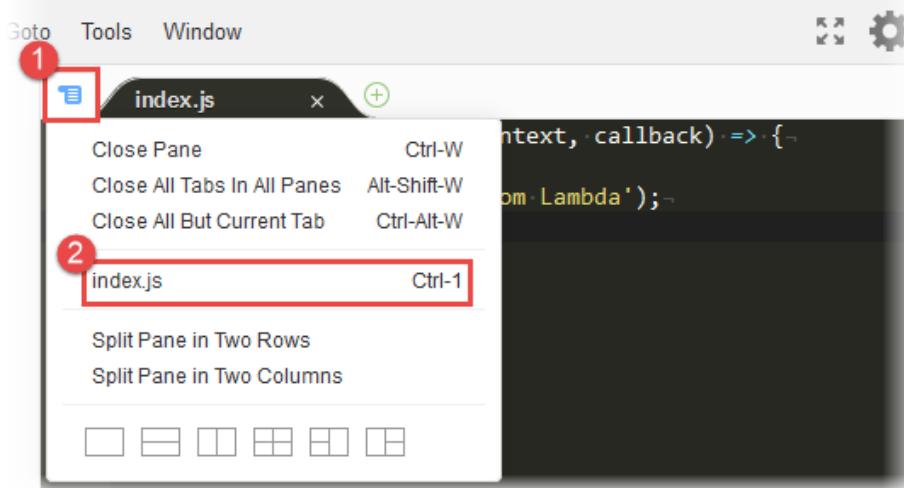
Utilisation des boutons de l'onglet

Utilisez la barre de boutons de l'onglet pour sélectionner, afficher et créer des fichiers.



Pour afficher le contenu d'un fichier ouvert, procédez de l'une des manières suivantes :

- Sélectionnez l'onglet du fichier.
- Sélectionnez le bouton du menu déroulant situé dans la barre de boutons de l'onglet, puis sélectionnez le nom du fichier.



Pour fermer un fichier ouvert, procédez de l'une des manières suivantes :

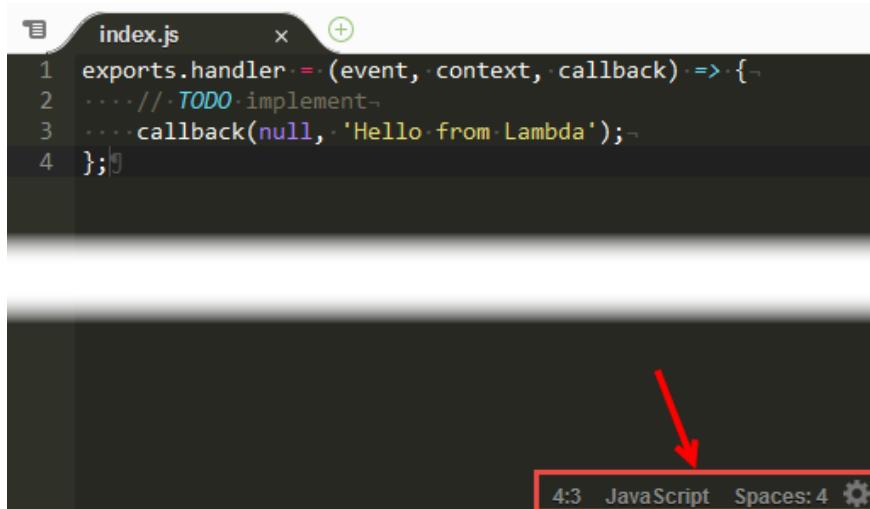
- Dans l'onglet du fichier, cliquez sur l'icône X.
- Sélectionnez l'onglet du fichier. Sélectionnez ensuite le bouton du menu déroulant situé dans la barre de boutons de l'onglet, puis cliquez sur Close Pane.

Pour fermer plusieurs fichiers ouverts, sélectionnez le menu déroulant dans la barre de boutons de l'onglet, puis cliquez sur Close All Tabs in All Panes ou sur Close All But Current Tab, selon vos besoins.

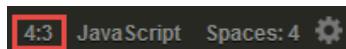
Pour créer un fichier, dans la barre de boutons de l'onglet, cliquez sur le bouton +, puis sélectionnez New File. Quand vous êtes prêt à enregistrer le fichier, sélectionnez File, Save ou File, Save As dans la barre de menus. Ensuite, dans la boîte de dialogue Save As qui s'affiche, nommez le fichier et indiquez où l'enregistrer.

Utilisation de la barre d'état

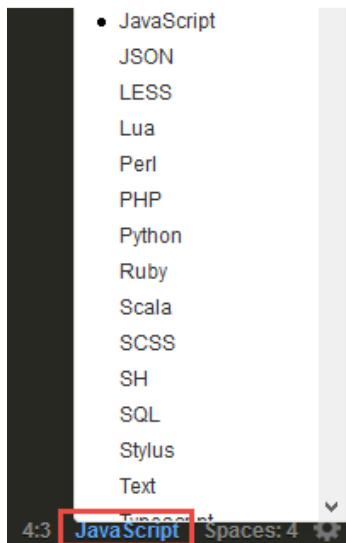
Utilisez la barre d'état pour accéder rapidement à une ligne du fichier actif et changer la façon dont s'affiche le code.



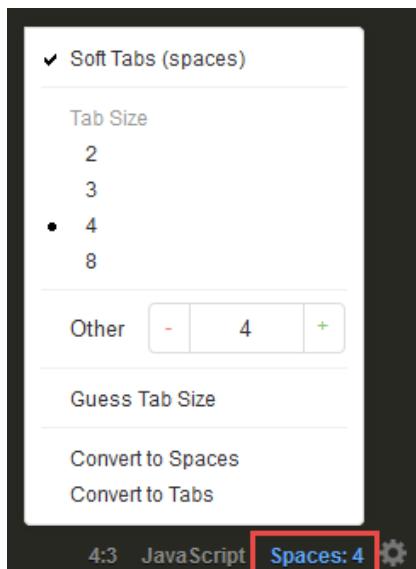
Pour passer rapidement à une ligne du fichier actif, choisissez le sélecteur de ligne, tapez le numéro de la ligne à atteindre, puis appuyez sur Enter.



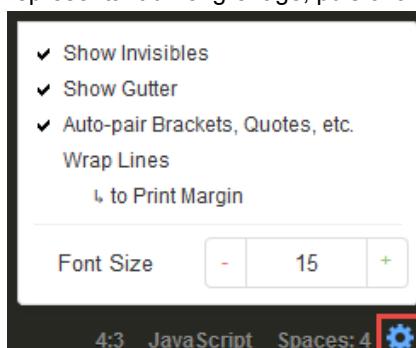
Pour modifier le modèle de couleurs du code dans le fichier actif, sélectionnez le sélecteur de modèle de couleurs du code, puis sélectionnez le nouveau modèle de couleurs du code.



Pour modifier, dans le fichier actif, si les onglets adoucis ou les espaces sont utilisés, la taille de l'onglet, ou s'il convient de les convertir en espaces ou en onglets, sélectionnez le sélecteur d'espaces et d'onglets, puis choisissez les nouveaux paramètres.



Pour afficher ou masquer les caractères invisibles ou la marge, coupler automatiquement les accolades ou les guillemets, retourner à la ligne ou modifier la taille de la police, pour tous les fichiers, cliquez sur l'icône représentant un engrenage, puis choisissez les nouveaux paramètres.



Utilisation de la barre de menus

Vous pouvez utiliser la barre de menus pour exécuter les commandes courantes.



Pour masquer la barre de menus, sélectionnez la flèche vers le haut dans la barre de menus.



Pour afficher la barre de menus si elle est masquée, sélectionnez la flèche vers le bas dans la barre de menus.



Pour obtenir la liste des rôles de chaque commande, consultez la [Référence des commandes de la barre de menus](#) dans le Guide de l'utilisateur AWS Cloud9. Remarque : Certaines des commandes répertoriées dans cette référence ne sont pas disponibles dans l'éditeur de code.

Vous pouvez également exécuter les commandes à l'aide de la fenêtre Commands. Pour plus d'informations, consultez [Utilisation des commandes \(p. 23\)](#).

Utilisation du mode plein écran

Afin d'avoir plus d'espace lorsque vous rédigez votre code, vous pouvez développer l'éditeur de code.

Pour développer l'éditeur de code jusqu'aux bords du navigateur web, cliquez sur le bouton Toggle fullscreen dans la barre de menus.



Pour restaurer l'éditeur de code à sa taille d'origine, cliquez de nouveau sur le bouton Toggle fullscreen.

En mode plein écran, des options supplémentaires apparaissent dans la barre de menus : Enregistrer et Tester. Sélectionnez Save pour enregistrer le code de la fonction. Sélectionnez Test ou Configure Events pour créer ou modifier les événements de test de la fonction.

Utilisation des préférences

Vous pouvez modifier certains paramètres de l'éditeur de code, en choisissant les conseils de codage et les avertissements qui s'affichent, les comportements de pliage de code ou de saisie automatique, etc.

Pour modifier les paramètres de l'éditeur de code, cliquez sur l'icône représentant un engrenage Préférences dans la barre de menus.



Pour obtenir la liste du rôle des paramètres, consultez les références suivantes dans le Guide de l'utilisateur AWS Cloud9.

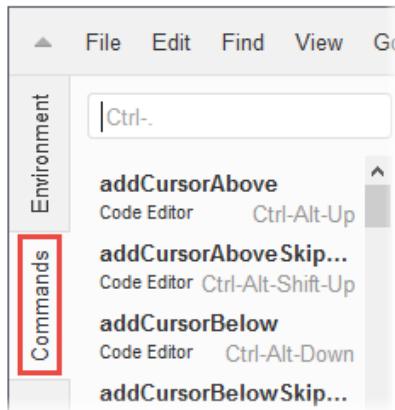
- [Modifications de paramètres de projet que vous pouvez effectuer](#)
- [Modifications de paramètres utilisateur que vous pouvez effectuer](#)

Remarque : Certains des paramètres répertoriés dans ces références ne sont pas disponibles dans l'éditeur de code.

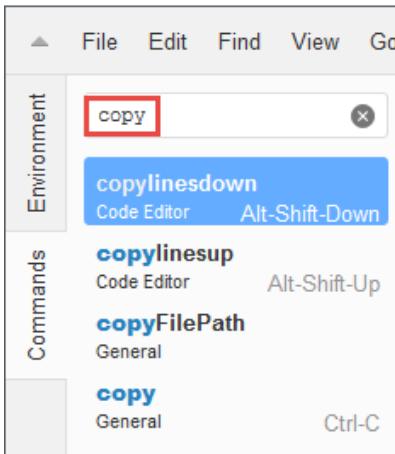
Utilisation des commandes

Vous pouvez utiliser la fenêtre Commands pour exécuter diverses commandes telles que celles disponibles dans la barre de menus, dans la fenêtre Environment, dans le volet de l'éditeur.

Pour afficher ou masquer la fenêtre Commands, choisissez le bouton Commands. Si le bouton Commands n'est pas visible, choisissez Window, Commands dans la barre de menus.



Pour exécuter une commande, sélectionnez-la dans la fenêtre Commands. Pour rechercher une commande, entrez son nom, en partie ou en totalité, dans la zone de recherche.



Pour obtenir la liste des rôles de chaque commande, consultez la [Référence des commandes](#) dans le Guide de l'utilisateur AWS Cloud9. Remarque : Certaines des commandes répertoriées dans cette référence ne sont pas disponibles dans l'éditeur de code.

Modèle de programmation

Vous écrivez le code de la fonction Lambda dans l'un des langages pris en charge par AWS Lambda. Quel que soit le langage que vous choisissez, un modèle commun incluant les concepts de base suivants régit la création du code pour une fonction Lambda :

- Gestionnaire : Le gestionnaire est la fonction appelée par AWS Lambda pour démarrer l'exécution de la fonction Lambda. Vous identifiez le gestionnaire lorsque vous créez la fonction Lambda. Lorsqu'une fonction Lambda est appelée, AWS Lambda démarre l'exécution de votre code en appelant la fonction de gestionnaire. AWS Lambda transmet toutes les données d'événement à ce gestionnaire comme premier paramètre. Le gestionnaire doit traiter les données d'événement entrantes et peut appeler d'autres fonctions/méthodes dans votre code.
- Objet de contexte et son mode d'interaction avec Lambda lors de l'exécution : AWS Lambda transmet également un objet de contexte à la fonction de gestionnaire, en tant que second paramètre. Cet objet de contexte permet à votre code de interagir avec AWS Lambda. Par exemple, le code peut trouver le temps d'exécution restant avant qu'AWS Lambda de mettre fin à la fonction Lambda.

En outre, pour les langages tels que Node.js, une plateforme asynchrone utilise les rappels. AWS Lambda fournit des méthodes supplémentaires sur cet objet de contexte. Ces méthodes d'objet contexte vous permettent de demander à AWS Lambda de mettre fin à la fonction Lambda et éventuellement de renvoyer les valeurs au mandataire.

- Journalisation : Votre fonction Lambda peut contenir des instructions de journalisation. AWS Lambda écrit ces journaux dans CloudWatch Logs. Ces instructions génèrent des entrées de journal, selon le langage que vous utilisez pour créer le code de la fonction Lambda.
- Exceptions : La fonction Lambda doit communiquer le résultat de l'exécution de la fonction à AWS Lambda. En fonction du langage utilisé pour créer le code de la fonction Lambda, il existe différentes façons de mettre fin à une requête avec succès ou d'informer AWS Lambda qu'une erreur s'est produite lors de l'exécution. Si vous appelez la fonction de manière synchrone, AWS Lambda renvoie le résultat au client.

Le code de la fonction Lambda doit être écrit dans un style sans état et n'avoir aucune affinité avec l'infrastructure de calcul sous-jacente. L'accès au système de fichiers local, les processus enfants et les éléments similaires sont limités à la durée de vie de la requête dans le code. L'état permanent doit être stocké dans Amazon S3, Amazon DynamoDB ou tout autre service de stockage cloud. L'obligation d'utiliser des fonctions sans état permet à AWS Lambda de lancer autant de copies d'une fonction que nécessaire pour s'adapter au nombre de requêtes et d'événements entrants. D'une requête à une autre, ces fonctions ne s'exécutent pas toujours sur la même instance de calcul et il n'est pas exclu qu'une instance particulière de la fonction Lambda soit utilisée plus d'une fois par AWS Lambda. Pour plus d'informations, consultez [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 283\)](#).

- [Création de fonctions Lambda avec Node.js \(p. 289\)](#)
- [Création de fonctions Lambda avec Python \(p. 301\)](#)
- [Création de fonctions Lambda avec Java \(p. 314\)](#)
- [Création de fonctions Lambda avec Go \(p. 344\)](#)
- [Création de fonctions Lambda avec C# \(p. 357\)](#)
- [Création de fonctions Lambda avec PowerShell \(p. 377\)](#)

Création d'un package de déploiement

Pour créer une fonction Lambda, vous devez d'abord concevoir un package de déploiement de la fonction Lambda, à savoir un fichier .zip ou .jar qui se compose de votre code et de toutes les dépendances. Lorsque vous créez le fichier zip, insérez uniquement le code et ses dépendances, pas le dossier dans lequel ils se trouvent. Vous devez alors définir les autorisations de sécurité appropriées pour le package zip.

- [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#)
- [Package de déploiement AWS Lambda dans Python \(p. 301\)](#)
- [Package de déploiement AWS Lambda dans Java \(p. 315\)](#)
- [Package de déploiement AWS Lambda dans Go \(p. 344\)](#)
- [Package de déploiement AWS Lambda dans C# \(p. 358\)](#)
- [Package de déploiement AWS Lambda dans PowerShell \(p. 377\)](#)

Stratégies d'autorisation sur les packages de déploiement Lambda

Les packages zip chargés avec les autorisations incorrectes peuvent entraîner un échec d'exécution. AWS Lambda nécessite des autorisations de lecture globales sur les fichiers de code et toute bibliothèque dépendante qui constitue votre package de déploiement. Pour vous assurer que les autorisations ne sont pas limitées à votre compte utilisateur, vous pouvez effectuer une vérification en utilisant les exemples suivants :

- Environnements Linux/Unix/OSX : Utilisez `zipinfo`, comme illustré dans l'exemple ci-dessous :

```
$ zipinfo test.zip
Archive: test.zip
Zip file size: 473 bytes, number of entries: 2
----- 3.0 unx      0 bx stor 17-Aug-10 09:37 exlib.py
----- 3.0 unx      234 tx defN 17-Aug-10 09:37 index.py
2 files, 234 bytes uncompressed, 163 bytes compressed:  30.3%
```

Le signe `-r-----` indique que seul le propriétaire du fichier possède des autorisations de lecture, ce qui peut provoquer des échecs d'exécution de la fonction Lambda. L'exemple suivant indique ce que vous devez voir s'il y a les autorisations de lecture globales requises :

```
$ zipinfo test.zip
Archive: test.zip
Zip file size: 473 bytes, number of entries: 2
-----r--r--r-- 3.0 unx      0 bx stor 17-Aug-10 09:37 exlib.py
-----r--r--r-- 3.0 unx      234 tx defN 17-Aug-10 09:37 index.py
2 files, 234 bytes uncompressed, 163 bytes compressed:  30.3%
```

Pour résoudre ce problème de façon récursive, exécutez la commande suivante :

```
$ chmod 644 $(find /tmp/package_contents -type f)
$ chmod 755 $(find /tmp/package_contents -type d)
```

- La première commande fait en sorte que tous les fichiers de `/tmp/package_contents` aient des autorisations en lecture/écriture accordées aux propriétaires, et des autorisations de lecture accordées au groupe et aux autres personnes.
- La deuxième commande répercute en cascade les mêmes autorisations pour les répertoires.

Note

Si vous vous servez d'un environnement Windows, nous vous recommandons d'utiliser [7-Zip](#) au lieu de `zipinfo`.

Une fois que vous avez terminé, définissez les autorisations IAM adéquates sur le package. Pour en savoir plus, consultez les stratégies [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

Accès aux ressources AWS à partir d'une fonction Lambda

Lambda n'applique aucune restriction à la logique de votre fonction : si vous pouvez en créer le code, vous pouvez l'exécuter au sein d'une fonction Lambda. Dans le cadre de votre fonction, il se peut que vous ayez besoin d'appeler d'autres API ou d'accéder à d'autres services AWS, tels que les bases de données.

Pour accéder aux services AWS

Pour accéder à d'autres services AWS, vous pouvez utiliser le kit AWS SDK ([Node.js](#), [Java](#), [Python](#), [C#](#) ou [Go](#)) ; AWS Lambda définit automatiquement les informations d'identification requises par le kit SDK sur celles du rôle IAM associé à votre fonction – aucune autre opération n'est nécessaire. Voici par exemple un exemple de code utilisant le kit de développement logiciel Python pour accéder à un objet S3 :

```
import boto3
import botocore

BUCKET_NAME = 'my-bucket' # replace with your bucket name
KEY = 'my_image_in_s3.jpg' # replace with your object key

s3 = boto3.resource('s3')

try:
    s3.Bucket(BUCKET_NAME).download_file(KEY, 'my_local_image.jpg')
except botocore.exceptions.ClientError as e:
    if e.response['Error']['Code'] == "404":
        print("The object does not exist.")
    else:
        raise
```

Note

Pour plus de commodité, AWS Lambda inclut des versions du kit de développement logiciel AWS dans le cadre de l'environnement d'exécution, de sorte que vous n'avez pas à l'inclure. Consultez [Environnement d'exécution Lambda et bibliothèques disponibles \(p. 104\)](#) pour connaître la version du kit de développement logiciel inclus. Il est recommandé d'inclure votre propre copie du kit de développement logiciel AWS pour les applications de production, afin que vous puissiez contrôler vos dépendances.

Accès aux services non AWS

Dans le cadre de votre fonction Lambda, vous pouvez inclure un kit de développement logiciel pour accéder à n'importe quel service. Vous pouvez par exemple inclure le [kit de développement logiciel pour Twilio](#) afin d'accéder aux informations de votre compte Twilio. Vous pouvez utiliser des [Variables d'environnement AWS Lambda \(p. 30\)](#) pour stocker les informations d'identification des kits de développement logiciel une fois celles-ci chiffrées.

Accès aux services privés ou aux ressources

Par défaut, pour qu'AWS Lambda puisse y accéder, votre service ou API doit être accessible via l'Internet public. Il se peut toutefois que certains services ou API ne soient pas exposés de cette façon. Généralement, vous créez ces ressources dans Amazon Virtual Private Cloud (Amazon VPC) afin qu'elles ne soient pas accessibles via le réseau Internet public. Il peut s'agir de ressources des services AWS, tels des entrepôts de données Amazon Redshift, des clusters Amazon ElastiCache ou des instances

Amazon RDS. Il peut également s'agir de vos propres services qui s'exécutent sur vos propres instances EC2. Par défaut, les ressources au sein d'un VPC ne sont pas accessibles à partir d'une fonction Lambda.

AWS Lambda exécute le code de la fonction de manière sécurisée au sein d'un VPC par défaut. Néanmoins, pour permettre à votre fonction Lambda d'accéder à des ressources au sein de votre VPC privé, vous devez fournir des informations de configuration supplémentaires spécifiques du VPC, en particulier les ID de sous-réseau VPC et les ID de groupe de sécurité. AWS Lambda utilise ces informations pour configurer les interfaces réseau Elastic (ENI) qui permettent à votre fonction de se connecter en toute sécurité à d'autres ressources au sein de votre VPC privé.

Important

AWS Lambda ne prend pas en charge la connexion aux ressources au sein des VPC à location dédiée. Pour plus d'informations, consultez [VPC dédiés](#).

Pour découvrir comment configurer une fonction Lambda afin d'accéder aux ressources au sein d'un VPC, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#).

Configuration des fonctions AWS Lambda

Vous pouvez utiliser la console ou l'API AWS Lambda pour configurer des paramètres sur vos fonctions Lambda. Les [paramètres de fonction de base \(p. 29\)](#) incluent la description, un rôle, le runtime et le rôle que vous spécifiez lorsque vous créez une fonction dans la console Lambda. Vous pouvez configurer plusieurs paramètres une fois que vous avez créé une fonction, ou utiliser l'API pour définir des éléments tels que le nom du gestionnaire, l'allocation de mémoire et les groupes de sécurité lors de la création.

Pour conserver des secrets hors de votre code de fonction, stockez-les dans la configuration de la fonction et lisez-les à partir de l'environnement d'exécution, au cours de l'initialisation. Les [variables d'environnement \(p. 30\)](#) sont toujours chiffrées au repos et peuvent également être chiffrées en transit. Utilisez des variables d'environnement pour rendre portable votre code de fonction en supprimant les chaînes de connexion, les mots de passe et les points de terminaison des ressources externes.

Les [versions et alias \(p. 38\)](#) sont des ressources secondaires que vous pouvez créer pour gérer le déploiement et l'appel des fonctions. Publiez les versions de votre fonction pour stocker son code et sa configuration en tant que ressource distincte qui ne peut pas être modifiée, et créez un alias qui pointe sur une version spécifique. Ensuite, vous pouvez configurer vos clients pour appeler un alias de fonction, et mettre à jour l'alias lorsque vous souhaitez pointer le client vers une nouvelle version, au lieu de mettre à jour le client.

Au fur et à mesure que vous ajoutez des bibliothèques et d'autres dépendances à votre fonction, la création et le chargement d'un package de déploiement peuvent ralentir le développement. Utilisez les [couches \(p. 56\)](#) afin de gérer les dépendances de votre fonction de manière indépendante et maintenir votre package de déploiement petit. Vous pouvez également utiliser des couches pour partager vos propres bibliothèques avec d'autres clients et utiliser des couches disponibles publiquement avec vos fonctions.

Pour utiliser votre fonction Lambda avec des ressources AWS dans un Amazon VPC, configurez-la avec des groupes de sécurité et des sous-réseaux pour [créer une connexion VPC \(p. 61\)](#). Lambda utilise des [interfaces réseau Elastic \(ENI\)](#) pour créer la connexion et, par conséquent, vous devez vous assurer que votre compte a une capacité ENI suffisante pour gérer le nombre de connexions effectuées lorsque votre fonction s'adapte à la hausse sous une charge.

Rubriques

- [Configuration de base d'une fonction AWS Lambda \(p. 29\)](#)
- [Variables d'environnement AWS Lambda \(p. 30\)](#)
- [Versions et alias des fonctions AWS Lambda \(p. 38\)](#)
- [Couches AWS Lambda \(p. 56\)](#)
- [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#)

Configuration de base d'une fonction AWS Lambda

Une fonction Lambda se compose du code et de toutes les dépendances associées. Des informations de configuration lui sont également associées. Vous spécifiez initialement les informations de configuration

lorsque vous créez une fonction Lambda. Lambda fournit une API pour vous permettre de mettre à jour certaines des données de configuration.

Configurer les paramètres de fonction

1. Ouvrez la console AWS Lambda à l'adresse <https://console.aws.amazon.com/lambda/>.
2. Choisissez une fonction.
3. Les paramètres sont affichés sur la première page, sous l'éditeur Code de fonction. Si vous choisissez d'autres ressources dans le panneau Designer, choisissez le nœud de la fonction pour retourner à ses paramètres.
4. Configurez l'une des options disponibles, puis choisissez Enregistrer.

Paramètres

- Mémoire : Spécifiez la quantité de mémoire que vous voulez allouer à votre fonction Lambda. AWS Lambda alloue la puissance d'UC proportionnelle à la mémoire en utilisant le même ratio que pour un type d'instance Amazon EC2 à usage général, comme un type M3 par exemple. Par exemple, si vous allouez 256 Mo de mémoire, votre fonction Lambda reçoit deux fois plus de puissance d'UC que si vous aviez alloué 128 Mo de mémoire seulement.

Vous pouvez mettre à jour la configuration et demander de la mémoire supplémentaire par incrément de 64 Mo, portée de 128 Mo à 3008 Mo. Pour en savoir plus sur les limites correspondantes, consultez la section [Limites AWS Lambda \(p. 288\)](#).

Vous pouvez également mettre à jour la taille de la mémoire de vos fonctions à l'aide de la commande AWS CLI suivante.

```
$ aws lambda update-function-configuration --function-name myfunction --memory-size 256
```

- Expiration : Vous payez pour les ressources AWS qui sont utilisées pour exécuter votre fonction Lambda. Afin d'empêcher l'exécution permanente de la fonction Lambda, vous devez spécifier un délai d'expiration. Une fois ce délai atteint, AWS Lambda met fin à l'exécution de votre fonction Lambda. Nous vous recommandons de définir cette valeur en fonction de la durée d'exécution prévue. La valeur par défaut pour les fonctions créées dans la console est de 3 secondes.

Note

Vous pouvez appeler une fonction Lambda de manière synchrone en appelant l'opération `Invoke` ou en utilisant un kit SDK AWS dans votre environnement d'exécution préféré. Si vous exécutez une fonction Lambda de longue durée, votre client peut expirer avant que l'exécution de la fonction ne soit terminée. Pour éviter ce problème, mettez à jour le délai d'expiration du client ou la configuration de votre kit SDK. Pour en savoir plus, consultez [Invoke \(p. 452\)](#).

- Rôle d'exécution : il s'agit du rôle qu'AWS Lambda assume lorsqu'il exécute la fonction Lambda en votre nom. Pour plus d'informations, consultez [Modèle d'autorisations AWS Lambda \(p. 260\)](#).
- Nom du gestionnaire : La méthode de gestionnaire est le point d'entrée qui exécute le code de votre fonction Lambda et toutes les dépendances de source d'événement que vous avez incluses dans le cadre de votre fonction Lambda. Ce paramètre s'affiche dans l'éditeur de code de la fonction.

Variables d'environnement AWS Lambda

Les variables d'environnement pour les fonctions Lambda vous permettent de transférer dynamiquement les paramètres vers le code de votre fonction et les bibliothèques, sans modifier votre code. Les variables d'environnement sont des paires clé-valeur que vous créez et modifiez dans le cadre de la configuration de

votre fonction, à l'aide de la console AWS Lambda, de l'interface de ligne de commande AWS Lambda ou du kit de développement logiciel AWS Lambda. AWS Lambda rend ces paires clé-valeur disponibles pour le code de votre fonction Lambda à l'aide des API standard prises en charge par le langage, par exemple `process.env` pour les fonctions Node.js.

Vous pouvez utiliser des variables d'environnement pour indiquer aux bibliothèques dans quel répertoire les fichiers doivent être installés, les emplacements de stockage des sorties, des connexions et des paramètres de connexion, etc. En séparant ces paramètres de la logique d'application, vous n'aurez pas besoin de mettre à jour le code de votre fonction pour modifier le comportement de cette dernière sur la base de différents paramètres.

Configuration

Supposons que vous souhaitez qu'une fonction Lambda se comporte différemment au fil des étapes du cycle de vie, depuis le développement au déploiement. Par exemple, les étapes de développement, de test et de production peuvent contenir des bases de données nécessaires à la fonction pour se connecter, lesquelles requièrent diverses informations de connexion et utilisent des noms de tables différents. Vous pouvez créer des variables d'environnement pour référencer les noms de base de données, les informations de connexion ou les noms de table et définir la valeur de la fonction selon l'étape dans laquelle elle s'exécute (par exemple, développement, test ou production) tout en maintenant le code de votre fonction inchangé.

Les captures d'écran suivantes indiquent comment modifier la configuration de votre fonction à l'aide de la console AWS. La première capture d'écran illustre la configuration des paramètres de la fonction correspondant à une étape de test. La seconde illustre la configuration des paramètres d'une étape de production.

The screenshot shows the 'Environment variables' section of the AWS Lambda configuration interface for a 'Test' stage. It displays two environment variables: 'Test_DB' with value 'TEST' and 'DB_Connection' with value 'Key'. Below this, the 'Encryption configuration' section is shown, featuring an unchecked checkbox for enabling helpers for encryption in transit, a dropdown for selecting a KMS key to encrypt at rest (set to '(default) aws/lambda'), and a button to 'Enter value'.

▼ Environment variables

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#).

Database	Test_DB	Remove
DB_Connection	TEST	Remove
Key	Value	Remove

▼ Encryption configuration

Enable helpers for encryption in transit [Info](#)

KMS key to encrypt at rest [Info](#)

Select a KMS key to encrypt the environment variables at rest, or simply let Lambda manage the encryption.

(default) aws/lambda

▼ Enter value

The screenshot shows the AWS Lambda Configuration page. In the 'Environment variables' section, there are two entries: 'Prod_DB' with value 'PROD' and 'DB_Connection'. The 'Prod_DB' entry is highlighted with a red box. Below it is a row with 'Key' and 'Value' fields, both empty, followed by a 'Remove' button. In the 'Encryption configuration' section, there is a checkbox labeled 'Enable helpers for encryption in transit' which is unchecked. Below it is a dropdown menu set to '(default) aws/lambda' with an 'Enter value' button next to it.

Notez la section **Encryption configuration**. Vous en apprendrez davantage sur son utilisation dans le didacticiel [Création d'une fonction Lambda à l'aide de variables d'environnement pour stocker les informations sensibles](#) (p. 37).

Vous pouvez également utiliser l'interface de ligne de commande AWS pour créer des fonctions Lambda qui contiennent des variables d'environnement. Pour en savoir plus, consultez [CreateFunction](#) (p. 410) et les API [UpdateFunctionConfiguration](#) (p. 514). Les variables d'environnement sont également prises en charge lors de la création et de la mise à jour des fonctions à l'aide d'AWS CloudFormation. Les variables d'environnement peuvent également être utilisées pour configurer les paramètres spécifiques à l'environnement d'exécution du langage ou à une bibliothèque incluse dans votre fonction. Par exemple, vous pouvez modifier `PATH` pour spécifier un répertoire où sont stockés les fichiers exécutables. Vous pouvez également définir des variables d'environnement spécifiques à l'environnement d'exécution, telles que `PYTHONPATH` pour Python ou `NODE_PATH` pour Node.js.

L'exemple suivant crée une nouvelle fonction Lambda qui définit la variable d'environnement `LD_LIBRARY_PATH` utilisée pour spécifier un répertoire dans lequel les bibliothèques partagées sont chargées dynamiquement lors de l'exécution. Dans cet exemple, le code de la fonction Lambda utilise la bibliothèque partagée dans le répertoire `/usr/bin/test/lib64`. Notez que le paramètre `Runtime` utilise `nodejs6.10`, mais que vous pouvez également spécifier `nodejs8.10`.

```
$ aws lambda create-function --function-name myTestFunction \
--zip-file fileb://package.zip \
--role role-arn \
--environment Variables="{LD_LIBRARY_PATH=/usr/bin/test/lib64}" \
--handler index.handler --runtime nodejs6.10
```

Règles relatives à l'attribution de noms pour les variables d'environnement

Aucune limite n'est imposée quant au nombre de variables d'environnement que vous pouvez créer, tant que la taille totale de l'ensemble ne dépasse pas 4 Ko.

Autres exigences :

- Le nom doit commencer par une lettre [a-zA-Z].
- Seuls les caractères alphanumériques et les traits de soulignement sont autorisés ([a-zA-Z0-9_]).

En outre, il existe un ensemble de clés réservées par AWS Lambda. Si vous tentez de définir des valeurs pour l'une de ces clés réservées, un message d'erreur s'affichera, vous indiquant que cette action n'est pas autorisée. Pour plus d'informations sur ces clés, consultez [Variables d'environnement disponibles pour les fonctions Lambda \(p. 104\)](#).

Variables d'environnement et gestion des versions des fonctions

La gestion des versions des fonctions permet de gérer le code de votre fonction Lambda en vous permettant de publier une ou plusieurs versions de votre fonction Lambda au fil de sa progression dans les différents environnements (développement, test et production). Pour chaque version d'une fonction Lambda que vous publiez, les variables d'environnement (ainsi que d'autres configurations spécifiques des fonctions, comme les limites `MemorySize` et `Timeout`) sont enregistrées sous la forme d'un instantané de la version et ces paramètres sont immuables (c'est-à-dire qu'ils ne peuvent être modifiés).

Compte tenu du fait que les exigences relatives aux applications et à la configuration évoluent, vous pouvez créer de nouvelles versions de votre fonction Lambda et mettre à jour les variables d'environnement afin de satisfaire à ces exigences avant que la version la plus récente ne soit publiée. La dernière version de votre fonction est `$LATEST`.

En outre, vous pouvez créer des alias, qui constituent des pointeurs vers une version particulière de votre fonction. Si vous avez besoin de restaurer une version précédente de votre fonction, vous pouvez pointer l'alias correspondant à cette version, qui contient les variables d'environnement requises pour cette version spécifique. Pour plus d'informations, consultez [Versions et alias des fonctions AWS Lambda \(p. 38\)](#).

Chiffrement des variables d'environnement

Lorsque vous créez ou mettez à jour des fonctions Lambda qui utilisent des variables d'environnement, AWS Lambda les chiffre à l'aide d'[AWS Key Management Service](#). Lorsque votre fonction Lambda est appelée, ces valeurs sont déchiffrées et mises à disposition pour le code Lambda.

La première fois que vous créez ou mettez à jour des fonctions Lambda utilisant des variables d'environnement dans une région, une clé de service par défaut est créée automatiquement au sein d'AWS KMS. Cette clé permet de chiffrer les variables d'environnement. Cependant, si vous souhaitez utiliser des assistants de chiffrement et employer KMS pour chiffrer les variables d'environnement après la création de la fonction Lambda, vous devez créer votre propre clé AWS KMS et la sélectionner à la place de la clé par défaut. La clé par défaut générera des erreurs si elle est choisie. La création de votre propre clé vous offre davantage de flexibilité, en vous permettant notamment de créer, de changer, de désactiver et de définir les contrôles d'accès, ainsi que de contrôler les clés de chiffrement utilisées pour protéger les données. Pour plus d'informations, consultez le [Manuel du développeur AWS Key Management Service](#).

Si vous utilisez votre propre clé, vous serez facturé conformément aux directives énoncées dans [Tarification d'AWS Key Management Service](#). Vous ne serez pas facturé si vous utilisez la clé de service par défaut fournie par AWS Lambda.

Si vous utilisez la clé de service KMS par défaut pour Lambda, aucune autre autorisation IAM n'est requise au sein du rôle d'exécution de votre fonction : votre rôle fonctionnera automatiquement sans aucune modification. Si vous devez fournir votre propre clé KMS (personnalisée), vous devrez ajouter `kms:Decrypt` à votre rôle d'exécution. En outre, l'utilisateur qui crée et met à jour la fonction Lambda doit disposer des autorisations nécessaires pour utiliser la clé KMS. Pour plus d'informations sur les clés KMS, consultez [Utilisation des stratégies de clé dans AWS KMS](#).

Note

AWS Lambda autorise votre fonction à utiliser la clé KMS par défaut par le biais d'un octroi d'utilisateur, qu'il ajoute lorsque le rôle est sélectionné pour la première fois. Si vous recréez un rôle d'exécution de la fonction (c'est-à-dire si vous supprimez et créez un rôle avec le même nom) et si le rôle ne dispose pas des autorisations `kms:Decrypt`, vous devez actualiser l'octroi du rôle. Vous pouvez le faire en faisant basculer le rôle d'exécution de la fonction après que le rôle a été recréé dans la console.

Stockage d'informations sensibles

Comme indiqué dans la section précédente, lorsque vous déployez votre fonction Lambda, toutes les variables d'environnement que vous avez spécifiées sont chiffrées par défaut, après, et non pendant, le processus de déploiement. Elles sont ensuite déchiffrées automatiquement par AWS Lambda lorsque la fonction est appelée. Si vous avez besoin de stocker des informations sensibles dans une variable d'environnement, nous vous recommandons vivement de chiffrer ces informations avant de déployer votre fonction Lambda.

Heureusement, la console Lambda facilite cette tâche en vous fournissant des assistants de chiffrement qui utilisent [AWS Key Management Service](#) pour stocker les informations sensibles en tant que `Ciphertext`. La console Lambda fournit également un code d'assistance pour le déchiffrement, qui permet de déchiffrer les informations à utiliser dans le code de votre fonction Lambda. Pour en savoir plus, consultez [Création d'une fonction Lambda à l'aide de variables d'environnement pour stocker les informations sensibles \(p. 37\)](#).

Scénarios d'erreur

Si la configuration de votre fonction excède 4 Ko, ou si vous utilisez des clés de variable d'environnement réservées par AWS Lambda, votre opération de mise à jour ou de création échouera, en affichant un message d'erreur de configuration. Il est possible que le chiffrement ou le déchiffrement des variables d'environnement échoue pendant l'exécution. Si AWS Lambda ne parvient pas à déchiffrer les variables d'environnement en raison d'une exception de service d'AWS KMS, AWS KMS renvoie un message sur l'exception, expliquant quelles sont les conditions d'erreur et, le cas échéant, les correctifs que vous pouvez appliquer pour remédier au problème. Ces erreurs seront consignées dans le flux de journaux de votre fonction, dans les journaux d'Amazon CloudWatch. Par exemple, si la clé KMS, que vous utilisez pour accéder aux variables d'environnement est désactivée, l'erreur suivante s'affichera :

```
Lambda was unable to configure access to your environment variables because the KMS key used is disabled.  
Please check your KMS key settings.
```

Étape suivante

[Création d'une fonction Lambda à l'aide des variables d'environnement \(p. 35\)](#)

Création d'une fonction Lambda à l'aide des variables d'environnement

Cette section indique comment vous pouvez modifier le comportement d'une fonction Lambda par le biais de modifications au niveau de la configuration, qui ne nécessitent aucune modification du code de la fonction Lambda.

Dans le cadre de ce didacticiel, vous effectuerez les tâches suivantes :

- Créer un package de déploiement avec un exemple de code qui renvoie la valeur d'une variable d'environnement spécifiant le nom d'un compartiment Amazon S3.
- Appeler une fonction Lambda et vérifier que le nom du compartiment Amazon S3 renvoyé correspond à la valeur définie par la variable d'environnement.
- Mettre à jour la fonction Lambda en modifiant le nom du compartiment Amazon S3 spécifié par la variable d'environnement.
- Appeler la fonction Lambda à nouveau et vérifier que le nom du compartiment Amazon S3 renvoyé correspond à la valeur à jour.

Étape 1 : Préparation

Assurez-vous que vous avez suivi les étapes ci-dessous :

- Vous avez créé un compte AWS, ainsi qu'un administrateur de compte (appelé adminuser). Pour obtenir des instructions, consultez [Configuration d'un compte AWS \(p. 4\)](#)
- Installez et configurez l'AWS CLI. Pour obtenir des instructions, consultez [Configuration de l'AWS CLI \(p. 12\)](#)

Étape 2 : Configurer l'environnement Lambda

Dans cette section, vous effectuez les opérations suivantes :

- Créez le package de déploiement d'une fonction Lambda avec l'exemple de code fourni.
- Créez un rôle d'exécution Lambda.
- Créez la fonction Lambda, en important le package de déploiement puis le tester en l'appelant manuellement.

Étape 2.1 : Créer le package de déploiement

L'exemple de code ci-dessous lit la variable d'environnement d'une fonction Lambda qui renvoie le nom d'un compartiment Amazon S3.

1. Ouvrez un éditeur de texte, puis copiez le code suivant :

```
var AWS = require('aws-sdk');

exports.handler = function(event, context, callback) {

    var bucketName = process.env.S3_BUCKET;
    callback(null, bucketName);
}
```

2. Enregistrez le fichier sous le nom index.js.

3. Compresez le fichier index.js dans un fichier zip appelé Test_Environment_Variables.zip.

Étape 2.2 : Créer un rôle d'exécution

Créez un rôle IAM (rôle d'exécution) que vous pouvez spécifier lors de la création de la fonction Lambda.

1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Pour créer un rôle IAM (rôle d'exécution), suivez les étapes [Rôles IAM](#) dans le IAM Guide de l'utilisateur. Lors de la création d'un rôle, prenez note des points suivants :
 - Dans Select Role Type, sélectionnez AWS Service Roles, puis AWS Lambda.
 - Dans Attach Policy, sélectionnez la stratégie nommée AWSLambdaBasicExecutionRole.
3. Prenez note de l'Amazon Resource Name (ARN) de ce rôle IAM. Vous en aurez besoin à l'étape suivante lorsque vous créerez la fonction Lambda.

Étape 2.3 : Créer la fonction Lambda et la tester

Dans cette section, vous créez une fonction Lambda contenant une variable d'environnement qui spécifie un compartiment Amazon S3 nommé `Test`. Lorsqu'elle est appelée, la fonction renvoie simplement le nom du compartiment Amazon S3. Vous mettrez ensuite à jour la configuration en renommant le compartiment Amazon S3 `Prod` et, lorsqu'elle sera appelée à nouveau, la fonction retournera le nouveau nom du compartiment Amazon S3.

Pour créer la fonction Lambda, ouvrez une invite de commande et exécutez la commande `Lambda create-function` de l'interface de ligne de commande AWS. Vous devez fournir le chemin d'accès du fichier .zip et l'ARN du rôle d'exécution.

```
$ aws lambda create-function --function-name ReturnBucketName \
--zip-file fileb://file-path/Test_Environment_Variables.zip \
--role role-arn \
--environment Variables={"S3_BUCKET=Test"} \
--handler index.handler --runtime nodejs8.10
```

Exécutez ensuite la commande `invoke` suivante de l'interface de ligne de commande Lambda pour appeler la fonction.

```
$ aws lambda invoke --function-name ReturnBucketName outputfile.txt
```

La fonction Lambda renverra le nom du compartiment Amazon S3, qui sera « `Test` ».

Ensuite, exécutez la commande `update-function-configuration` de l'interface de ligne de commande Lambda pour mettre à jour la variable d'environnement Amazon S3 en la pointant vers le compartiment `Prod`.

```
$ aws lambda update-function-configuration --function-name ReturnBucketName \
--environment Variables={"S3_BUCKET=Prod"}
```

Exécutez la commande `aws lambda invoke` à nouveau avec les mêmes paramètres. Cette fois-ci, la fonction Lambda renverra le nom du compartiment Amazon S3, qui sera `Prod`.

Création d'une fonction Lambda à l'aide de variables d'environnement pour stocker les informations sensibles

Vous pouvez non seulement spécifier les paramètres de configuration de votre fonction Lambda, mais également utiliser des variables d'environnement afin de stocker des informations sensibles, par exemple un mot de passe de base de données, à l'aide d'[AWS Key Management Service](#) et des assistants de chiffrement de la console Lambda. Pour plus d'informations, consultez [Chiffrement des variables d'environnement \(p. 33\)](#). L'exemple suivant indique comment s'y prendre et comment déchiffrer ces informations à l'aide de KMS.

Ce didacticiel montre comment utiliser la console Lambda pour chiffrer une variable d'environnement contenant des informations sensibles. Notez que si vous mettez à jour une fonction existante, vous pouvez passer directement à l'étape des instructions de la section indiquant comment développer des variables d'environnement dans [Etape 2 : Configurer la fonction Lambda \(p. 37\)](#).

Étape 1 : Créer la fonction Lambda

1. Connectez-vous à la AWS Management Console et ouvrez la console AWS Lambda à l'adresse <https://console.aws.amazon.com/lambda/>.
2. Choisissez Créer une fonction Lambda.
3. Dans Select blueprint, choisissez le bouton Author from scratch.
4. Dans Basic information, procédez comme suit :
 - Dans Nom, spécifiez le nom de votre fonction Lambda.
 - Dans Rôle, choisissez Choisir un rôle existant.
 - Dans Rôle existant, choisissez lambda_basic_execution.

Note

Si la stratégie du rôle d'exécution ne dispose pas de l'autorisation `decrypt`, vous devrez l'ajouter.

- Sélectionnez Create function.

Etape 2 : Configurer la fonction Lambda

1. Sous Configuration, spécifiez l'exécution (Runtime) de votre choix.
2. Dans la section Code de fonction Lambda, vous pouvez utiliser l'option Modifier le code en ligne pour remplacer le code du gestionnaire de la fonction Lambda par votre code personnalisé.
3. Vous remarquerez l'onglet Déclencheurs. La page Déclencheurs vous permet, le cas échéant, de sélectionner un service qui déclenche automatiquement la fonction Lambda si vous cliquez sur le bouton Ajouter des déclencheurs et choisissez la zone grise avec une ellipse (...) pour afficher la liste des services disponibles. Pour cet exemple, ne configurez pas de déclencheur et choisissez Configuration.
4. Vous remarquerez l'onglet Surveillance. Cette page fournit des métriques CloudWatch immédiates pour vos appels de fonction Lambda, ainsi que des liens vers d'autres guides utiles, notamment [Utilisation d'AWS X-Ray \(p. 231\)](#).
5. Développez la section Environment variables.
6. Entrez votre paire clé-valeur. Développez la section Encryption configuration. Notez que Lambda fournit sous Clé KMS pour le chiffrement au repos une clé de service par défaut qui chiffre vos informations une fois que celles-ci sont chargées. Si les informations que vous avez fournies sont

sensibles, vous pouvez également cocher la case `Enable helpers for encryption in transit` et fournir une clé personnalisée. La valeur que vous avez saisie sera alors masquée et entraînera un appel à AWS KMS pour chiffrer la valeur et la renvoyer sous la forme d'un `Ciphertext`. Si vous n'avez pas créé de clé KMS pour votre compte, un lien vers la console AWS IAM vous sera fourni, afin de vous permettre d'en créer une. Le compte doit disposer des autorisations `encrypt` et `decrypt` pour cette clé. Notez que le bouton `Encrypt` devient `Decrypt` une fois que vous l'avez choisi. Cela vous permet ainsi de mettre à jour les informations. Une fois que vous avez terminé, choisissez le bouton `Encrypt`.

Le bouton `Code` fournit un exemple de code de déchiffrement spécifique de l'environnement d'exécution de votre fonction Lambda, que vous pouvez utiliser avec votre application.

Note

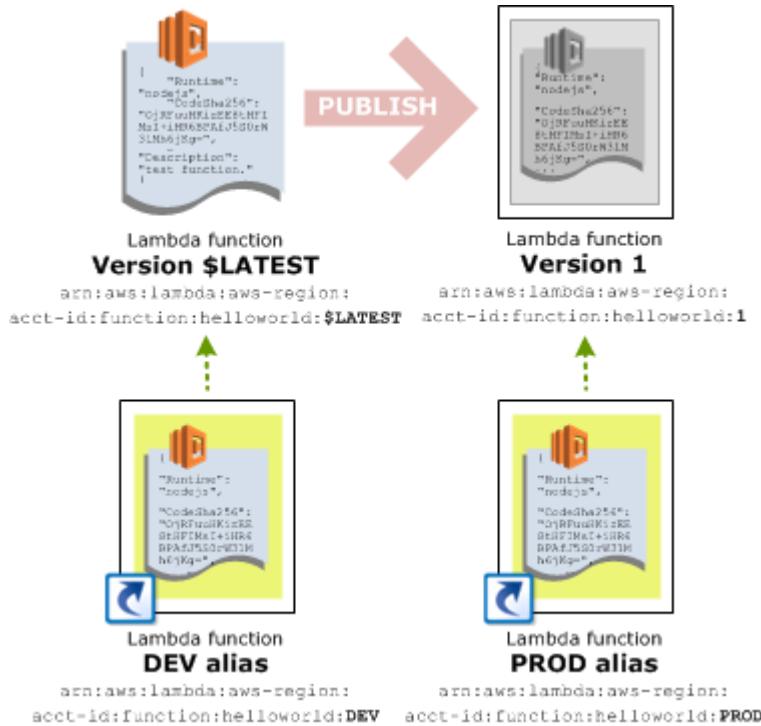
Vous ne pouvez pas utiliser la clé de service Lambda par défaut pour chiffrer les informations sensibles côté client. Pour en savoir plus, consultez [Chiffrement des variables d'environnement \(p. 33\)](#).

Versions et alias des fonctions AWS Lambda

Grâce à la gestion des versions, vous pouvez mieux gérer votre code de fonction en production dans AWS Lambda. Lorsque vous utilisez la gestion des versions dans AWS Lambda, vous pouvez publier une ou plusieurs versions de votre fonction Lambda. Par conséquent, vous pouvez utiliser différentes variantes de la fonction Lambda dans votre flux de travail de développement, telles qu'une version de développement, une version bêta et une version de production.

Chaque version d'une fonction Lambda possède un Amazon Resource Name (ARN) unique. Une fois que vous publiez une version, il est irréversible (autrement dit, vous ne pouvez pas le modifier).

AWS Lambda prend également en charge la création d'alias pour chaque version de la fonction Lambda. D'un point de vue conceptuel, un alias AWS Lambda s'apparente à un pointeur vers une version de fonction Lambda spécifique. Il s'agit également d'une ressource similaire à une fonction Lambda et chaque alias possède un ARN unique. Chaque alias utilise un ARN pour la version de fonction vers laquelle il pointe. Un alias peut uniquement pointer vers une version de fonction, et non vers un autre alias. Contrairement aux versions, qui sont irréversibles, les alias sont réversibles (cela signifie qu'ils peuvent être modifiés). Vous pouvez mettre à jour des alias afin qu'ils pointent vers différentes versions de fonctions.



Les alias vous permettent de faire la distinction entre les versions Lambda lors leur migration en production à partir du mappage de la version de fonction Lambda et de sa source d'événement.

Par exemple, supposons qu'Amazon S3 soit la source d'événement qui appelle la fonction Lambda lorsque des objets sont créés dans un compartiment. Dans ce cas, vous stockez les informations de mappage de source d'événement dans la configuration des notifications du compartiment. Dans cette configuration, vous pouvez identifier l'ARN de la fonction Lambda qu'Amazon S3 peut appeler. Cependant, dans ce cas, chaque fois que vous publiez une nouvelle version de votre fonction Lambda vous avez besoin de mettre à jour la configuration des notifications de manière à ce qu'Amazon S3 appelle la version correcte.

En revanche, imaginons que vous spécifiez l'ARN de l'alias dans la configuration des notifications (par exemple, l'ARN de l'alias PROD) au lieu de spécifier l'ARN de la fonction. Lorsque vous promouvez de nouvelles versions de votre fonction Lambda en production, il suffit de mettre à jour l'alias PROD pour pointer sur la version stable la plus récente. Vous n'avez pas besoin de mettre à jour la configuration des notifications dans Amazon S3.

Il en est de même lorsque vous souhaitez restaurer une version précédente de la fonction Lambda. Dans ce scénario, vous vous contentez de mettre à jour l'alias PROD pour qu'il pointe vers une autre version de la fonction. Vous n'avez pas besoin de mettre à jour les mappages de source d'événement.

Nous vous recommandons d'utiliser les versions et les alias pour déployer les fonctions Lambda lors de la création d'applications impliquant plusieurs dépendances et plusieurs développeurs.

Rubriques

- [Présentation des versions AWS Lambda \(p. 40\)](#)
- [Présentation des alias AWS Lambda \(p. 43\)](#)
- [Versions, alias et stratégies de ressources \(p. 51\)](#)
- [Gestion des versions via AWS Management Console, l'AWS CLI ou les opérations d'API Lambda \(p. 52\)](#)
- [Déplacement du trafic à l'aide des alias \(p. 54\)](#)

Présentation des versions AWS Lambda

Vous trouverez ci-dessous des informations sur la création d'une fonction Lambda et sur la publication d'une version à partir de cette fonction. Vous trouverez également des informations sur la mise à jour des informations de configuration et du code de la fonction lorsque vous avez une ou plusieurs versions publiées. Par ailleurs, cette section inclut des informations sur la suppression des versions de fonction, qu'il s'agisse d'une version spécifique ou d'une fonction Lambda complète, avec tous ses alias et versions associées.

Création d'une fonction Lambda (version \$LATEST)

Lorsque vous créez une fonction Lambda, il n'y a qu'une seule version, la version \$LATEST.



Lambda function
Version \$LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld:\$LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld

Vous pouvez vous reporter à cette fonction à l'aide de son Amazon Resource Name (ARN). Deux ARN sont associés à cette version initiale :

- ARN qualifié : ARN de la fonction avec le suffixe de version.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:$LATEST
```

- ARN non qualifié : ARN de la fonction ARN sans le suffixe de version.

Vous pouvez utiliser cet ARN non qualifié dans toutes les opérations appropriées. Cependant, vous ne pouvez pas l'utiliser pour créer un alias. Pour en savoir plus, consultez [Présentation des alias AWS Lambda \(p. 43\)](#).

L'ARN non qualifié a ses propres stratégies de ressource.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld
```

Note

À moins que vous ne choisissez de publier des versions, la version de fonction \$LATEST est la seule version de la fonction Lambda dont vous disposez. Vous pouvez utiliser l'ARN qualifié ou non qualifié dans le mappage de source d'événement pour appeler la version \$LATEST.

Voici un exemple de réponse d'un appel d'API `CreateFunction`.

```
{  
    "CodeSize": 287,  
    "Description": "test function."  
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",  
    "FunctionName": "helloworld",
```

```
"Handler": "helloworld.handler",
"LastModified": "2015-07-16T00:34:31.322+0000",
"MemorySize": 128,
"Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",
"Runtime": "nodejs6.10",
"Timeout": 3,
"CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",
"Version": "$LATEST"
}
```

Pour plus d'informations, consultez [CreateFunction \(p. 410\)](#).

Dans cette réponse, AWS Lambda renvoie l'ARN non qualifié de la fonction qui vient d'être créée, ainsi que sa version, \$LATEST. La réponse indique également que la valeur Version correspond à \$LATEST. CodeSha256 est le total de contrôle du package de déploiement que vous avez importé.

Publication d'une version de fonction AWS Lambda

Lorsque vous publiez une version, AWS Lambda copie un instantané du code de la fonction Lambda (et de sa configuration) dans la version \$LATEST. Une version publiée est immuable. Autrement dit, vous ne pouvez pas modifier le code ni les informations de configuration. La nouvelle version possède un ARN unique qui inclut un suffixe de numéro de version, comme illustré ci-après.



Vous pouvez publier une version à l'aide de l'une des méthodes suivantes :

- Publier une version explicitement : vous pouvez utiliser l'opération d'API `PublishVersion` pour publier explicitement une version. Pour en savoir plus, consultez [PublishVersion \(p. 482\)](#). Cette opération génère une nouvelle version avec le code et la configuration de la version \$LATEST.
- Publier une version au moment de la création ou de la mise à jour d'une fonction Lambda : vous pouvez utiliser les requêtes `CreateFunction` ou `UpdateFunctionCode` pour publier une version en ajoutant le paramètre `publish` facultatif dans la requête :
 - Spécifiez le paramètre `publish` dans votre requête `CreateFunction` pour créer une nouvelle fonction Lambda (la version \$LATEST). Vous pouvez ensuite immédiatement publier la nouvelle fonction en créant un instantané et en lui attribuant le numéro de version 1. Pour en savoir plus sur `CreateFunction`, consultez [CreateFunction \(p. 410\)](#).
 - Spécifiez le paramètre `publish` dans votre demande `UpdateFunctionCode` pour mettre à jour le code dans la version \$LATEST. Vous pouvez ensuite publier une version à partir de \$LATEST. Pour en savoir plus sur `UpdateFunctionCode`, consultez [UpdateFunctionCode \(p. 508\)](#).

Si vous spécifiez le paramètre `publish` lorsque vous créez une fonction Lambda, les informations de configuration renvoyées par AWS Lambda pour cette fonction indiquent le numéro de la nouvelle version publiée. Dans l'exemple suivant, la version est 1.

```
{
  "CodeSize": 287,
```

```

    "Description": "test function."
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",
    "FunctionName": "helloworld",
    "Handler": "helloworld.handler",
    "LastModified": "2015-07-16T00:34:31.322+0000",
    "MemorySize": 128,
    "Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",
    "Runtime": "nodejs6.10",
    "Timeout": 3,
    "CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",
    "Version": "1"
}

```

Note

Lambda publie uniquement une nouvelle version si le code n'a pas encore été publié ou s'il a changé par rapport à la version \$LATEST. S'il n'y a aucun changement, la version \$LATEST publiée est renvoyée.

Nous vous recommandons de publier une version en même temps que vous créez votre fonction Lambda ou que vous mettez son code à jour. Cette recommandation s'applique en particulier lorsque plusieurs développeurs contribuent au développement d'une même fonction Lambda. Pour ce faire, vous pouvez utiliser le paramètre publish dans votre requête.

Lorsque plusieurs développeurs participent à un projet, vous pouvez rencontrer ce scénario : le développeur A crée une fonction Lambda (version \$LATEST). Avant que le développeur A publie cette version, le développeur B peut mettre à jour le code (le package de déploiement) associé à la version \$LATEST. Dans ce cas, vous perdez le code d'origine importé par le développeur A. Lorsque les deux développeurs ajoutent le paramètre publish, cela empêche cette condition de concurrence.

Chaque version d'une fonction Lambda est une ressource unique avec un Amazon Resource Name (ARN). L'exemple suivant affiche l'ARN du numéro de version 1 de la fonction Lambda helloworld.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

Vous pouvez publier plusieurs versions d'une fonction Lambda. Chaque fois que vous publiez une version, AWS Lambda copie la version \$LATEST (code et informations de configuration) afin de créer une nouvelle version. Lorsque vous publiez des versions supplémentaires, AWS Lambda attribue un numéro de séquence croissant de façon monotone aux versions, même si la fonction a été supprimée et recréée. Les numéros de version ne sont jamais réutilisés, même pour une fonction qui a été supprimée et recréée. Cette approche signifie que l'utilisateur d'une version de fonction sait que l'exécutable de cette version ne changera jamais (sauf s'il est supprimé).

Si vous souhaitez réutiliser une qualification, utilisez des alias avec vos versions. Les alias peuvent être supprimés et recréés avec le même nom.



Mise à jour du code et de la configuration de la fonction Lambda

AWS Lambda gère le dernier code de la fonction dans la version `$LATEST`. Lorsque vous mettez à jour le code de la fonction, AWS Lambda remplace le code dans la version `$LATEST` de la fonction Lambda. Pour en savoir plus, consultez [UpdateFunctionCode \(p. 508\)](#).

Les options suivantes s'offrent à vous pour publier une nouvelle version lorsque vous mettez à jour le code de la fonction Lambda :

- Publier une version dans la même demande de mise à jour du code : utilisez l'opération d'API `UpdateFunctionCode` (approche recommandée).
- Commencer par mettre à jour le code, puis publier explicitement une version : utilisez l'opération d'API `PublishVersion`.

Vous pouvez mettre à jour le code et les informations de configuration (par exemple, la description, la taille de la mémoire et le délai d'exécution) de la version `$LATEST` de la fonction Lambda.

Suppression d'une fonction Lambda et d'une version spécifique

Les versions offrent les possibilités suivantes :

- Supprimer une version spécifique : pour supprimer une version de fonction Lambda, spécifiez celle que vous souhaitez supprimer dans la requête `DeleteFunction`. Si des alias dépendent de cette version, la requête échoue. AWS Lambda supprime la version uniquement si aucun alias n'en dépend. Pour en savoir plus sur les alias, consultez la section [Présentation des alias AWS Lambda \(p. 43\)](#).
- Supprimer la fonction Lambda complète (versions et alias) : pour supprimer la fonction Lambda ainsi que toutes ses versions, ne spécifiez aucune version dans la requête `DeleteFunction`. Cela supprimera l'ensemble de la fonction, y compris ses versions et les alias.

Présentation des alias AWS Lambda

Vous pouvez créer un ou plusieurs alias pour une fonction Lambda. Un alias AWS Lambda s'apparente à un pointeur vers une version de fonction Lambda spécifique. Pour en savoir plus sur la gestion des versions, consultez [Présentation des versions AWS Lambda \(p. 40\)](#).

Les alias vous permettent d'accéder à la fonction Lambda vers laquelle ils pointent (par exemple, pour appeler la fonction) sans que le mandataire n'ait besoin de connaître la version spécifique à laquelle ils sont associés.

Les alias AWS Lambda répondent aux besoins suivants :

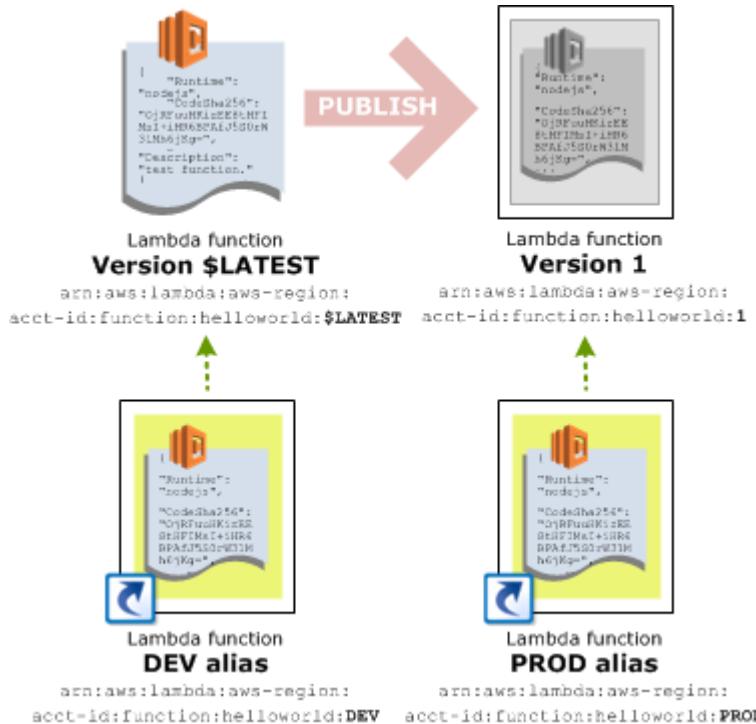
- Rationaliser la migration des nouvelles versions de fonctions Lambda et, le cas échéant, de la restauration d'une version : après la création initiale d'une fonction Lambda (version `$LATEST`), vous pouvez publier la version 1. En créant un alias nommé PROD qui pointe vers la version 1, vous pouvez désormais utiliser l'alias PROD pour appeler la version 1 de la fonction Lambda.

Désormais, vous pouvez mettre à jour le code (version `$LATEST`) avec toutes vos améliorations, puis publier une autre version stable et améliorée (version 2). Vous pouvez migrer la version 2 en production en remappant l'alias PROD pour qu'il pointe vers la version 2. Si cette version pose problème, vous pouvez facilement restaurer la version de production 1 en mappant une nouvelle fois l'alias PROD pour qu'il renvoie vers la version 1.

- Simplifier la gestion des mappages de source d'événement : au lieu d'utiliser les ARN (Amazon Resource Name) pour la fonction Lambda dans les mappages de source d'événement, vous pouvez utiliser un ARN d'alias. Cette approche signifie que vous n'avez pas besoin de mettre à jour vos mappages de

source d'événement lorsque vous migrez une nouvelle version ou lorsque vous restaurez une version précédente.

La fonction Lambda et l'alias sont des ressources AWS Lambda et, à l'instar de toutes les autres ressources AWS, ils possèdent un ARN unique. L'exemple suivant présente une fonction Lambda (version \$LATEST), avec une version publiée. Un alias renvoie vers chacune des versions.



Vous pouvez accéder à la fonction via son ARN ou celui de l'alias.

- Étant donné que la version d'une fonction non qualifiée correspond toujours à \$LATEST, vous pouvez y accéder à l'aide de l'ARN qualifié ou non qualifié de la fonction. L'exemple suivant illustre un ARN de fonction qualifié avec le suffixe de version \$LATEST.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:$LATEST
```

- Un ARN d'alias est un ARN qualifié. Chacun d'eux comporte un suffixe de nom d'alias.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:PROD
arn:aws:lambda:aws-region:acct-id:function:helloworld:BETA
arn:aws:lambda:aws-region:acct-id:function:helloworld:DEV
```

AWS Lambda fournit les opérations d'API suivantes pour vous permettre de créer et de gérer des alias :

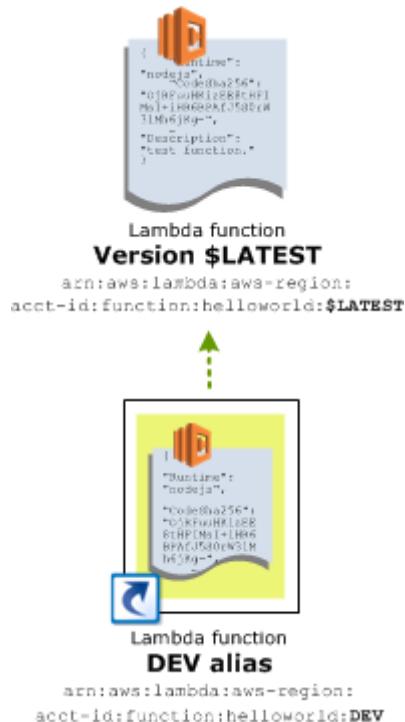
- [CreateAlias \(p. 402\)](#)
- [UpdateAlias \(p. 500\)](#)
- [GetAlias \(p. 431\)](#)
- [ListAliases \(p. 459\)](#)
- [DeleteAlias \(p. 418\)](#)

Exemple : Utilisation des alias pour gérer des versions de fonction Lambda

Voici un exemple de scénario relatif illustrant comment utiliser les versions et les alias pour migrer de nouvelles versions de fonctions Lambda en production.

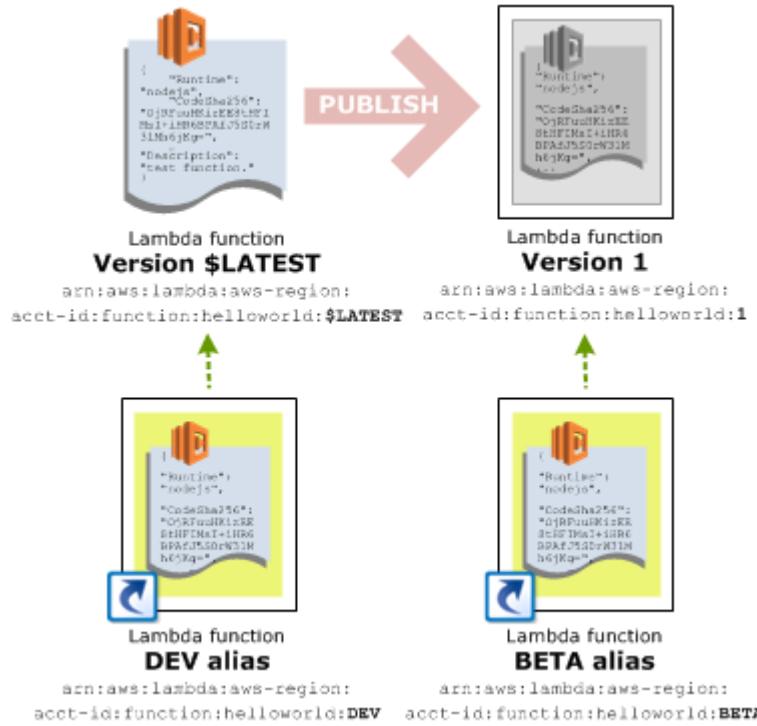
Au départ, vous créez une fonction Lambda.

La fonction que vous créez est la version \$LATEST. Vous créez également un alias (DEV, pour le développement) qui pointe vers la fonction que vous venez de créer. Les développeurs peuvent utiliser cet alias pour tester la fonction avec les sources d'événements dans un environnement de développement.



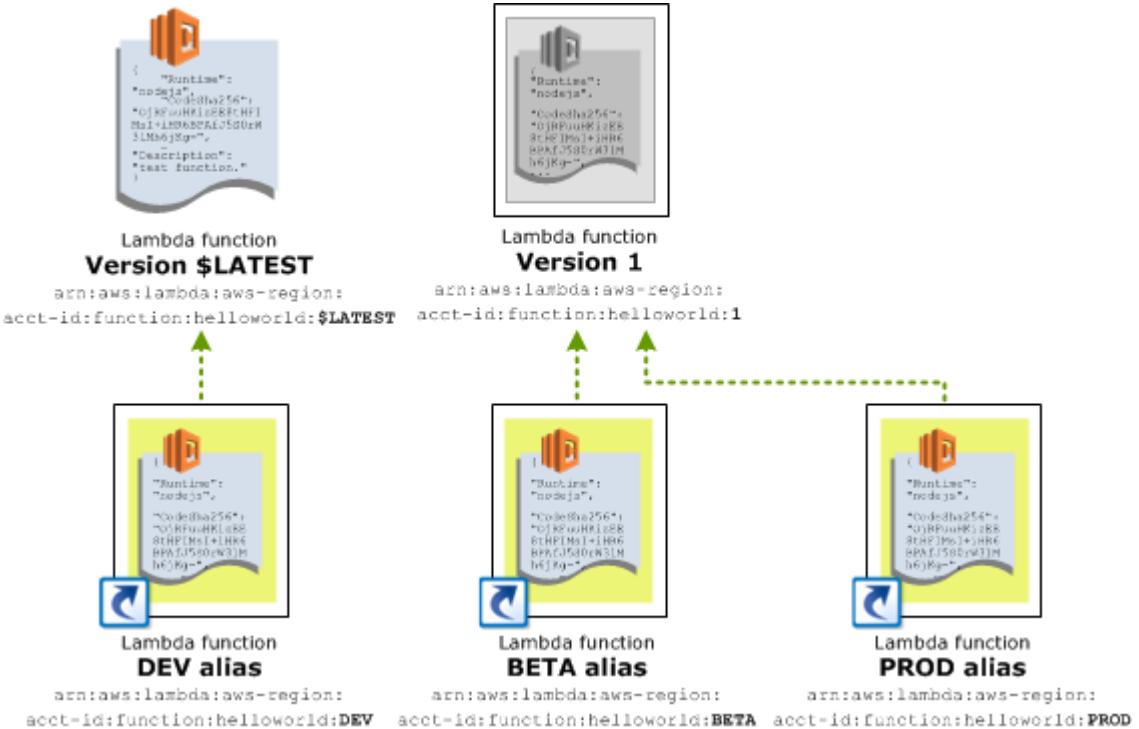
Lorsque vous testez la version de la fonction à l'aide des sources d'événements d'un environnement bêta de façon stable tout en continuant à développer des versions plus récentes.

Vous publiez une version à partir de la version \$LATEST avec un autre alias (BETA) qui renvoie vers cette dernière. Cette approche vous permet d'associer les sources d'événements bêta à cet alias spécifique. Dans les mappages de source d'événement, utilisez l'alias BETA pour associer la fonction Lambda à la source d'événement.



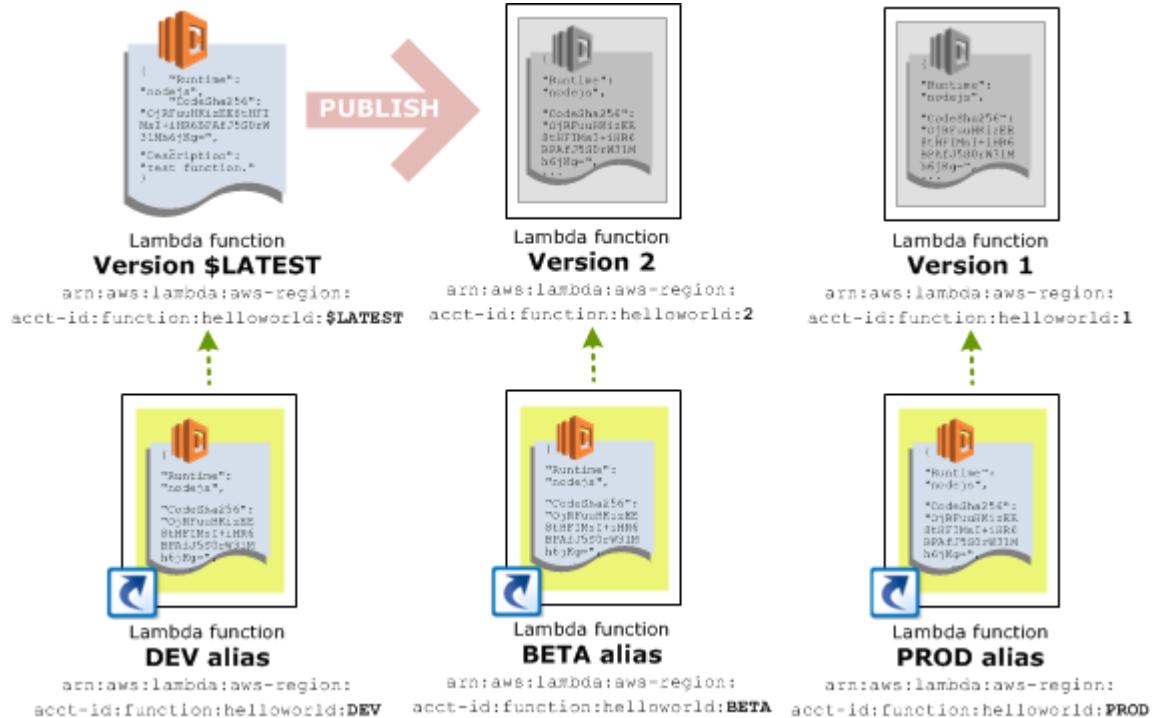
Vous migrez ensuite la version de la fonction Lambda en production afin d'exploiter les sources d'événements dans l'environnement de production.

Après avoir testé la version BETA de la fonction, vous pouvez créer un alias mappé à la version 1 pour définir la version de production. Dans cette approche, vous renvoyez les sources d'événements de production vers cette version spécifique. Pour ce faire, créez un alias PROD et utilisez l'ARN d'alias PROD ARN dans l'ensemble des mappages de source d'événement de production.



Vous poursuivez le développement, publiez davantage de versions et les testez.

Lorsque vous développez le code, vous pouvez mettre à jour la version `$LATEST`. Pour ce faire, vous pouvez importer le code mis à jour, puis le publier dans l'environnement de test bêta avec l'alias `BETA` pointant vers cette version. Ce remappage simple de l'alias `BETA` vous permet de transférer la version 2 de la fonction Lambda en bêta sans avoir à modifier les sources d'événements. Dans cette approche, les alias vous permettent de contrôler les versions de fonction utilisées avec des sources d'événements spécifiques dans votre environnement de développement.



Si vous voulez essayer de créer cette configuration via l'AWS Command Line Interface, consultez [Didacticiel : Utilisation des alias AWS Lambda \(p. 47\)](#).

Didacticiel : Utilisation des alias AWS Lambda

Basé sur l'AWS CLI, ce didacticiel crée des versions de fonction Lambda et des alias qui pointent vers elles, comme décrit dans la section [Exemple : Utilisation des alias pour gérer des versions de fonction Lambda \(p. 45\)](#).

Cet exemple utilise la région us-west-2 (USA Ouest (Oregon)) pour créer la fonction Lambda et les alias.

1. Créez un package de déploiement que vous pouvez importer pour créer la fonction Lambda :

a. Ouvrez un éditeur de texte, puis copiez le code suivant.

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, "message");
};
```

- b. Enregistrez le fichier sous le nom `helloworld.js`.
 - c. Compressez le fichier `helloworld.js` sous le nom `helloworld.zip`.
2. Créez un rôle AWS Identity and Access Management (IAM) (rôle d'exécution) que vous pouvez spécifier lors de la création de la fonction Lambda :
 - a. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
 - b. Pour créer un rôle IAM (rôle d'exécution), suivez les étapes [Rôles IAM](#) dans le IAM Guide de l'utilisateur. Lors de la création d'un rôle, prenez note des points suivants :
 - Pour Sélectionner un type de rôle, sélectionnez Rôles de service AWS, puis AWS Lambda.
 - Pour Attacher la stratégie, sélectionnez la stratégie nommée `AWSLambdaBasicExecutionRole`.
 - c. Prenez note de l'Amazon Resource Name (ARN) de ce rôle IAM. Vous en aurez besoin à l'étape suivante lorsque vous créerez la fonction Lambda.
 3. Créez une fonction Lambda (`helloworld`).

```
aws lambda create-function \
--region region \
--function-name helloworld \
--zip-file fileb://file-path/helloworld.zip \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler helloworld.handler \
--runtime nodejs6.10 \
--profile adminuser
```

La réponse renvoie les informations de configuration indiquant `$LATEST` en tant que version de fonction, comme l'illustre l'exemple suivant.

```
{  
    "CodeSha256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",  
    "FunctionName": "helloworld",  
    "CodeSize": 287,  
    "MemorySize": 128,  
    "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",  
    "Version": "$LATEST",  
    "Role": "arn:aws:iam::account-id:role/lambda_basic_execution",  
    "Timeout": 3,  
    "LastModified": "2015-09-30T18:39:53.873+0000",  
    "Handler": "helloworld.handler",  
    "Runtime": "nodejs6.10",  
    "Description": ""  
}
```

4. Créez un alias (DEV) qui pointe vers la version `$LATEST` de la fonction `helloworld` d'Lambda :

```
aws lambda create-alias \
--region region \
--function-name helloworld \
--description "sample alias" \
--function-version "\$LATEST" \
--name DEV \
--profile adminuser
```

La réponse renvoie les informations concernant l'alias, y compris la version de fonction vers laquelle il pointe et l'ARN de l'alias. Cet ARN est identique à la celui de la fonction ARN, avec un suffixe de nom d'alias en plus. Voici un exemple de réponse.

```
{  
    "AliasArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld:DEV",  
    "FunctionVersion": "$LATEST",  
    "Name": "DEV",  
    "Description": "sample alias"  
}
```

- Publiez une version de la fonction helloworld d'Lambda.

```
aws lambda publish-version \  
--region region \  
--function-name helloworld \  
--profile adminuser
```

La réponse renvoie les informations de configuration de la version de la fonction, y compris le numéro de version et l'ARN de la fonction avec le suffixe de version. Voici un exemple de réponse.

```
{  
    "CodeSha256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",  
    "FunctionName": "helloworld",  
    "CodeSize": 287,  
    "MemorySize": 128,  
    "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld:1",  
    "Version": "1",  
    "Role": "arn:aws:iam::account-id:role/lambda_basic_execution",  
    "Timeout": 3,  
    "LastModified": "2015-10-03T00:48:00.435+0000",  
    "Handler": "helloworld.handler",  
    "Runtime": "nodejs6.10  
",  
    "Description": ""  
}
```

- Créez un alias (BETA) pour la version 1 de la fonction helloworld d'Lambda.

```
aws lambda create-alias \  
--region region \  
--function-name helloworld \  
--description "sample alias" \  
--function-version 1 \  
--name BETA \  
--profile adminuser
```

Vous avez désormais deux alias pour la fonction helloworld. L'alias DEV renvoie vers la version de fonction \$LATEST tandis que l'alias BETA pointe vers la version 1 de la fonction Lambda.

- Supposons que vous souhaitez migrer la version 1 de la fonction helloworld en production. Créez un autre alias (PROD) qui pointe vers la version 1.

```
aws lambda create-alias \  
--region region \  
--function-name helloworld \  
--description "sample alias" \  
--function-version 1 \  
--name PROD \  
--profile adminuser
```

Pour l'instant, les alias BETA et PROD pointent tous les deux vers la version 1 de la fonction Lambda.

8. Vous pouvez désormais publier une version plus récente (par exemple, la version 2), mais vous devez d'abord mettre à jour le code et importer un package de déploiement modifié. Si la version \$LATEST n'est pas modifiée, vous ne pouvez pas publier plus d'une version de celle-ci. En supposant que vous avez mis à jour le package de déploiement, que vous l'avez chargé et que vous avez publié la version 2, vous pouvez désormais modifier l'alias BETA pour qu'il renvoie vers la version 2 de la fonction Lambda.

```
aws lambda update-alias \  
--region region \  
--function-name helloworld \  
--function-version 2 \  
--name BETA \  
--profile adminuser
```

Maintenant, vous avez trois alias pointant vers une version distincte de la fonction Lambda : l'alias DEV renvoie vers la version \$LATEST, l'alias BETA vers la version 2 et l'alias PROD vers la version 1 de la fonction Lambda.

Pour en savoir plus sur l'utilisation de la console AWS Lambda pour gérer les versions, consultez [Gestion des versions via AWS Management Console, l'AWS CLI ou les opérations d'API Lambda \(p. 52\)](#).

Accorder des autorisations dans un modèle Push

Dans un modèle push (consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#)), les sources d'événement comme Amazon S3 appellent la fonction Lambda. Ces sources d'événements gèrent un mappage qui identifie la version de fonction ou l'alias appelé lorsque des événements se produisent. Notez bien ce qui suit :

- Nous vous recommandons de spécifier un alias de fonction Lambda existant dans la configuration du mappage (consultez [Présentation des alias AWS Lambda \(p. 43\)](#)). Par exemple, si la source d'événement est Amazon S3, vous spécifiez l'ARN de l'alias dans la configuration des notifications du compartiment pour qu'Amazon S3 puisse appeler cet alias quand il détecte des événements spécifiques.
- Dans le modèle push, vous accordez des autorisations aux sources d'événement à l'aide d'une stratégie de ressource que vous associez à la fonction Lambda. Dans les versions, les autorisations que vous ajoutez sont spécifiques à la qualification que vous définissez dans la requête AddPermission (voir [Versions, alias et stratégies de ressources \(p. 51\)](#)).

Par exemple, la commande AWS CLI suivante accorde à Amazon S3 les autorisations lui permettant d'appeler l'alias PROD de la fonction helloworld d'Lambda (notez que le paramètre --qualifier spécifie le nom de l'alias).

```
aws lambda add-permission \  
--region region \  
--function-name helloworld \  
--qualifier PROD \  
--statement-id 1 \  
--principal s3.amazonaws.com \  
--action lambda:InvokeFunction \  
--source-arn arn:aws:s3:::examplebucket \  
--source-account 111111111111 \  
--profile adminuser
```

Dans ce cas, Amazon S3 est désormais en mesure d'appeler l'alias PROD, et AWS Lambda peut ensuite exécuter la fonction helloworld d'Lambda vers laquelle renvoie l'alias PROD. Pour que cela fonctionne, vous devez spécifier l'ARN de l'alias PROD dans la configuration des notifications du compartiment S3.

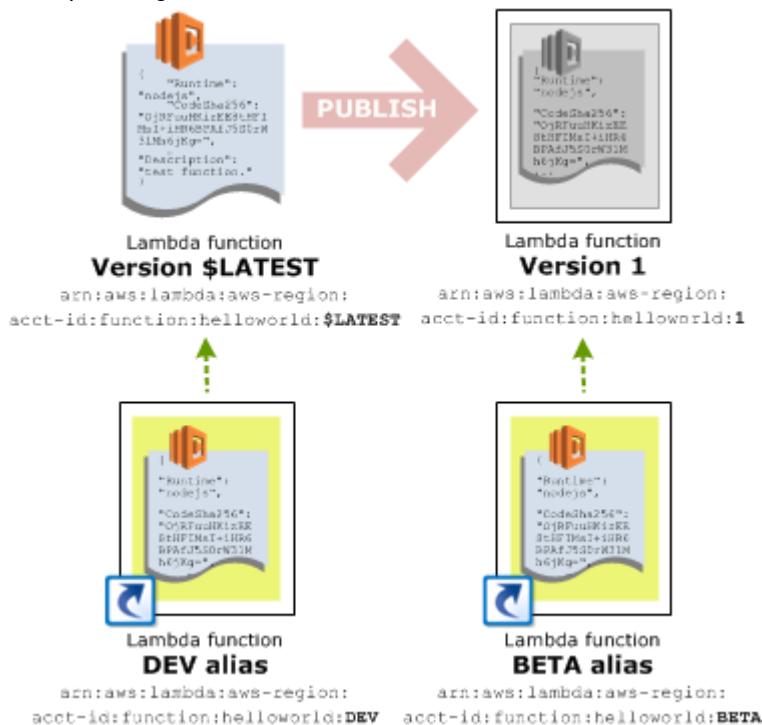
Pour en savoir plus sur la gestion des événements Amazon S3, consultez [Didacticiel : Utilisation d'AWS Lambda avec Amazon S3 \(p. 128\)](#).

Note

Si vous utilisez la console AWS Lambda pour ajouter une source d'événements à la fonction Lambda, la console ajoute les autorisations nécessaires pour vous.

Versions, alias et stratégies de ressources

Avec les versions et les alias, vous pouvez accéder à une fonction Lambda à l'aide de différents ARN. Par exemple, imaginez le scénario suivant.



Vous pouvez appeler, par exemple, la version 1 de la fonction `helloworld` avec les deux ARN suivants :

- Utilisation de l'ARN de fonction qualifié comme indiqué ci-dessous.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

- Utilisation de l'ARN d'alias BETA comme indiqué ci-dessous.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:BETA
```

Dans un modèle push, les sources d'événements (par exemple, les applications personnalisées et Amazon S3) peuvent appeler les versions de fonction Lambda tant que vous accordez à ces sources d'événements les autorisations nécessaires à l'aide d'une stratégie d'accès associée à la fonction Lambda. Pour en savoir plus sur le modèle push, consultez la section [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

En supposant que vous accordé les autorisations requises, une autre question se pose : « une source d'événement peut-elle appeler une version de fonction avec l'un des ARN associés ? » Cela

dépend de la façon dont vous avez identifié la fonction dans la requête d'ajout des autorisations (voir [AddPermission \(p. 398\)](#)). Pour être plus précis, sachez que l'autorisation que vous accordez s'applique uniquement à l'ARN utilisé dans la requête d'ajout de cette autorisation :

- Si vous utilisez un nom de fonction qualifié (tel que `helloworld:1`), l'autorisation permet d'appeler la version 1 de la fonction `helloworld` uniquement avec son ARN qualifié (l'utilisation d'autre ARN entraîne une erreur d'autorisation).
- Si vous utilisez un nom d'alias (par exemple, `helloworld:BETA`), l'autorisation permet uniquement d'appeler la fonction `helloworld` avec l'ARN d'alias BETA (l'utilisation d'autre ARN entraîne une erreur d'autorisation, y compris l'ARN de la version de fonction vers laquelle pointe l'alias).
- Si vous utilisez un nom de fonction non qualifié (par exemple, `helloworld`), l'autorisation permet uniquement d'appeler la fonction `helloworld` avec l'ARN de fonction non qualifié (l'utilisation d'autre ARN entraînera une erreur d'autorisation).

Note

Notez que même si la stratégie d'accès ne concerne que l'ARN non qualifié, le code et la configuration de la fonction Lambda appellée viennent de la version de fonction `$LATEST`. L'ARN de fonction non qualifié est mappé avec la version `$LATEST`, mais les autorisations que vous ajoutez sont spécifiques à l'ARN.

- Si vous utilisez un nom de fonction qualifié avec la version `$LATEST` (`helloworld:$LATEST`), l'autorisation permet d'appeler la version `$LATEST` de la fonction `helloworld` uniquement à l'aide de son ARN qualifié (l'utilisation d'un ARN non qualifié entraîne une erreur d'autorisation).

Gestion des versions via AWS Management Console, l'AWS CLI ou les opérations d'API Lambda

Vous pouvez gérer les versions des fonctions Lambda par programmation à l'aide des kits SDK AWS (ou effectuer directement les appels d'API AWS Lambda, si nécessaire), de l'AWS Command Line Interface(AWS CLI) ou de la console AWS Lambda.

AWS Lambda fournit les API suivantes pour gérer les versions et les alias :

[PublishVersion \(p. 482\)](#)

[ListVersionsByFunction \(p. 475\)](#)

[CreateAlias \(p. 402\)](#)

[UpdateAlias \(p. 500\)](#)

[DeleteAlias \(p. 418\)](#)

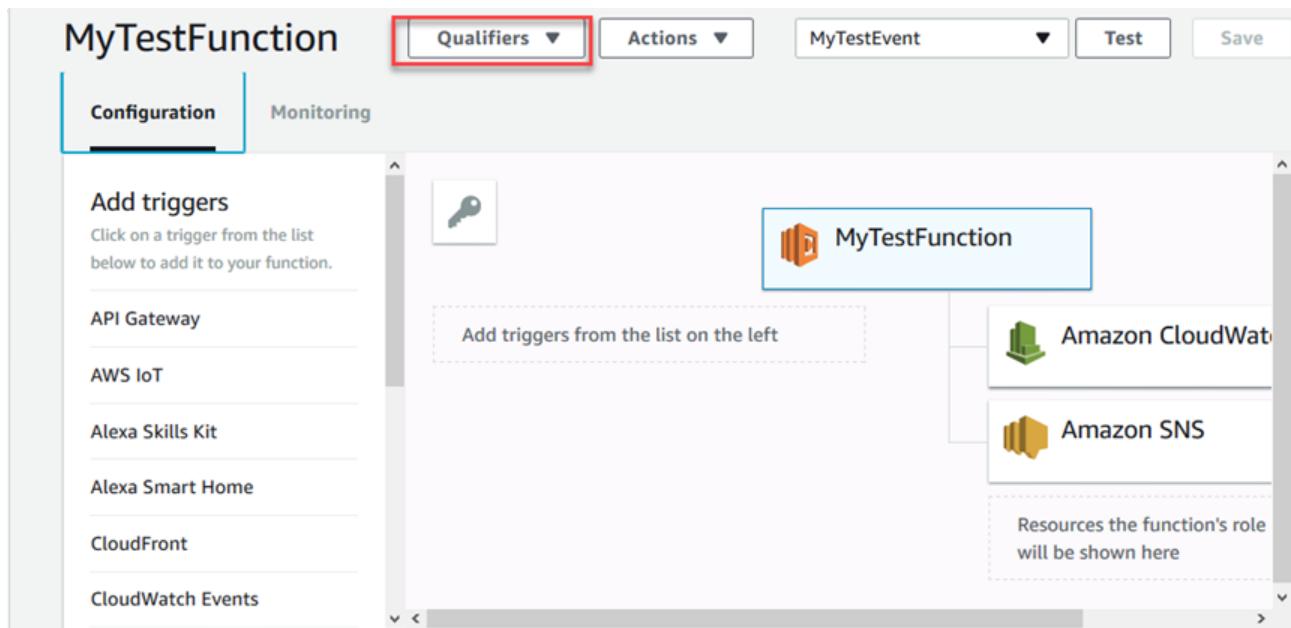
[GetAlias \(p. 431\)](#)

[ListAliases \(p. 459\)](#)

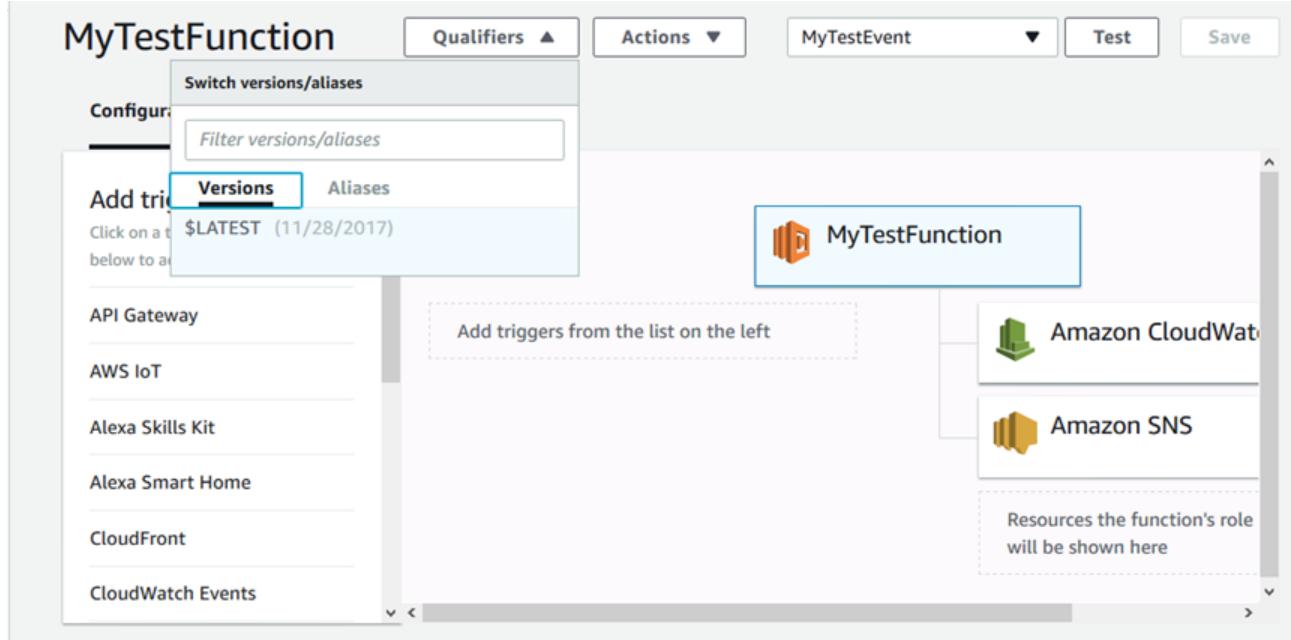
En plus de ces API, des API spécifiques prennent également en charge les opérations liées aux versions.

Pour voir un exemple de la façon dont vous pouvez utiliser l'AWS CLI, consultez [Didacticiel : Utilisation des alias AWS Lambda \(p. 47\)](#).

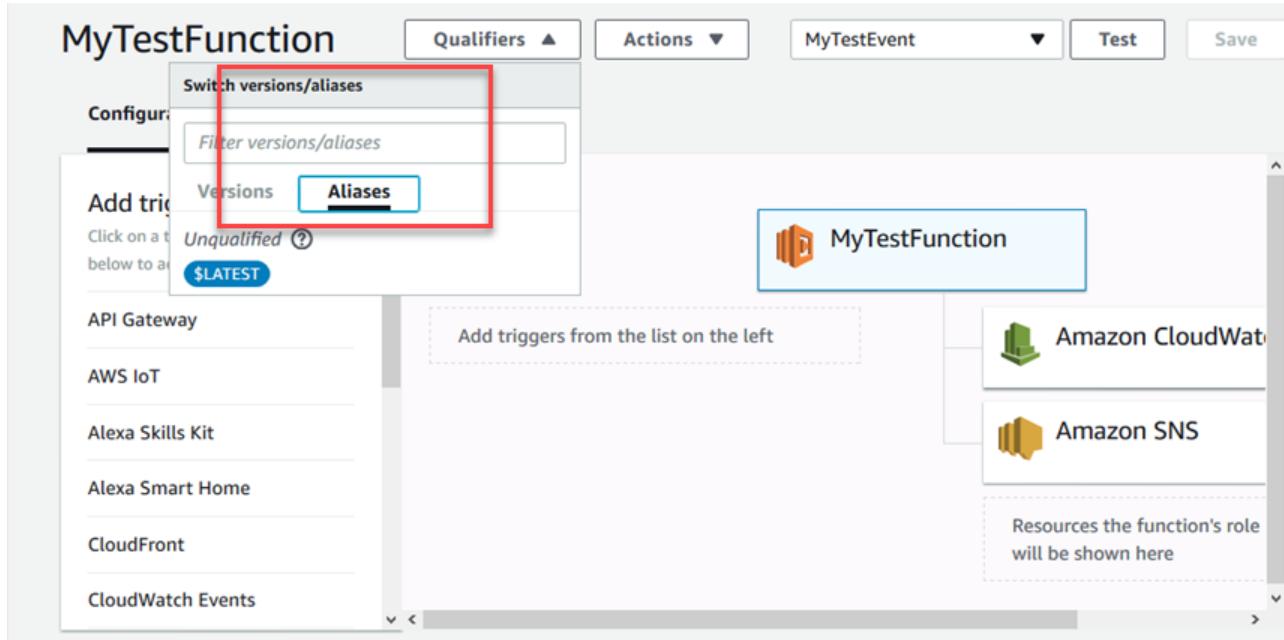
Celle-ci explique comment utiliser la console AWS Lambda pour gérer les versions. Dans la console AWS Lambda, sélectionnez une fonction, puis cliquez sur Qualificateurs.



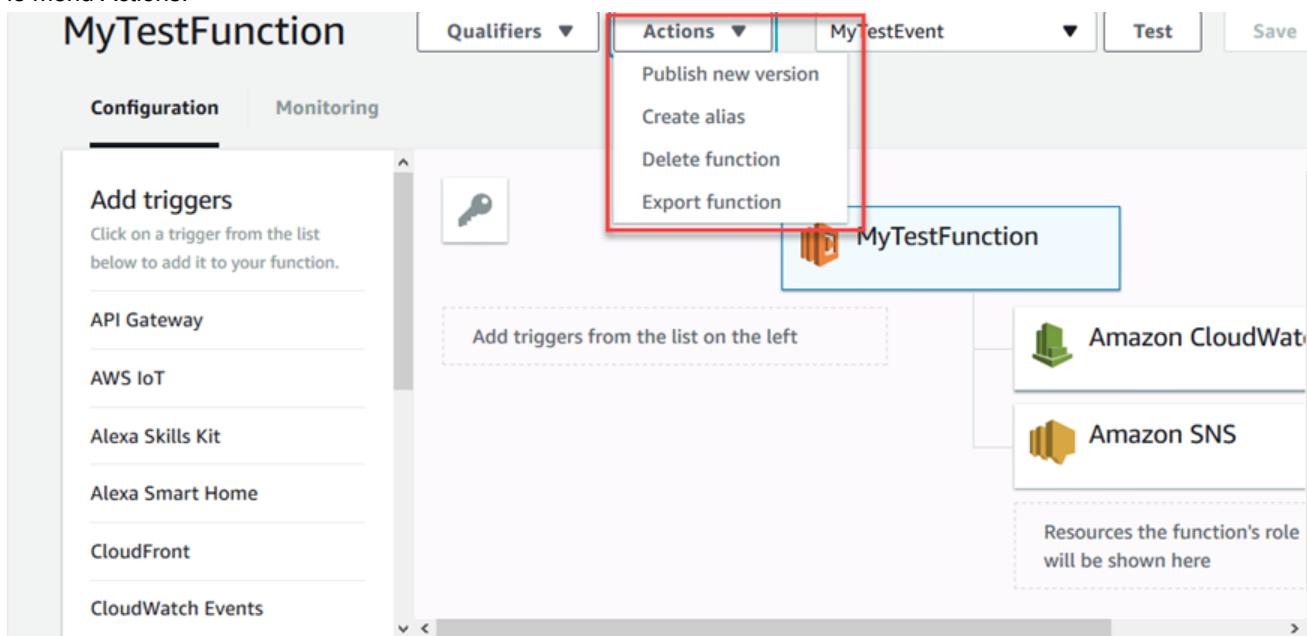
Le menu Qualifiers développé affiche les onglets Versions et Alias onglet, comme illustré dans la capture d'écran suivante. Le volet Versions présente une liste des versions de la fonction sélectionnée. Si vous n'avez pas encore publié de version pour la fonction sélectionnée, le volet Versions répertorie uniquement la version \$LATEST, comme illustré ci-après.



Sélectionnez l'onglet Alias pour afficher une liste des alias de la fonction. Au départ, vous n'avez aucun alias, comme illustré ci-après.



Vous pouvez désormais publier une version ou créer des alias pour la fonction Lambda sélectionnée avec le menu Actions.



Pour en savoir plus sur les versions et les alias, consultez la page [Versions et alias des fonctions AWS Lambda \(p. 38\)](#).

Déplacement du trafic à l'aide des alias

Par défaut, un alias pointe vers une seule version de fonction Lambda. Lorsque l'alias est mis à jour afin de pointer vers une autre version de fonction, le trafic des demandes entrantes pointe instantanément vers la version mise à jour. L'alias est alors exposé à d'éventuelles instabilités liées à la nouvelle version. Pour réduire cet impact, vous pouvez implémenter le paramètre `routing-config` de l'alias Lambda qui vous

permet de pointer vers deux versions différentes de la fonction Lambda et imposer le pourcentage de trafic entrant envoyé à chaque version.

Par exemple, vous pouvez spécifier que seulement 2 % du trafic entrant est acheminé vers la nouvelle version le temps de vérifier si cette version est prête pour un environnement de production, et que les 98 % restants sont acheminés vers la version d'origine. Au fur et à mesure que la nouvelle version évolue, vous pouvez progressivement mettre à jour ce rapport en fonction de vos besoins, jusqu'à ce que vous ayez validé la stabilité de la nouvelle version. Vous pouvez ensuite mettre à jour l'alias afin d'acheminer l'ensemble du trafic vers la nouvelle version.

Note

Vous pouvez faire pointer un alias vers deux versions de fonction Lambda maximum. En outre :

- Les deux versions doivent avoir la même configuration [Files d'attente Lettre Morte \(p. 81\)](#) (ou aucune configuration de file d'attente de lettres mortes).
- Les deux versions doivent avoir le même rôle d'exécution IAM.
- Lorsqu'un alias est configuré pour pointer vers plusieurs versions, l'alias ne peut pas pointer vers `$LATEST`.

Déplacement du trafic à l'aide d'un alias (interface de ligne de commande)

Pour configurer un alias afin qu'il redirige le trafic vers deux versions de fonction selon des pondérations en utilisant l'opération [CreateAlias \(p. 402\)](#), vous devez configurer le paramètre `routing-config`. L'exemple suivant fait pointer un alias vers deux versions de fonction Lambda différentes. La version 2 reçoit 2 % du trafic et la version 1 98 %.

```
aws lambda create-alias --name alias name --function-name function-name \\ --function-version 1  
--routing-config AdditionalVersionWeights={"2":0.02}
```

Vous pouvez mettre à jour le pourcentage de trafic entrant à diriger votre nouvelle version (version 2) à l'aide de l'opération [UpdateAlias \(p. 500\)](#). Par exemple, vous pouvez augmenter le trafic dirigé vers votre nouvelle version à 5 %, comme indiqué ci-dessous.

```
aws lambda update-alias --name alias name --function-name function-name \\  
--routing-config AdditionalVersionWeights={"2":0.05}
```

Pour acheminer l'ensemble du trafic vers la version 2, utilisez à nouveau l'opération [UpdateAlias](#) pour modifier la propriété `function-version` afin de faire pointer vers la version 2. Définissez ensuite le paramètre `routing-config` sur une chaîne vide, comme indiqué ci-dessous.

```
aws lambda update-alias --name alias name --function-name function-name \\  
--function-version 2 --routing-config ''
```

Déplacement du trafic à l'aide d'un alias (console)

Vous pouvez configurer le déplacement du trafic avec un alias en utilisant la console Lambda, comme décrit ci-dessous :

1. Ouvrez votre fonction Lambda et vérifiez que vous disposez d'au moins deux versions déjà publiées. Si ce n'est pas le cas, vous pouvez consulter [Présentation des versions AWS Lambda \(p. 40\)](#) pour en savoir plus sur la gestion des versions et publier votre première version de fonction.

2. Pour Actions, choisissez Create alias.
3. Dans la fenêtre Créer un alias, indiquez une valeur pour le champ Nom*, et éventuellement pour les champs Description et Version*, de la fonction Lambda vers laquelle pointe l'alias. Ici, il s'agit de la version 1.
4. Dans Additional version, spécifiez les éléments suivants :
 - a. Spécifiez une seconde version de fonction Lambda.
 - b. Saisissez une valeur de pondération pour la fonction. La pondération est le pourcentage de trafic qui est affecté à cette version lorsque l'alias est appelé. La première version reçoit le reste du trafic. Par exemple, si vous spécifiez 10 % pour Additional version, 90 % du trafic est automatiquement affecté à la première version.
5. Sélectionnez Create.

Détermination de la version appelée

Lorsque votre alias déplace le trafic entre deux versions de fonction, vous pouvez déterminer de deux manières quelle version de fonction Lambda a été appelée :

1. CloudWatch Logs : Lambda émet automatiquement une entrée de journal START contenant l'ID de la version appelée dans CloudWatch Logs pour chaque appel de fonction. Un exemple suit.

```
19:44:37 START RequestId: request id Version: $version
```

Lambda utilise la dimension Executed Version pour filtrer les données de métriques en fonction de la version exécutée. S'applique uniquement aux appels d'alias. Pour plus d'informations, consultez [Dimensions AWS Lambda CloudWatch \(p. 231\)](#).

2. Charge utile de réponse (appels synchrones) : les réponses aux appels de fonction synchrones incluent un en-tête **x-amz-executed-version** indiquant la version de fonction appelée.

Couches AWS Lambda

Vous pouvez configurer votre fonction Lambda pour extraire le code supplémentaire et le contenu sous la forme de couches. Une couche est une archive ZIP qui contient des bibliothèques, un [runtime personnalisé \(p. 107\)](#) ou d'autres dépendances. Avec les couches, vous pouvez utiliser les bibliothèques dans votre fonction sans avoir besoin de les inclure dans votre package de déploiement.

Les couches vous permettent de maintenir votre package de déploiement petit, ce qui facilite le développement. Vous pouvez éviter les erreurs qui peuvent se produire lorsque vous installez et regroupez les dépendances avec le code de votre fonction. Pour les fonctions Node.js, Python et Ruby, vous pouvez [développer le code de votre fonction dans la console Lambda \(p. 17\)](#) tant que vous maintenez votre package de déploiement sous 3 MB.

Note

Une fonction peut utiliser un maximum de 5 layers à la fois. La taille totale décompressée de la fonction avec toutes les couches ne peut pas dépasser la taille limite du package de déploiement décompressé de 250 MB. Pour en savoir plus, consultez [Limites AWS Lambda \(p. 288\)](#).

Vous pouvez créer des couches ou utiliser des couches publiées par AWS d'autres clients AWS. Les couches prennent en charge les [stratégies basées sur les ressources \(p. 60\)](#) pour l'octroi des comptes AWS spécifiques aux autorisations, [AWS Organizations](#) ou tous les comptes.

Les couches sont extraites dans le répertoire /opt de l'environnement d'exécution de la fonction. Chaque runtime recherche les bibliothèques dans un autre emplacement sous /opt, en fonction du langage.

[Structurez votre couche \(p. 59\)](#) afin que votre code de fonction puisse accéder aux bibliothèques sans configuration supplémentaire.

Vous pouvez également utiliser Modèle d'application sans serveur AWS (AWS SAM) pour gérer les couches et la configuration des couches de votre fonction. Pour plus d'informations, consultez la section [Déclarer les ressources sans serveur](#) dans le Modèle d'application sans serveur AWS Manuel du développeur.

Sections

- [Configuration d'une fonction pour utiliser les couches \(p. 57\)](#)
- [Gestion des couches \(p. 58\)](#)
- [Inclusion de dépendances de bibliothèques dans une couche \(p. 59\)](#)
- [Autorisations d'utilisation des couches \(p. 60\)](#)

Configuration d'une fonction pour utiliser les couches

Vous pouvez spécifier jusqu'à 5 layers dans la configuration de votre fonction, pendant ou après la création de la fonction. Vous choisissez une version spécifique d'une couche à utiliser. Si vous souhaitez utiliser une autre version ultérieurement, actualisez la configuration de votre fonction.

Pour ajouter des couches dans votre fonction, utilisez la commande `update-function-configuration`. L'exemple suivant ajoute deux couches : une à partir du même compte que la fonction, et l'autre à partir d'un autre compte.

```
$ aws lambda update-function-configuration --function-name my-function \
--layers arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3
` 
arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2
{
    "FunctionName": "test-layers",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs8.10",
    "Role": "arn:aws:iam::123456789012:role/service-role/lambda-role",
    "Handler": "index.handler",
    "CodeSize": 402,
    "Description": "",
    "Timeout": 5,
    "MemorySize": 128,
    "LastModified": "2018-11-14T22:47:04.542+0000",
    "CodeSha256": "kDHALEY62Ni3OovMwVO8tNvgbRoRa6I0OKqShm7bSWF4=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "81cc64f5-5772-449a-b63e-12330476bcc4",
    "Layers": [
        {
            "Arn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3",
            "CodeSize": 169
        },
        {
            "Arn": "arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2",
            "CodeSize": 169
        }
    ]
}
```

Vous devez spécifier la version de chaque couche à utiliser en fournissant l'ARN complet de la version de la couche. Lorsque vous ajoutez des couches à une fonction qui possède déjà des couches, la liste

précédente est remplacée par la nouvelle. Incluez toutes les couches chaque fois que vous mettez à jour la configuration de la couche. Pour supprimer toutes les couches, spécifiez une liste vide.

```
$ aws lambda update-function-configuration --function-name my-function --layers []
```

Votre fonction peut accéder au contenu de la couche lors de l'exécution dans le répertoire /opt. Les couches sont appliquées dans l'ordre spécifié, avec la fusion de tous les dossiers portant le même nom. Si le même fichier apparaît dans plusieurs couches, la version dans la dernière couche appliquée est utilisée.

Le créateur d'une couche peut supprimer la version de la couche que vous utilisez. Dans ce cas, votre fonction continue de s'exécuter comme si la version de la couche existait toujours. Toutefois, lorsque vous mettez à jour la configuration de la couche, vous devez supprimer la référence à la version supprimée.

Gestion des couches

Pour créer une couche, utilisez la commande `publish-layer-version` avec un nom, une description, une archive ZIP et une liste des [runtimes \(p. 103\)](#) qui sont compatibles avec la couche. La liste des runtimes est facultative, mais elle rend la couche plus facile à détecter.

```
$ aws lambda publish-layer-version --layer-name my-layer --description "My layer" --license-info "MIT" \
--content S3Bucket=lambda-layers-us-east-2-123456789012,S3Key=layer.zip --compatible-runtimes python3.6 python3.7
{
    "Content": {
        "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-layer-4aaa2fbb-ff77-4b0a-ad92-5b78a716a96a?
versionId=27iWyA73cCAYqyh...",
        "CodeSha256": "tv9jJO+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",
        "CodeSize": 169
    },
    "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
    "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:1",
    "Description": "My layer",
    "CreatedDate": "2018-11-14T23:03:52.894+0000",
    "Version": 1,
    "LicenseInfo": "MIT",
    "CompatibleRuntimes": [
        "python3.6",
        "python3.7"
    ]
}
```

Chaque fois que vous appelez `publish-layer-version`, vous créez une nouvelle version. Les fonctions qui utilisent la couche font référence directement à une version de la couche. Vous pouvez [configurer des autorisations \(p. 60\)](#) sur une version de couche existante, mais pour effectuer d'autres modifications, vous devez créer une nouvelle version.

Pour trouver les couches qui sont compatibles avec le runtime de votre fonction, utilisez la commande `list-layers`.

```
$ aws lambda list-layers --compatible-runtime python3.7
{
    "Layers": [
        {
            "LayerName": "my-layer",
            "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
            "LatestMatchingVersion": {
                "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
            }
        }
    ]
}
```

```
"Version": 2,  
"Description": "My layer",  
"CreatedDate": "2018-11-15T00:37:46.592+0000",  
"CompatibleRuntimes": [  
    "python3.6",  
    "python3.7"  
]  
}  
]  
}
```

Vous pouvez omettre l'option de runtime pour répertorier toutes les couches. Les détails dans la réponse reflètent la version la plus récente de la couche. Consultez toutes les versions d'une couche avec `list-layer-versions`. Pour obtenir plus d'informations sur une version, utilisez `get-layer-version`.

```
$ aws lambda get-layer-version --layer-name my-layer --version-number 2  
{  
    "Content": {  
        "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/  
snapshots/123456789012/my-layer-91e9ea6e-492d-4100-97d5-a4388d442f3f?  
versionId=GmvvP.3090EpkfnN...",  
        "CodeSha256": "tv9jJO+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",  
        "CodeSize": 169  
    },  
    "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",  
    "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:2",  
    "Description": "My layer",  
    "CreatedDate": "2018-11-15T00:37:46.592+0000",  
    "Version": 2,  
    "CompatibleRuntimes": [  
        "python3.6",  
        "python3.7"  
    ]  
}
```

Le lien dans la réponse vous permet de télécharger l'archive de la couche. Il reste valide pendant 10 minutes. Pour supprimer une version de couche, utilisez la commande `delete-layer-version`.

```
$ aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Lorsque vous supprimez une version de couche, vous ne pouvez plus configurer des fonctions pour l'utiliser. En revanche, toute fonction qui utilise déjà la version continue d'y avoir accès.

Inclusion de dépendances de bibliothèques dans une couche

Vous pouvez déplacer les dépendances de runtime de votre code de fonction en les plaçant dans une couche. Les runtimes Lambda incluent des chemins d'accès dans le répertoire `/opt` afin de s'assurer que le code de votre fonction ait accès aux bibliothèques qui sont incluses dans les couches.

Pour inclure des bibliothèques dans une couche, placez-les dans l'un des dossiers pris en charge par votre runtime.

- Node.js – `nodejs/node_modules`, `nodejs/node8/node_modules` (`NODE_PATH`)

Example Kit SDK AWS X-Ray pour Node.js

```
xray-sdk.zip
```

```
# nodejs/node_modules/aws-xray-sdk
```

- Python – python, python/lib/python3.7/site-packages (répertoires du site)

Example Pillow

```
pillow.zip
# python/PIL
# python/Pillow-5.3.0.dist-info
```

- Java – java/lib (chemin de classe)

Example Jackson

```
jackson.zip
# java/lib/jackson-core-2.2.3.jar
```

- Ruby – ruby/gems/2.5.0 (GEM_PATH), ruby/lib (RUBY_LIB)

Example JSON

```
json.zip
# ruby/gems/2.5.0/json
```

- Tous – bin (PATH), lib (LD_LIBRARY_PATH)

Example JQuery

```
jq.zip
# bin/jq
```

Pour plus d'informations sur les paramètres des chemins d'accès dans l'environnement d'exécution Lambda, consultez [Environnement d'exécution Lambda et bibliothèques disponibles \(p. 104\)](#).

Autorisations d'utilisation des couches

Les autorisations d'utilisation des couches sont gérées sur la ressource. Pour configurer une fonction à une couche, vous avez besoin de l'autorisation d'appeler `GetLayerVersion` sur la version de la couche. Pour les fonctions dans votre compte, vous pouvez obtenir cette autorisation à partir de votre [stratégie d'utilisateur \(p. 244\)](#) ou à partir de la [stratégie basée sur les ressources \(p. 257\)](#) de la fonction. Pour utiliser une couche dans un autre compte, vous avez besoin d'une autorisation sur votre stratégie d'utilisateur, et le propriétaire de l'autre compte doit accorder l'autorisation à votre compte avec une stratégie basée sur les ressources.

Pour accorder une autorisation d'utilisation de la couche à un autre compte, ajoutez une instruction à la stratégie d'autorisations de la version de couche avec la commande `add-layer-version-permission`. Dans chaque instruction, vous pouvez accorder une autorisation à un compte unique, à tous les comptes ou à une organisation.

L'exemple suivant accorde à tous les comptes d'une organisation l'autorisation d'utiliser la version 3 d'une couche.

```
$ aws lambda add-layer-version-permission --layer-name my-layer \
--statement-id engineering-org --version-number 3 --principal '*' \
--action lambda:GetLayerVersion --organization-id o-t194hfs8cz
{
```

```
"Statement": "{\"Sid\":\"engineering-org\", \"Effect\":\"Allow\", \"Principal\\\":\"*\", \"Action\":\"lambda:GetLayerVersion\", \"Resource\":\"arn:aws:lambda:us-west-2:123456789012:layer:my-layer:3\", \"Condition\":{\"StringEquals\":{\\\"aws:PrincipalOrgID\\\":\\\"o-t194hfs8cz\\\"}}},\n    \"RevisionId\": \"b0cd9796-d4eb-4564-939f-de7fe0b42236\"\n}
```

Pour octroyer une autorisation à tous les comptes AWS, utilisez * pour le mandataire, et omettez l'ID de l'organisation. Pour octroyer une autorisation à un compte unique, utilisez le numéro de compte du mandataire. Pour plusieurs comptes ou organisations, ajoutez plusieurs instructions.

Les autorisations ne s'appliquent qu'à une seule version d'une couche. Répétez la procédure chaque fois que vous créez une nouvelle version de la couche.

Utilisez la commande `get-layer-version-policy` pour afficher les autorisations sur une couche, et `remove-layer-version-permission` pour supprimer les instructions de la stratégie.

```
$ aws lambda get-layer-version-policy --layer-name my-layer --version-number 3\n{\n    \"Statement\": {\"Sid\":\"engineering-org\", \"Effect\":\"Allow\", \"Principal\\\":\"*\", \"Action\":\"lambda:GetLayerVersion\", \"Resource\":\"arn:aws:lambda:us-west-2:123456789012:layer:my-layer:3\", \"Condition\":{\"StringEquals\":{\\\"aws:PrincipalOrgID\\\":\\\"o-t194hfs8cz\\\"}}},\n        \"RevisionId\": \"b0cd9796-d4eb-4564-939f-de7fe0b42236\"\n}\n$ aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3 --\nstatement-id engineering-org
```

Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC

Généralement, vous créez des ressources dans Amazon Virtual Private Cloud (Amazon VPC) pour qu'elles ne soient pas accessibles via le réseau Internet public. Il peut s'agir de ressources des services AWS, telles que des entrepôts de données Amazon Redshift, des clusters Amazon ElastiCache ou des instances Amazon RDS. Il peut également s'agir de vos propres services qui s'exécutent sur vos propres instances EC2. Par défaut, les ressources au sein d'un VPC ne sont pas accessibles à partir d'une fonction Lambda.

AWS Lambda exécute le code de la fonction de manière sécurisée au sein d'un VPC par défaut. Néanmoins, pour permettre à votre fonction Lambda d'accéder à des ressources au sein de votre VPC privé, vous devez fournir des informations de configuration supplémentaires spécifiques du VPC, en particulier les ID de sous-réseau VPC et les ID de groupe de sécurité. AWS Lambda utilise ces informations pour configurer les interfaces réseau Elastic (ENI) qui permettent à votre fonction de se connecter en toute sécurité à d'autres ressources au sein de votre VPC privé.

AWS Lambda ne prend pas en charge la connexion aux ressources au sein des VPC à location dédiée. Pour plus d'informations, consultez [VPC dédiés](#).

Configuration d'une fonction Lambda pour l'accès à un Amazon VPC

Pour ajouter des informations relatives au VPC à la configuration de la fonction Lambda, utilisez le paramètre `VpcConfig` lorsque vous créez cette fonction (voir [CreateFunction \(p. 410\)](#)). Vous pouvez également les ajouter à une configuration de fonction Lambda existante (voir [UpdateFunctionConfiguration \(p. 514\)](#)). Voici quelques exemples de commandes AWS CLI :

- La commande CLI `create-function` spécifie le paramètre `--vpc-config` pour fournir les informations du VPC lorsque vous créez une fonction Lambda.

```
$ aws lambda create-function \
--function-name ExampleFunction \
--runtime go1.x \
--role execution-role-arn \
--zip-file fileb://path/app.zip \
--handler app.handler \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=comma-separated-security-group-ids \
--memory-size 1024
```

Note

Le rôle d'exécution de la fonction Lambda doit disposer des autorisations requises pour créer, décrire et supprimer des ENI. AWS Lambda fournit une stratégie d'autorisations, `AWSLambdaVPCAccessExecutionRole`, avec les autorisations requises pour les actions EC2 nécessaires (`ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces` et `ec2:DeleteNetworkInterface`) que vous pouvez utiliser lorsque vous créez un rôle. Vous pouvez examiner cette stratégie dans la console IAM. Ne supprimez pas ce rôle immédiatement après l'exécution de votre fonction Lambda. Il y a un décalage entre l'exécution de votre fonction Lambda et la suppression de l'ENI. Si vous supprimez le rôle immédiatement après l'exécution de la fonction, vous devez vous charger de la suppression des ENI.

- La commande CLI `update-function-configuration` spécifie le paramètre `--vpc-config` pour ajouter les informations du VPC à une configuration de fonction Lambda existante.

```
$ aws lambda update-function-configuration \
--function-name ExampleFunction \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=security-group-ids
```

Pour supprimer les informations liées au VPC de la configuration de la fonction Lambda, utilisez l'API `UpdateFunctionConfiguration` en fournissant une liste vide d'ID de sous-réseau et d'ID de groupes de sécurité, comme illustré dans l'exemple suivant de commande CLI.

```
$ aws lambda update-function-configuration \
--function-name ExampleFunction \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

Notez les considérations supplémentaires suivantes :

- Lorsque vous ajoutez la configuration du VPC à une fonction Lambda, elle ne peut accéder qu'aux ressources de ce VPC. Si une fonction Lambda a besoin d'accéder aux ressources du VPC et au réseau Internet public, le VPC doit disposer d'une instance de traduction d'adresses réseau (NAT).
- Lorsqu'une fonction Lambda est configurée pour s'exécuter au sein d'un VPC, elle subit une perte de performances au démarrage des ENI. Autrement dit, la résolution d'adresse peut être retardée lors de la tentative de connexion aux ressources réseau.

Accès Internet pour les fonctions Lambda

AWS Lambda utilise les informations de VPC que vous fournissez pour configurer les ENI permettant à la fonction Lambda d'accéder aux ressources VPC. Chaque ENI se voit attribuer une adresse IP privée provenant de la plage d'adresses IP dans les sous-réseaux que vous spécifiez, mais ne reçoit aucune adresse IP publique. Par conséquent, si la fonction Lambda nécessite un accès Internet (notamment pour

accéder aux services AWS qui ne disposent pas de points de terminaison de VPC), vous pouvez configurer une instance NAT dans le VPC ou utiliser une passerelle NAT Amazon VPC. Pour plus d'informations, consultez [Passerelles NAT](#) dans le Amazon VPC Guide de l'utilisateur. Vous ne pouvez pas utiliser une passerelle Internet reliée à votre VPC, car l'interface réseau Elastic nécessiterait des adresses IP publiques.

Important

Si la fonction Lambda a besoin d'un accès Internet, ne l'associez pas à un sous-réseau public ou à un sous-réseau privé sans accès Internet. Au lieu de cela, reliez-la uniquement à des sous-réseaux privés avec accès Internet via une instance NAT ou une passerelle Amazon VPC NAT.

Consignes pour la configuration de fonctions Lambda basées sur un VPC

La fonction Lambda s'adapte automatiquement en fonction du nombre d'événements qu'elle traite. Voici des consignes générales pour la configuration de fonctions Lambda basées sur un VPC afin de permettre cette adaptabilité.

- Si la fonction Lambda accède à un VPC, vous devez vous assurer que ce dernier dispose d'une capacité ENI suffisante pour répondre aux besoins évolutifs de cette fonction. Vous pouvez utiliser la formule suivante pour déterminer les exigences approximatives de l'interface réseau Elastic.

Projected peak concurrent executions * (Memory in GB / 3GB)

Où :

- Projected peak concurrent execution : utilisez les informations de la section [Gestion de la simultanéité \(p. 279\)](#) pour déterminer cette valeur.
- Memory : quantité de mémoire que vous avez configurée pour la fonction Lambda.
- Les sous-réseaux que vous spécifiez doivent avoir suffisamment d'adresses IP disponibles pour correspondre au nombre d'ENI.

Nous vous recommandons également de spécifier au moins un sous-réseau dans chaque zone de disponibilité de la configuration de la fonction Lambda. En spécifiant les sous-réseaux dans chaque zone de disponibilité, la fonction Lambda peut fonctionner dans une autre zone si l'une d'elles tombe en panne ou n'a plus d'adresses IP.

Si le VPC ne dispose pas de suffisamment d'ENI ou d'adresses IP de sous-réseau, la fonction Lambda ne s'adapte pas à l'augmentation des requêtes, et vous observerez davantage d'erreurs d'appels (`EC2ThrottledException`). Pour l'appel asynchrone, si vous voyez une augmentation des erreurs sans CloudWatch Logs correspondant,appelez la fonction Lambda de façon synchrone dans la console pour obtenir les réponses d'erreurs.

Didacticiel : Configuration d'une fonction Lambda pour accéder à Amazon ElastiCache dans un Amazon VPC

Dans ce didacticiel, vous effectuez les opérations suivantes :

- Créez un cluster Amazon ElastiCache dans votre Amazon Virtual Private Cloud par défaut. Pour plus d'informations sur Amazon ElastiCache, consultez [Amazon ElastiCache](#).
- Créez une fonction Lambda pour accéder au cluster ElastiCache. Lorsque vous créez la fonction Lambda, vous fournissez des ID de sous-réseau dans votre Amazon VPC, ainsi qu'un groupe de sécurité

pour permettre à la fonction Lambda d'accéder aux ressources de votre VPC. Pour illustrer ce propos dans ce didacticiel, la fonction Lambda génère un UUID, l'écrit dans le cache et l'extrait du cache.

- Appelez la fonction Lambda et vérifiez qu'elle a accédé au cluster ElastiCache de votre VPC.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Autorisations – AWSLambdaVPCAccessExecutionRole.
 - Nom de rôle – **lambda-vpc-role**.

Le rôle AWSLambdaVPCAccessExecutionRole possède les autorisations dont la fonction a besoin pour gérer les connexions réseau à un VPC.

Créez un cluster ElastiCache.

Créez un cluster ElastiCache dans votre VPC par défaut.

1. Exécutez la commande d'AWS CLI suivante pour créer un cluster Memcached.

```
$ aws elasticache create-cache-cluster --cache-cluster-id ClusterForLambdaTest \  
--cache-node-type cache.m3.medium --engine memcached --num-cache-nodes 1 \  
--security-group-ids your-default-vpc-security-group
```

Vous pouvez rechercher le groupe de sécurité par défaut du VPC dans la console VPC, sous Security Groups. L'exemple de fonction Lambda ajoute et récupère un élément dans ce cluster.

2. Notez le point de terminaison de configuration du cluster de cache que vous avez lancé. Cette information est disponible dans la console Amazon ElastiCache. Vous spécifierez cette valeur dans le code de la fonction Lambda dans la section suivante.

Créer un package de déploiement

L'exemple suivant de code Python lit et écrit un élément dans le cluster ElastiCache.

Example app.py

```
from __future__ import print_function
import time
import uuid
import sys
import socket
import elasticache_auto_discovery
from pymemcache.client.hash import HashClient

#elasticache settings
elasticache_config_endpoint = "your-elasticsearch-cluster-endpoint:port"
nodes = elasticache_auto_discovery.discover(elasticache_config_endpoint)
nodes = map(lambda x: (x[1], int(x[2])), nodes)
memcache_client = HashClient(nodes)

def handler(event, context):
    """
    This function puts into memcache and get from it.
    Memcache is hosted using elasticache
    """

    #Create a random UUID... this will be the sample element we add to the cache.
    uid_inserted = str(uuid.uuid4().hex)
    #Put the UUID to the cache.
    memcache_client.set('uuid', uid_inserted)
    #Get item (UUID) from the cache.
    uid_obtained = memcache_client.get('uuid')
    if uid_obtained.decode("utf-8") == uid_inserted:
        # this print should go to the CloudWatch Logs and Lambda console.
        print ("Success: Fetched value %s from memcache" %(uid_inserted))
    else:
        raise Exception("Value is not the same as we put :(. Expected %s got %s"
        %(uid_inserted, uid_obtained))

    return "Fetched value from memcache: " + uid_obtained.decode("utf-8")
```

Dépendances

- [pymemcache](#) – Le code de la fonction Lambda utilise cette bibliothèque afin de créer un objet HashClient pour définir et obtenir les éléments à partir de memcache.
- [elasticache-auto-discovery](#) – La fonction Lambda utilise cette bibliothèque pour obtenir les nœuds de votre cluster Amazon ElastiCache.

Install dependencies with Pip and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Création de la fonction Lambda

Créez la fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name AccessMemCache --timeout 30 --memory-size 1024 \
\ --zip-file fileb://function.zip --handler app.handler --runtime python3.7 \
--role execution-role-arm \
```

```
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=default-security-group-id
```

Vous trouverez les ID de sous-réseau et l'ID du groupe de sécurité par défaut de votre VPC dans la console VPC.

Test de la fonction Lambda

Au cours de cette étape, vous allez appeler la fonction Lambda manuellement à l'aide de la commande `invoke`. Lorsque la fonction Lambda est exécutée, elle génère un UUID et l'écrit dans le cluster ElastiCache que vous avez spécifié dans le code Lambda. La fonction Lambda récupère ensuite l'élément à partir du cache.

1. Appelez la fonction Lambda à l'aide de la commande `invoke`.

```
$ aws lambda invoke --function-name AccessMemCache output.txt
```

2. Vérifiez que l'exécution de la fonction Lambda a réussi, comme suit :

- Passez en revue le fichier `output.txt`.
- Examinez les résultats dans la console AWS Lambda.
- Vérifiez les résultats dans CloudWatch Logs.

Maintenant que vous avez créé une fonction Lambda qui accède à un cluster ElastiCache de votre VPC, vous pouvez faire en sorte qu'elle soit appelée en réponse à des événements. Pour obtenir plus d'informations sur la configuration des sources d'événements et pour voir des exemples, consultez la section [Cas d'utilisation \(p. 126\)](#).

Didacticiel : Configuration d'une fonction Lambda pour accéder à Amazon RDS dans un Amazon VPC

Dans ce didacticiel, vous effectuez les opérations suivantes :

- Lancez une instance de moteur de base de données Amazon RDS MySQL dans votre Amazon VPC par défaut. Dans l'instance MySQL, vous créerez une base de données (ExampleDB) contenant un exemple de table (`Employee`). Pour plus d'informations sur Amazon RDS, consultez [Amazon RDS](#).
- Créez une fonction Lambda pour accéder à la base de données ExampleDB, créer une table (`Employee`), ajouter quelques enregistrements et les extraire de la table.
- Appelez la fonction Lambda et vérifiez les résultats de la requête. Cette approche permet de confirmer que la fonction Lambda a pu accéder à l'instance RDS MySQL dans le VPC.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Autorisations – AWSLambdaVPCAccessExecutionRole.
 - Nom de rôle – **lambda-vpc-role**.

Le rôle AWSLambdaVPCAccessExecutionRole possède les autorisations dont la fonction a besoin pour gérer les connexions réseau à un VPC.

Créer une instance de base de données Amazon RDS

Dans ce didacticiel, cet exemple de fonction Lambda crée une table (`Employee`), insère quelques enregistrements, puis les extrait. La table que la fonction Lambda crée présente le schéma suivant :

```
Employee(EmpID, Name)
```

Où `EmpID` est la clé primaire. Vous devez maintenant ajouter quelques enregistrements à cette table.

Tout d'abord, lancez une instance RDS MySQL dans votre VPC par défaut avec la base de données `ExampleDB`. Si une instance RDS MySQL est déjà en cours d'exécution dans votre VPC par défaut, passez cette étape.

Vous pouvez lancer une instance RDS MySQL via l'une des méthodes suivantes :

- Suivez les instructions sous [Création d'une instance de base de données MySQL et connexion à une base de données sur une instance de base de données MySQL](#) dans le Amazon RDS Guide de l'utilisateur.
- Utilisez la commande AWS CLI suivante :

```
$ aws rds create-db-instance --db-name ExampleDB --engine MySQL \
--db-instance-identifier MySQLForLambdaTest --backup-retention-period 3 \
--db-instance-class db.t2.micro --allocated-storage 5 --no-publicly-accessible \
--master-username username --master-user-password password
```

Notez le nom de la base de données, le nom d'utilisateur et le mot de passe. Vous avez également besoin de l'adresse d'hôte (point de terminaison) de l'instance de base de données, que vous pouvez obtenir à partir de la console RDS. Vous devrez peut-être patienter jusqu'à ce que le statut de l'instance soit disponible et que la valeur du point de terminaison s'affiche dans la console.

Créer un package de déploiement

L'exemple suivant de code Python exécute une requête SELECT relative à la table Employee dans l'instance RDS MySQL que vous avez créée dans le VPC. Ce code crée une table dans la base de données ExampleDB, ajoute des exemples d'enregistrements et les extrait.

Example app.py

```
import sys
import logging
import rds_config
import pymysql
#rds settings
rds_host = "rds-instance-endpoint"
name = rds_config.db_username
password = rds_config.db_password
db_name = rds_config.db_name

logger = logging.getLogger()
logger.setLevel(logging.INFO)

try:
    conn = pymysql.connect(rds_host, user=name, passwd=password, db=db_name,
                           connect_timeout=5)
except:
    logger.error("ERROR: Unexpected error: Could not connect to MySql instance.")
    sys.exit()

logger.info("SUCCESS: Connection to RDS mysql instance succeeded")
def handler(event, context):
    """
    This function fetches content from mysql RDS instance
    """

    item_count = 0

    with conn.cursor() as cur:
        cur.execute("create table Employee3 ( EmpID int NOT NULL, Name varchar(255) NOT
NULL, PRIMARY KEY (EmpID))")
        cur.execute('insert into Employee3 (EmpID, Name) values(1, "Joe")')
        cur.execute('insert into Employee3 (EmpID, Name) values(2, "Bob")')
        cur.execute('insert into Employee3 (EmpID, Name) values(3, "Mary")')
        conn.commit()
        cur.execute("select * from Employee3")
        for row in cur:
            item_count += 1
            logger.info(row)
            #print(row)
        conn.commit()

    return "Added %d items from RDS MySQL table" %(item_count)
```

L'exécution de `pymysql.connect()` en dehors du gestionnaire permet à votre fonction de réutiliser la connexion de base de données afin d'améliorer les performances.

Un second fichier contient les informations de connexion de la fonction.

Example rds_config.py

```
#config file containing credentials for rds mysql instance
db_username = "username"
db_password = "password"
```

```
db_name = "ExampleDB"
```

Dépendances

- `pymysql` : le code de la fonction Lambda utilise cette bibliothèque pour accéder à votre instance MySQL (voir [PyMySQL](#)).

Install dependencies with Pip and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Création de la fonction Lambda

Créez la fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name CreateTableAddRecordsAndRead --runtime python3.7 \
--zip-file fileb://app.zip --handler app.handler \
--role execution-role-arn \
--vpc-config SubnetIds=comma-separated-subnet-ids,SecurityGroupIds=default-vpc-security-group-id
```

Test de la fonction Lambda

Au cours de cette étape, vous allez appeler la fonction Lambda manuellement à l'aide de la commande `invoke`. Lorsque la fonction Lambda est exécutée, elle applique la requête SELECT sur la table Employee dans l'instance RDS MySQL et affiche les résultats, lesquels sont également transmis à CloudWatch Logs.

1. Appelez la fonction Lambda à l'aide de la commande `invoke`.

```
$ aws lambda invoke --function-name CreateTableAddRecordsAndRead output.txt
```

2. Vérifiez que l'exécution de la fonction Lambda a réussi, comme suit :

- Passez en revue le fichier `output.txt`.
- Examinez les résultats dans la console AWS Lambda.
- Vérifiez les résultats dans CloudWatch Logs.

Maintenant que vous avez créé une fonction Lambda qui accède à une base de données dans votre VPC, vous pouvez faire en sorte que la fonction soit appelée en réponse à des événements. Pour obtenir plus d'informations sur la configuration des sources d'événements et pour voir des exemples, consultez la section [Cas d'utilisation \(p. 126\)](#).

Appel de fonctions AWS Lambda

Lors de la création d'applications sur AWS Lambda, dont les applications [sans serveur](#), les composants de base sont les fonctions Lambda et les sources d'événements. Une source d'événement est le service AWS ou l'application personnalisée qui publie des événements, tandis qu'une fonction Lambda correspond au code personnalisé qui traite les événements. Pour bien comprendre cette distinction, prenez en compte les scénarios suivants :

- Traitement des fichiers – Supposons que votre application permette de partager des photos. Les personnes l'utilisent pour importer des photos que l'application stocke dans un compartiment Amazon S3. L'application crée ensuite une miniature des photos de chaque utilisateur et les affiche sur leur page de profil. Dans ce scénario, vous pouvez choisir de créer une fonction Lambda qui génère une miniature automatiquement. Amazon S3 est l'une des sources d'événements AWS prises en charge. Elle permet de publier les événements générés par des objets et d'appeler votre fonction Lambda. Le code de la fonction Lambda permet de lire l'objet photo dans le compartiment S3, d'en créer une version miniature et de l'enregistrer dans un autre compartiment S3.
- Données et analyses – Supposons que vous conceviez une application d'analyse et que vous stockiez les données brutes dans une table DynamoDB. Lorsque vous écrivez, mettez à jour ou supprimez des éléments dans une table, les flux DynamoDB peuvent publier des événements de mise à jour de ces éléments dans un flux associé à cette table. Dans ce cas, les données d'événement fournissent la clé de l'élément, le nom de l'événement (insertion, mise à jour ou suppression, par exemple) et tout autre détail pertinent. Vous pouvez écrire une fonction Lambda de sorte à regrouper les données brutes afin de générer des métriques personnalisées.
- Sites web – Supposons que vous créez un site web et que vous souhaitez héberger la logique backend dans Lambda. Vous pouvez appeler votre fonction Lambda via HTTP avec Amazon API Gateway comme point de terminaison HTTP. Désormais, le client web peut appeler l'API, puis API Gateway peut acheminer la demande vers Lambda.
- Applications mobiles – Supposons que vous ayez une application mobile personnalisée qui génère des événements. Vous pouvez créer une fonction Lambda pour traiter les événements publiés par votre application personnalisée. Par exemple, dans ce scénario, vous pouvez configurer une fonction Lambda pour traiter les clics au sein de votre application mobile personnalisée.

Chacune de ces sources utilise un format spécifique pour les données d'événement. Pour plus d'informations, consultez [Exemples d'événements publiés en fonction des sources \(p. 91\)](#). Quand une fonction Lambda est appelée, elle reçoit l'événement comme paramètre pour la fonction Lambda.

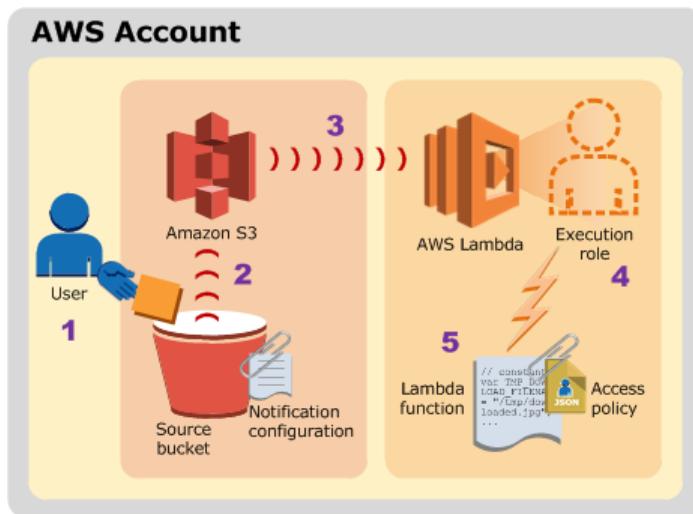
AWS Lambda prend en charge de nombreux services AWS en tant que sources d'événements. Pour plus d'informations, consultez [Sources d'événements prises en charge \(p. 82\)](#). Lorsque vous configurez ces sources d'événements pour déclencher une fonction Lambda, cette dernière est appelée automatiquement lorsque des événements se produisent. Vous définissez un mappage des sources d'événements, qui détermine comment identifier les événements à suivre et la fonction Lambda à appeler.

Outre les services AWS pris en charge, les applications utilisateurs peuvent aussi générer des événements : vous pouvez créer vos propres sources d'événements personnalisées. Les sources d'événements personnalisées appellent une fonction Lambda à l'aide de l'opération AWS Lambda [Invoke \(p. 452\)](#). Les applications utilisateurs, telles que les applications client, mobiles ou web, peuvent publier des événements et appeler les fonctions Lambda à la demande via les kits SDK AWS ou AWS Mobile (AWS Mobile SDK pour Android, par exemple).

Voici des exemples introductifs de sources d'événements et de fonctionnement de l'expérience complète.

Exemple 1 : Amazon S3 transmet les événements et appelle une fonction Lambda

Amazon S3 peut publier des événements de différents types, tels que les événements d'objets PUT, POST, COPY et DELETE au niveau d'un compartiment. Avec la fonctionnalité des notifications de compartiment, vous pouvez configurer un mappage de source d'événement qui indique à Amazon S3 d'appeler une fonction Lambda lorsqu'un type spécifique d'événement se produit, comme illustré ci-après.



Ce diagramme illustre la séquence suivante :

1. L'utilisateur crée un objet dans un compartiment.
2. Amazon S3 détecte l'événement de création d'objet.
3. Amazon S3 appelle votre fonction Lambda en utilisant les autorisations fournies par le rôle d'exécution.
Pour plus d'informations sur les rôles d'exécution, consultez [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#). Amazon S3 sait quelle fonction Lambda appeler grâce au mappage de source d'événement qui est stocké dans la configuration des notifications de compartiment.
4. AWS Lambda exécute la fonction Lambda en spécifiant l'événement comme paramètre.

Notez bien ce qui suit :

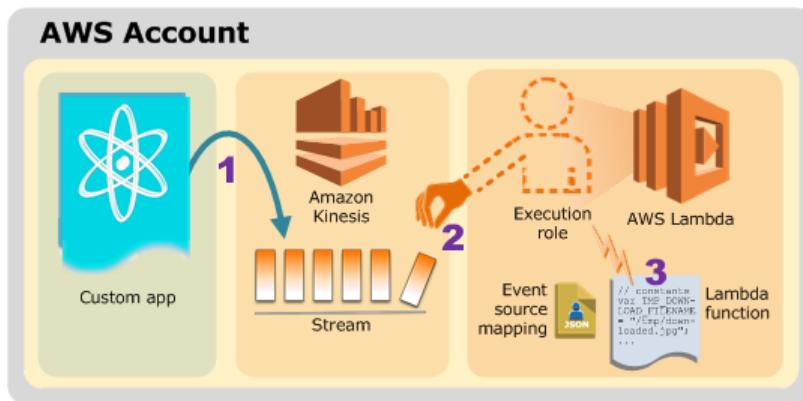
- Le mappage de source d'événement est géré dans le service correspond, Amazon S3 dans ce scénario. C'est le cas pour toutes les sources d'événements AWS prises en charge, sauf celles qui sont basées sur les interrogations (flux Kinesis et DynamoDB ou files d'attente Amazon SQS). L'exemple suivant explique les sources d'événements basées sur les interrogations.
- La source d'événement (Amazon S3) appelle la fonction Lambda (dénommée modèle push). Là aussi, c'est le cas pour tous les services AWS pris en charge, sauf les sources d'événements basées sur les interrogations.
- Pour que la source d'événement (Amazon S3) appelle la fonction Lambda, vous devez accorder des autorisations à l'aide de la stratégie correspondante associée à la fonction Lambda.

Exemple 2 : AWS Lambda extrait les événements d'un flux Kinesis et appelle une fonction Lambda

Pour les sources d'événements basées sur les interrogations, AWS Lambda interroge la source puis appelle la fonction Lambda lorsque des enregistrements sont détectés dans cette source. Pour les sources basées sur les interrogations, les informations de mappage d'une source d'événement sont stockées dans AWS Lambda. AWS Lambda fournit une API qui vous permet de créer et de gérer ces mappages de source d'événement.

- [CreateEventSourceMapping \(p. 406\)](#)
- [UpdateEventSourceMapping \(p. 504\)](#)

Le schéma suivant montre comment une application personnalisée écrit des enregistrements dans un flux Kinesis.



Ce diagramme illustre la séquence suivante :

1. L'application personnalisée écrit les enregistrements dans un flux Kinesis.
2. AWS Lambda interroge continuellement le flux et appelle la fonction Lambda lorsque le service détecte de nouveaux enregistrements dans le flux. AWS Lambda sait quel flux interroger et quelle fonction Lambda appeler en fonction du mappage de source d'événement que vous créez dans Lambda.
3. La fonction Lambda est appelée avec l'événement entrant.

Notez bien ce qui suit :

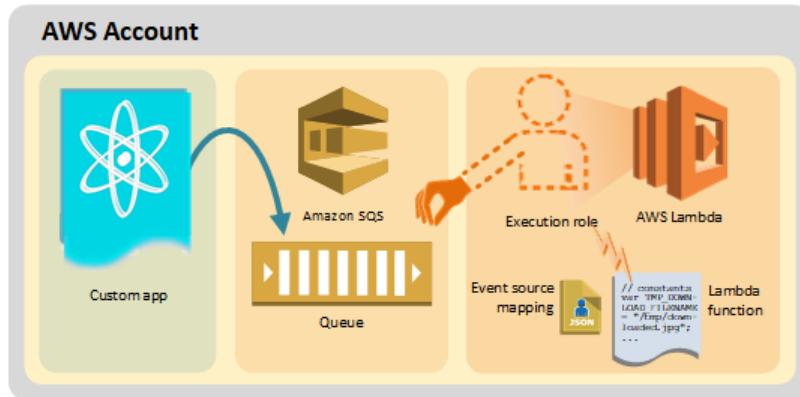
- Lorsque vous utilisez les sources d'événements basées sur les flux, les affirmations suivantes s'appliquent :
 - Vous créez des mappages de source d'événement dans AWS Lambda.
 - AWS Lambda appelle la fonction Lambda de façon synchrone (dénommée modèle pull).
- AWS Lambda ne nécessite pas d'autorisation pour appeler votre fonction Lambda. Par conséquent, vous n'avez pas besoin d'ajouter d'autorisations à la stratégie d'autorisations associée à votre fonction Lambda.
- Le rôle Lambda doit disposer des autorisations requises pour accéder au flux en lecture.

Exemple 3 : AWS Lambda extrait les événements d'une file d'attente Amazon SQS et appelle une fonction Lambda

Pour les sources d'événements basées sur les interrogations, AWS Lambda interroge la source puis appelle la fonction Lambda lorsque des enregistrements sont détectés dans cette source. Pour les sources basées sur les interrogations, les informations de mappage d'une source d'événement sont stockées dans AWS Lambda. AWS Lambda fournit une API qui vous permet de créer et de gérer ces mappages de source d'événement.

- [CreateEventSourceMapping \(p. 406\)](#)
- [UpdateEventSourceMapping \(p. 504\)](#)

Le schéma suivant montre comment une application personnalisée écrit des enregistrements dans une file d'attente Amazon SQS :



Ce diagramme illustre la séquence suivante :

1. L'application personnalisée écrit les enregistrements dans une file d'attente Amazon SQS.
2. AWS Lambda interroge la file d'attente en permanence et appelle la fonction Lambda lorsque le service détecte de nouveaux enregistrements. AWS Lambda sait quelle file d'attente interroger et quelle fonction Lambda appeler en fonction du mappage de source d'événement que vous créez dans Lambda.
3. La fonction Lambda est appelée avec l'événement entrant.

Notez bien ce qui suit :

- Lorsque vous utilisez des événements basés sur les interrogations, les affirmations suivantes s'appliquent :
 - Vous créez des mappages de source d'événement dans AWS Lambda.
 - AWS Lambda appelle la fonction Lambda de façon synchrone (dénommée modèle pull).
- AWS Lambda ne nécessite pas d'autorisation pour appeler votre fonction Lambda. Par conséquent, vous n'avez pas besoin d'ajouter d'autorisations à la stratégie d'autorisations associée à votre fonction Lambda.
- Votre rôle Lambda a besoin d'une autorisation à lire depuis la file d'attente et ses attributs.

Types d'appel

AWS Lambda prend en charge l'appel synchrone et asynchrone d'une fonction Lambda. Vous pouvez contrôler le type d'appel uniquement lorsque vous appelez une fonction Lambda (appel à la demande). Les exemples suivants illustrent des appels à la demande :

- Votre application personnalisée appelle une fonction Lambda
- Vousappelez manuellement une fonction Lambda (par exemple, à l'aide d'AWS CLI) à des fins de test.

Dans les deux cas, vousappelez la fonction Lambda à l'aide de l'opération [Invoke \(p. 452\)](#) et vous pouvez spécifier le type d'appel comme étant synchrone ou asynchrone.

Lorsque vousutilisez un service AWS comme déclencheur, le type d'appel est prédéfini pour chaque service. Vousn'avez aucun contrôle sur le type d'appel que ces sources d'événements utilisent lorsqu'elles appellent votre fonction Lambda.

Par exemple, Amazon S3 appelle toujours une fonction Lambda de manière asynchrone, et Amazon Cognito toujours de manière synchrone. Pour les services AWS basés sur des interrogations (Amazon Kinesis, Amazon DynamoDB, Amazon Simple Queue Service), AWS Lambda interroge le flux ou la file d'attente de messages et appelle votre fonction Lambda de manière synchrone.

Mappage de source d'événement AWS Lambda

Les fonctions Lambda et les sources d'événements sont les composants clés d'AWS Lambda. Une source d'événement est l'entité qui publie des événements, tandis qu'une fonction Lambda est le code personnalisé qui traite ces événements. Les sources d'événements prises en charge désignent les services AWS qui peuvent être préconfigurés pour fonctionner avec AWS Lambda. Cette configuration porte le nom de mappage de source d'événement, qui associe une source d'événement à une fonction Lambda. Elle permet un appel automatique de la fonction Lambda lorsque des événements se produisent.

Chaque mappage de source d'événement identifie le type d'événement à publier et la fonction Lambda à appeler en cas d'événement. La fonction Lambda spécifique reçoit ensuite les informations d'événement comme paramètre, et le code de la fonction Lambda peut alors traiter l'événement.

Vous pouvez également créer des applications personnalisées pour inclure les événements de ressources AWS et appeler une fonction Lambda. Pour en savoir plus, veuillez consulter [Utilisation d'AWS Lambda avec l'AWS Command Line Interface \(p. 217\)](#)

Vous vous posez peut-être des questions. Où vaut-il mieux conserver les informations de mappage d'événement ? Au sein de la source d'événement ou dans AWS Lambda ? Les sections suivantes expliquent le mappage de source d'événement pour chaque catégorie de source d'événement. Elles décrivent également comment la fonction Lambda est appelée et comment gérer les autorisations associées.

Rubriques

- [Mappage de source d'événement pour les services AWS \(p. 75\)](#)
- [Mappage de source d'événement pour les services basés sur les interrogations \(p. 76\)](#)
- [Mappage de source d'événement pour les applications personnalisées \(p. 77\)](#)

Mappage de source d'événement pour les services AWS

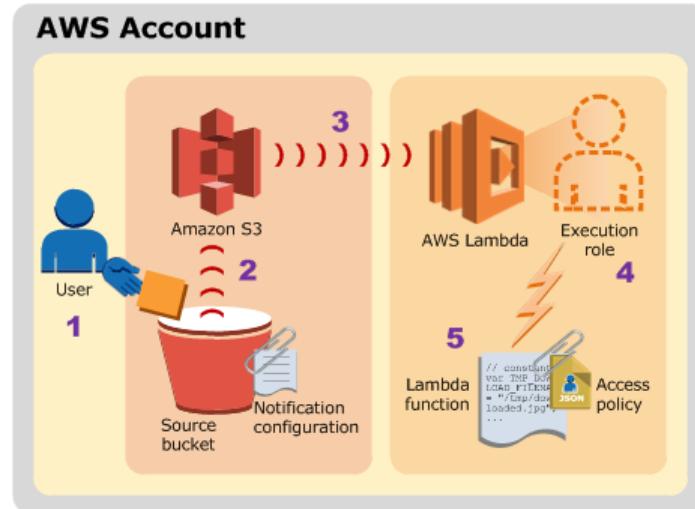
À l'exception des services AWS basés sur les interrogations (Amazon Kinesis Data Streams, flux DynamoDB et files d'attente Amazon SQS), les autres services AWS pris en charge publient des événements et peuvent également appeler la fonction Lambda (modèle push). Dans le modèle push, notez les éléments suivants :

- Les mappages d'événement source sont gérés au sein de la source d'événement. Des APIs spécifiques dans les sources d'événement vous permettent de créer et de gérer les mappages de source d'événement. Par exemple, Amazon S3 fournit l'API de configuration des notifications de compartiment. Celle-ci vous permet de configurer un mappage de source d'événement qui identifie les événements de compartiment à publier et la fonction Lambda à appeler.
- Étant donné que les sources d'événement appellent votre fonction Lambda, vous devez leur accorder les autorisations nécessaires à l'aide d'une stratégie basée sur les ressources (stratégie de la fonction Lambda). Pour en savoir plus, consultez [Modèle d'autorisations AWS Lambda \(p. 260\)](#).

Les exemples suivants illustrent le principe de fonctionnement de ce modèle.

Example – Amazon S3 transmet les événements et appelle une fonction Lambda

Supposons que vous souhaitez appeler la fonction AWS Lambda pour chaque événement de création d'objet dans le compartiment. Vous ajoutez le mappage de source d'événement nécessaire dans la configuration des notifications du compartiment.



Le diagramme suivant illustre le processus :

1. L'utilisateur crée un objet dans un compartiment.
2. Amazon S3 détecte l'événement de création d'objet.
3. Amazon S3 appelle la fonction Lambda selon le mappage de source d'événement décrit dans la configuration des notifications du compartiment.
4. AWS Lambda vérifie la stratégie d'autorisations associée à la fonction Lambda afin de garantir qu'Amazon S3 dispose des autorisations nécessaires. Pour plus d'informations sur les stratégies d'autorisations, reportez-vous à la section [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).
5. Une fois qu'AWS Lambda a vérifié la stratégie d'autorisations associée, il exécute la fonction Lambda. N'oubliez pas que la fonction Lambda reçoit l'événement comme paramètre.

Mappage de source d'événement pour les services basés sur les interrogations

AWS Lambda prend en charge les services basés sur les interrogations suivants :

- Amazon Kinesis
- Amazon DynamoDB
- Amazon SQS

Une fois que vous avez configuré le [CreateEventSourceMapping \(p. 406\)](#) nécessaire, AWS Lambda interroge la source d'événement et appelle la fonction Lambda (modèle pull). Dans le modèle pull, notez les éléments suivants :

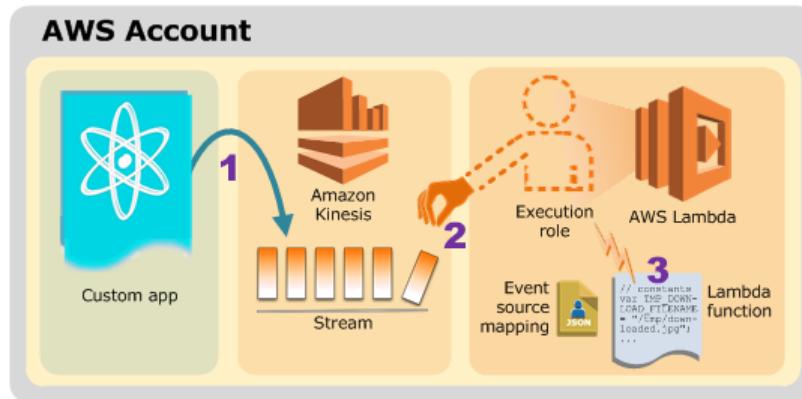
- Les mappages de source d'événement sont conservés dans AWS Lambda. AWS Lambda fournit les API pertinentes pour créer et gérer les mappages de source d'événement. Pour plus d'informations, consultez [CreateEventSourceMapping \(p. 406\)](#).
- AWS Lambda nécessite votre autorisation pour interroger les flux Kinesis et DynamoDB ou les files d'attente Amazon SQS, et lire les enregistrements. Vous accordez ces autorisations via le rôle d'exécution, à l'aide de la stratégie d'autorisations associée au rôle que vous spécifiez lorsque vous créez la fonction Lambda. AWS Lambda ne nécessite pas d'autorisations pour appeler la fonction Lambda.

Les exemples suivants illustrent le principe de fonctionnement de ce modèle.

Example – AWS Lambda extrait les événements d'un flux Amazon Kinesis et appelle une fonction Lambda

Le schéma suivant illustre une application personnalisée qui écrit des enregistrements dans un flux Kinesis et décrit comment AWS Lambda interroge le flux. Quand AWS Lambda détecte un nouvel enregistrement au niveau du flux, il appelle la fonction Lambda.

Supposons que votre application personnalisée écrive des enregistrements dans un flux Kinesis. Vous voulez appeler une fonction Lambda lorsque de nouveaux enregistrements y sont détectés. Vous créez une fonction Lambda et le mappage de source d'événement nécessaire dans AWS Lambda.



Ce diagramme illustre la séquence suivante :

1. L'application personnalisée écrit les enregistrements dans un flux Amazon Kinesis.
2. AWS Lambda interroge continuellement le flux et appelle la fonction Lambda lorsque le service y détecte de nouveaux enregistrements. AWS Lambda sait quel flux interroger et quelle fonction Lambda appeler en fonction du mappage de source d'événement créé dans AWS Lambda.

3. En supposant que la stratégie d'autorisation associée, qui permet à AWS Lambda d'interroger le flux, est vérifiée, AWS Lambda exécute la fonction Lambda. Pour plus d'informations sur les stratégies d'autorisations, reportez-vous à la section [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

Cet exemple utilise un flux Kinesis, mais le même processus s'applique lorsqu'il s'agit d'un flux DynamoDB.

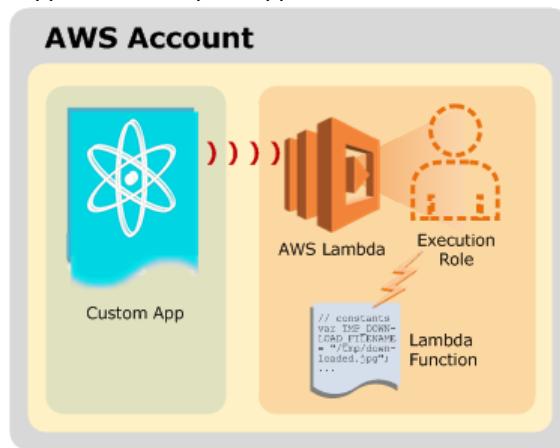
Mappage de source d'événement pour les applications personnalisées

Si vos applications personnalisées publient et gèrent des événements, vous pouvez créer une fonction Lambda pour traiter ces derniers. Dans ce cas, aucune pré-configuration n'est requise : vous n'avez pas besoin de définir un mappage de source d'événement. Au lieu de cela, la source d'événement utilise l'API `Invoke` d'AWS Lambda. Si l'application et la fonction Lambda appartiennent à des comptes AWS distincts, le compte AWS qui possède la fonction Lambda doit activer des autorisations entre comptes dans la stratégie d'autorisations associée à la fonction Lambda.

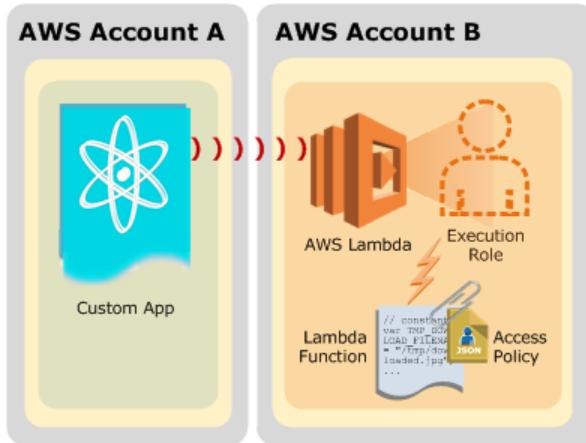
L'exemple suivant illustre comment cela fonctionne.

Example – L'application personnalisée publie des événements et appelle une fonction Lambda

Le schéma suivant montre comment une application personnalisée de votre compte peut appeler une fonction Lambda. Dans cet exemple, comme l'application personnalisée utilise les informations d'identification du compte qui possède la fonction Lambda, elle ne requiert pas d'autorisations supplémentaires pour appeler cette dernière.



Dans l'exemple suivant, l'application utilisateur et la fonction Lambda appartiennent à des comptes AWS distincts. Dans ce cas, le compte AWS qui possède la fonction Lambda doit avoir des autorisations entre comptes dans la stratégie d'autorisations associée à la fonction Lambda. Pour en savoir plus, consultez [Modèle d'autorisations AWS Lambda \(p. 260\)](#).



Comportement de nouvelle tentative d'AWS Lambda

Une fonction Lambda peut échouer pour l'une des raisons suivantes :

- La fonction expire alors qu'elle essaie de se connecter à un point de terminaison.
- La fonction ne parvient pas à analyser correctement les données d'entrée.
- La fonction fait l'objet de contraintes de ressources, telles que des erreurs de saturation de mémoire ou de dépassement de délais d'expiration.

Si ces erreurs se produisent, la fonction génère une exception. Le traitement de cette exception dépend du type d'appel de la fonction Lambda :

- Sources d'événements non basées sur les flux : certaines de ces sources d'événements sont configurées pour appeler une fonction Lambda de façon synchrone, et d'autres de façon asynchrone. Par conséquent, les exceptions sont gérées comme suit :
 - Appel synchrone : l'application qui est à l'origine de l'appel reçoit l'erreur 429 et est chargée de réessayer. Pour obtenir la liste des sources d'événements prises en charge et des types d'appel qu'elles utilisent, consultez [Sources d'événements prises en charge](#). L'intégration de ces sources d'événements peut inclure des tentatives supplémentaires.

Si vous avez appelé la fonction Lambda directement via des kits de développement logiciel (SDK) AWS, le client reçoit l'erreur et peut choisir de réessayer.

- Appel asynchrone : les événements asynchrones sont ajoutés à une file d'attente avant d'être utilisés pour appeler la fonction Lambda. Si AWS Lambda ne parvient pas à traiter entièrement l'événement, il retentera automatiquement l'appel deux fois, avec des temps d'attente entre les tentatives. Si vous avez spécifié une file d'attente de lettres mortes pour votre fonction, l'événement en échec sera envoyé à la file d'attente Amazon SQS ou à la rubrique Amazon SNS spécifiée. Si vous n'avez pas spécifié

une file d'attente de lettres mortes, laquelle n'est pas obligatoire et constitue le paramètre par défaut, l'événement sera ignoré. Pour en savoir plus, consultez [Files d'attente Lettre Morte \(p. 81\)](#).

- Sources d'événements basées sur les interrogations (modèle pull) qui sont basées sur les flux : elles comprennent Kinesis Data Streams ou DynamoDB. Lorsque l'appel d'une fonction Lambda échoue, AWS Lambda tente de traiter le lot d'enregistrements correspondant tant que les données n'expirent pas, ce qui peut aller jusqu'à sept jours.

L'exception est considérée comme un blocage. Dès lors, AWS Lambda cesse de lire les enregistrements à partir de cette partition jusqu'à ce que le lot d'enregistrements restreint aboutisse ou expire. Cette approche garantit qu'AWS Lambda traite les événements de flux dans l'ordre.

- Sources d'événements basées sur les interrogations qui ne sont pas basées sur les flux : elles comprennent Amazon Simple Queue Service. Si vous configurez une file d'attente Amazon SQS en tant que source d'événement, AWS Lambda interroge un lot d'enregistrements dans la file d'attente et appelle votre fonction Lambda. Si l'appel échoue ou expire, chaque message du lot est renvoyé à la file d'attente et sera disponible pour le traitement une fois que le [délai de visibilité](#) aura expiré. (Un délai de visibilité est une période au cours de laquelle Amazon Simple Queue Service empêche les autres consommateurs de recevoir et de traiter le message.)

Lorsqu'un appel traite avec succès un lot, chaque message figurant dans ce lot est supprimé de la file d'attente. Lorsqu'un message n'est pas traité correctement, il est ignoré ou, si vous avez configuré une [file d'attente de lettres mortes Amazon SQS](#), les informations d'échec y sont dirigées pour que vous les analysez.

Si vous n'avez pas besoin d'un ordre de traitement d'événements, l'avantage de l'utilisation des files d'attente Amazon SQS est que AWS Lambda continuera à traiter les nouveaux messages, même si l'appel d'un message précédent échoue. En d'autres termes, le traitement de nouveaux messages ne sera pas bloqué.

Pour plus d'informations sur les modes d'appel, consultez la section [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

Présentation du comportement de dimensionnement

Les exécutions simultanées désignent le nombre d'exécutions du code de la fonction qui se produisent à un moment donné. Vous pouvez estimer le nombre d'exécutions simultanées, mais celui-ci varie selon que votre fonction Lambda traite ou non des événements à partir d'une source d'événements basée sur les interrogations.

Si vous créez une fonction Lambda pour traiter des événements issus de sources d'événements qui ne sont pas basées sur les flux (par exemple, Lambda peut traiter chaque événement issu d'autres sources, comme Amazon S3 ou API Gateway), chaque événement publié représente une unité de travail, en parallèle, dans les limites de votre compte. Par conséquent, le nombre d'événements (ou de requêtes) que ces sources d'événements publient influence la simultanéité. Vous pouvez utiliser cette formule pour estimer le nombre d'appels simultanés de la fonction Lambda :

```
events (or requests) per second * function duration
```

Par exemple, imaginons une fonction Lambda qui traite des événements Amazon S3. Supposons que cette fonction prenne en moyenne trois secondes et qu'Amazon S3 publie 10 événements par seconde. Dans ce cas, le nombre d'exécutions simultanées de la fonction Lambda s'élève à 30.

Le nombre d'exécutions simultanées pour les sources d'événements basées sur les interrogations dépend également d'autres facteurs, comme indiqué ci-après :

- Sources d'événements basées sur les interrogations qui sont basées sur les flux
 - Amazon Kinesis Data Streams
 - Amazon DynamoDB

Pour les fonctions Lambda qui traitent des flux Kinesis ou DynamoDB, le nombre de partitions est l'unité de simultanéité. Si le flux contient 100 partitions actives, il y aura au plus 100 appels de fonctions Lambda s'exécutant ensemble. Ceci s'explique du fait que Lambda traite les événements de chaque partition par ordre.

- Sources d'événements basées sur les interrogations qui ne sont pas basées sur les flux : pour les fonctions Lambda qui traitent les files d'attente Amazon SQS, AWS Lambda redimensionne automatiquement l'interrogation de la file d'attente jusqu'à ce que le niveau de simultanéité maximum soit atteint, où chaque lot de messages peut être considéré comme une seule unité simultanée. Le comportement de redimensionnement automatique d'AWS Lambda est conçu pour conserver les coûts d'interrogation au plus bas lorsqu'une file d'attente est vide, tout en vous permettant d'atteindre un débit élevé lorsque la file d'attente est grandement sollicitée.

Lorsqu'un mappage de source d'événement Amazon SQS est initialement activé, Lambda commence l'interrogation longue de la file d'attente Amazon SQS. L'interrogation longue permet de réduire le coût de l'interrogation de Amazon Simple Queue Service en réduisant le nombre de réponses vides, tout en fournissant une latence de traitement optimale lorsque les messages arrivent.

Lorsque l'afflux de messages dans une file d'attente augmente, AWS Lambda met automatiquement à l'échelle l'activité d'interrogation jusqu'à ce que le nombre d'exécutions de fonctions simultanées atteigne 1 000, la limite de simultanéité du compte ou la limite de simultanéité de la fonction (facultative), selon la valeur la plus basse. Amazon Simple Queue Service prend en charge une première rafale de 5 appels de fonction simultanés et augmente la simultanéité de 60 appels simultanés par minute.

Note

Les [limites au niveau du compte](#) sont affectées par d'autres fonctions dans le compte, et la simultanéité par fonction s'applique à tous les événements envoyés à une fonction. Pour plus d'informations, consultez [Gestion de la simultanéité \(p. 279\)](#).

Débit de demandes

Le débit de demandes désigne la fréquence à laquelle la fonction Lambda est appelée. Pour tous les services non basés sur le flux, il s'agit de la vitesse à laquelle les sources d'événements génèrent les événements. Pour les services basés sur le flux, AWS Lambda calcule le taux de demandes comme suit :

```
request rate = number of concurrent executions / function duration
```

Par exemple, si cinq partitions sont actives dans un flux (autrement dit, si cinq fonctions Lambda sont exécutées en parallèle) et que la fonction Lambda dure environ deux secondes, le débit est de 2,5 requêtes par seconde.

Dimensionnement automatique

AWS Lambda dimensionne dynamiquement l'exécution de la fonction en réponse à une augmentation du trafic, jusqu'à votre [limite de simultanéité \(p. 288\)](#). Sous charge constante, la simultanéité de votre fonction augmente brusquement à un niveau initial situé entre 500 et 3 000 exécutions simultanées qui varie en fonction de la région. Après l'augmentation initiale, la capacité de la fonction augmente de 500 exécutions simultanées supplémentaires chaque minute jusqu'à ce que la charge soit gérée, ou que la simultanéité totale de toutes les fonctions de la région atteigne la limite.

Région	Augmentation initiale de la simultanéité
USA Est (Ohio), USA Ouest (Californie du Nord), Canada (Centre)	500
USA Ouest (Oregon), USA Est (Virginie du Nord)	3000
Asie-Pacifique (Séoul), Asie-Pacifique (Mumbai), Asie-Pacifique (Singapour), Asie-Pacifique (Sydney)	500
Asie-Pacifique (Tokyo)	1 000
UE (Londres), UE (Paris)	500
UE (Francfort)	1 000
UE (Irlande)	3000
Amérique du Sud (São Paulo)	500
Chine (Pékin), Chine (Ningxia)	500
AWS GovCloud (US-West)	500

Note

Si votre fonction est connectée à un VPC, la [limite d'interface réseau d'Amazon VPC](#) peut empêcher la mise à l'échelle. Pour en savoir plus, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#).

Pour limiter la mise à l'échelle, vous pouvez configurer des fonctions avec une simultanéité réservée. Pour en savoir plus, consultez [Gestion de la simultanéité \(p. 279\)](#).

Files d'attente Lettre Morte

Toute fonction Lambda appelée de manière asynchrone est retentée deux fois, à la suite de quoi l'événement est ignoré. Si les tentatives échouent et que vous demandez pourquoi, utilisez des files d'attente de lettres mortes pour diriger les événements non traités vers une file d'attente Amazon SQS ou une rubrique Amazon SNS afin d'analyser la défaillance.

AWS Lambda dirige les événements qui ne peuvent pas être traités vers la [rubrique Amazon SNS](#) ou la [file d'attente Amazon SQS](#) spécifiée. Les fonctions qui ne spécifient pas de file d'attente de lettres mortes ignorent les événements une fois que le nombre de tentatives autorisé est épousseté. Pour plus d'informations sur les stratégies relatives aux nouvelles tentatives, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Vous configurez une file d'attente de lettres mortes en spécifiant la valeur Amazon Resource Name [TargetArn](#) pour le paramètre `DeadLetterConfig` de la fonction Lambda.

```
{
  "Code": {
    "ZipFile": blob,
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string"
  },
  "Description": "string",
  "FunctionName": "string",
  "Handler": "string",
  "MemorySize": 128,
  "Role": "arn:aws:iam::123456789012:role/LambdaRole"
}
```

```

    "Handler": "string",
    "MemorySize": number,
    "Role": "string",
    "Runtime": "string",
    "Timeout": number
    "Publish": bool,
    "DeadLetterConfig": {
        "TargetArn": "string"
    }
}

```

De plus, vous devez ajouter des autorisations pour le [rôle d'exécution \(p. 260\)](#) de la fonction Lambda, en fonction du service vers lequel vous avez dirigé les événements non traités :

- Pour Amazon SQS : [SendMessage](#)
- Pour Amazon SNS : [Publier](#)

La charge utile écrite sur l'ARN cible de la file d'attente de lettres mortes est la charge utile de l'événement d'origine, sans aucune modification apportée au corps du message. Les attributs du message contiennent des informations qui vous permettront de comprendre pourquoi l'événement n'a pas été traité :

Attributs de message de file d'attente de lettres mortes

Nom	Tapez	Value
RequestID	String	Identificateur de demande unique
ErrorCode	Numéro	Code d'erreur HTTP à 3 chiffres
ErrorMessage	String	Message d'erreur (tronqué à 1 Ko)

Les messages de file d'attente de lettres mortes n'atteignent parfois pas leur cible en raison de problèmes d'autorisations, ou si la taille totale du message dépasse la limite définie pour la file d'attente ou la rubrique cible. Par exemple, si une notification Amazon SNS avec un corps proche de 256 KB déclenche une fonction qui génère une erreur, le message peut dépasser la taille maximale autorisée dans la file d'attente de lettres mortes en raison des données d'événement supplémentaires ajoutées par Amazon SNS, combinées aux attributs ajoutés par Lambda. Lorsqu'il ne peut pas écrire dans la file d'attente de lettres mortes, Lambda supprime l'événement et émet la métrique [DeadLetterErrors \(p. 228\)](#).



Si vous utilisez Amazon SQS comme source d'événement, configurez une file d'attente de lettres mortes sur la file d'attente Amazon SQS elle-même et non pas sur la fonction Lambda. Pour plus d'informations, consultez [Utilisation de AWS Lambda avec Amazon SQS \(p. 153\)](#).

Sources d'événements prises en charge

Cette rubrique répertorie les services AWS pris en charge que vous pouvez configurer en tant que sources d'événements pour les fonctions AWS Lambda. Une fois que vous préconfigurez le mappage de

source d'événement, la fonction Lambda est appelée automatiquement lorsque ces sources détectent des événements. Pour en savoir plus sur les modes d'appel, consultez la section [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

Pour toutes les sources d'événements répertoriées dans cette rubrique, notez les éléments suivants :

- Les sources d'événements gèrent le mappage de source d'événement, sauf pour les services basés sur les interrogations (Amazon Kinesis Data Streams, Amazon DynamoDB Flux et Amazon Simple Queue Service). Pour les services basés sur les interrogations, AWS Lambda gère le mappage de source d'événement. AWS Lambda fournit l'opération [CreateEventSourceMapping \(p. 406\)](#), qui vous permet de créer et de gérer le mappage de source d'événement. Pour en savoir plus, consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#).
- Le type d'appel que ces sources d'événements utilisent lors de l'appel d'une fonction Lambda est également préconfiguré. Par exemple, Amazon S3 appelle toujours une fonction Lambda de manière asynchrone, et Amazon Cognito de manière synchrone. La seule fois où vous pouvez contrôler le type d'appel est lorsque vous appelez la fonction Lambda vous-même via l'opération [Invoke \(p. 452\)](#) (appel d'une fonction Lambda à la demande à partir de votre application personnalisée, par exemple).
- Pour traiter les événements AWS, vos fonctions Lambda peuvent avoir besoin d'inclure des bibliothèques supplémentaires, selon le langage de programmation utilisé pour créer la fonction. Les fonctions écrites en Node.js ou Python n'ont pas besoin de bibliothèques supplémentaires. Pour C #, vous devez inclure [AWS Lambda for .NET Core](#). Pour Java, vous devez inclure [aws-lambda-java-libs](#).

Important

Chacun des packages inclus doit être utilisé sans modification. Le retrait de dépendances, l'ajout de dépendances conflictuelles ou l'inclusion sélective de classes depuis les packages peut entraîner un comportement inattendu.

Vous pouvez également appeler une fonction Lambda à la demande. Pour en savoir plus, consultez [Autres sources d'événements : appel d'une fonction Lambda à la demande \(p. 91\)](#).

Pour obtenir des exemples d'événements publiés par ces sources d'événements, consultez la page [Exemples d'événements publiés en fonction des sources \(p. 91\)](#).

Rubriques

- [Amazon S3 \(p. 84\)](#)
- [Amazon DynamoDB \(p. 84\)](#)
- [Amazon Kinesis Data Streams \(p. 84\)](#)
- [Amazon Simple Notification Service \(p. 85\)](#)
- [Amazon Simple Email Service \(p. 85\)](#)
- [Amazon Simple Queue Service \(p. 86\)](#)
- [Amazon Cognito \(p. 86\)](#)
- [AWS CloudFormation \(p. 86\)](#)
- [Amazon CloudWatch Logs \(p. 87\)](#)
- [Amazon CloudWatch Events \(p. 87\)](#)
- [AWS CodeCommit \(p. 88\)](#)
- [Événements planifiés \(via Amazon CloudWatch Events\) \(p. 88\)](#)
- [AWS Config \(p. 88\)](#)
- [Amazon Alexa \(p. 89\)](#)
- [Amazon Lex \(p. 89\)](#)
- [Amazon API Gateway \(p. 89\)](#)
- [Bouton AWS IoT \(p. 90\)](#)

- [Amazon CloudFront \(p. 90\)](#)
- [Amazon Kinesis Data Firehose \(p. 90\)](#)
- [Autres sources d'événements : appel d'une fonction Lambda à la demande \(p. 91\)](#)

Amazon S3

Vous pouvez écrire des fonctions Lambda pour traiter les événements des compartiments S3, tels que la création ou la suppression d'objets. Par exemple, lorsqu'un utilisateur importe une photo dans un compartiment, vous pouvez faire en sorte qu'Amazon S3 appelle la fonction Lambda pour qu'elle lise l'image et crée une miniature de cette photo.

Vous pouvez utiliser la fonctionnalité de configuration des notifications de compartiment d'Amazon S3 pour configurer le mappage de source d'événement, en identifiant les événements de compartiment qu'Amazon S3 doit publier, ainsi que la fonction Lambda à appeler.

Pour voir un exemple d'événement Amazon S3, consultez [Structure des messages d'événements, Exemple d'événement Put Amazon S3 \(p. 97\)](#) et [Exemple d'événement de suppression Amazon S3 \(p. 97\)](#).

Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation de AWS Lambda avec Amazon S3 \(p. 126\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon S3 appelle la fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon DynamoDB

Vous pouvez utiliser les fonctions Lambda comme déclencheurs de votre table Amazon DynamoDB. Les déclencheurs sont des actions personnalisées que vous effectuez en réponse aux mises à jour apportées à la table DynamoDB. Pour créer un déclencheur, vous devez d'abord activer Amazon DynamoDB Flux au niveau de la table. AWS Lambda interroge le flux, tandis que la fonction Lambda traite les mises à jour publiées dans le flux.

Il s'agit d'une source d'événements basée sur les flux. Pour un service basé sur les flux, vous créez le mappage de source d'événement dans AWS Lambda, en identifiant les flux à interroger et la fonction Lambda à appeler.

Pour un exemple d'événement DynamoDB, consultez [Test de la fonction Lambda \(p. 166\)](#) et [Exemple d'événement de mise à jour Amazon DynamoDB \(p. 94\)](#). Pour le format général, consultez [GetRecord](#) dans le Amazon DynamoDB API Reference. Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation de AWS Lambda avec Amazon DynamoDB \(p. 162\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon d'appeler Lambda. DynamoDB est une source d'événements basée sur les flux. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Kinesis Data Streams

Vous pouvez configurer AWS Lambda pour qu'il interroge automatiquement votre flux et pour qu'il traite les nouveaux enregistrements, tels que les flux de clics sur un site web, les transactions financières, les flux de réseaux sociaux, les journaux informatiques et les événements de géolocalisation. AWS Lambda interroge ensuite le flux régulièrement (une fois par seconde) à la recherche de nouveaux enregistrements.

Pour des services basés sur les flux, vous créez le mappage de source d'événement dans AWS Lambda, en identifiant le flux à interroger et la fonction Lambda à appeler.

Pour voir un exemple d'événement, consultez la section [Exemple d'événement Amazon Kinesis Data Streams \(p. 96\)](#). Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation de AWS Lambda avec Amazon Kinesis \(p. 142\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon d'appeler Lambda. Amazon Kinesis Data Streams est une source d'événements basée sur les flux. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Simple Notification Service

Vous pouvez écrire des fonctions Lambda pour traiter les notifications Amazon Simple Notification Service. Lorsqu'un message est publié dans une rubrique Amazon SNS, le service peut appeler la fonction Lambda en transmettant la charge utile du message comme un paramètre. Le code de la fonction Lambda peut ensuite traiter l'événement (par exemple, la publication du message dans d'autres rubriques Amazon SNS ou son envoi à d'autres services AWS).

Cela vous permet également de déclencher une fonction Lambda en réponse à des alarmes Amazon CloudWatch et à d'autres services AWS qui utilisent Amazon SNS.

Vous configurez le mappage de source d'événement dans Amazon SNS via la configuration de l'abonnement aux rubriques. Pour plus d'informations, consultez [Invocation des fonctions Lambda en utilisant des notifications Amazon SNS](#) dans le Amazon Simple Notification Service Manuel du développeur.

Pour voir un exemple d'événement, consultez [Annexe : formats de message et JSON](#) et [Exemple d'événement Amazon SNS \(p. 94\)](#). Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation d'AWS Lambda avec Amazon SNS à partir de différents comptes \(p. 181\)](#).

Lorsqu'un utilisateur appelle l'API SNS Publish sur une rubrique à laquelle votre fonction Lambda est abonnée, Amazon SNS appelle Lambda pour appeler votre fonction de façon asynchrone. Lambda renvoie ensuite un état de remise. Si une erreur s'est produite lors de l'appel de la fonction Lambda, Amazon SNS retente l'appel jusqu'à trois fois. Au-delà de trois tentatives, si Amazon SNS n'est pas parvenu à appeler la fonction Lambda, il envoie un message d'échec de la remise à CloudWatch.

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon SNS appelle la fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) est un service de messagerie électronique rentable. Avec Amazon SES, en plus d'envoyer des e-mails, vous pouvez également en recevoir. Pour plus d'informations sur Amazon SES, consultez [Amazon Simple Email Service](#). Lorsque vous utilisez Amazon SES pour recevoir des messages, vous pouvez configurer ce service de manière à appeler la fonction Lambda lorsque des messages arrivent. Le service peut ensuite appeler votre fonction Lambda en transmettant l'événement de réception d'e-mail (qui est en fait un message Amazon SES dans un événement Amazon SNS) en tant que paramètre. Pour voir des exemples de scénarios, consultez [Examen du cas d'utilisation pour la réception d'e-mails avec Amazon SES](#).

Vous configurez un mappage de source d'événement à l'aide de la configuration de règles dans Amazon SES. Les rubriques suivantes fournissent des informations supplémentaires dans le Amazon Simple Email Service Guide du développeur :

- Pour voir des exemples d'événements, consultez [Action Lambda](#) et [Exemple d'événement de réception d'e-mail Amazon SES \(p. 92\)](#).
- Pour voir des exemples de fonctions Lambda, consultez [Exemples de fonctions Lambda](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon SES appelle la fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS) vous permet de créer des workflows asynchrones. Pour plus d'informations sur Amazon SQS, consultez [Amazon Simple Queue Service](#). Vous pouvez configurer AWS Lambda afin qu'il interroge ces messages au fur et à mesure qu'ils arrivent, puis qu'il transmette l'événement à une fonction Lambda. Pour un exemple d'événement, consultez [Événement Amazon SQS \(p. 99\)](#).

Pour configurer Amazon Simple Queue Service en tant que source d'événement pour AWS Lambda, vous devez d'abord créer ou mettre à jour une file d'attente Amazon SQS et sélectionner des valeurs personnalisées pour les paramètres de la file. Les paramètres suivants affecteront le comportement d'interrogation de Amazon SQS :

- `VisibilityTimeout` : peut affecter le délai entre les tentatives.
- `TimeToWait` : détermine la durée des interrogations longues. La valeur par défaut est de 20 secondes.

Pour en savoir plus, consultez [Présentation d'Amazon Simple Queue Service](#). Vous pouvez ensuite créer votre fonction AWS Lambda en ajoutant le code de votre fonction qui gère le traitement des messages de la file d'attente Amazon SQS. Enfin, vous pouvez configurer AWS Lambda afin qu'il réponde aux nouveaux messages disponibles sur les files d'attente Amazon SQS à l'aide de l'une des opérations d'API suivantes :

- [CreateEventSourceMapping \(p. 406\)](#)
- [UpdateEventSourceMapping \(p. 504\)](#)

La gestion des erreurs pour une source d'événement donnée dépend de la façon d'appeler Lambda. Amazon SQS est un service basé sur les interrogations. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Cognito

La fonctionnalité d'événements Amazon Cognito vous permet d'exécuter des fonctions Lambda en réponse à des événements dans Amazon Cognito. Par exemple, vous pouvez appeler une fonctionLambda pour les événements de déclencheur de synchronisation, laquelle est publiée chaque fois qu'un ensemble de données est synchronisé. Pour en savoir plus et pour découvrir un exemple, consultez la section [Introducing Amazon Cognito Events: Sync Triggers](#) dans le blog Mobile Development.

Vous configurez un mappage de source d'événement à l'aide de la configuration des abonnements aux événements Amazon Cognito. Pour en savoir plus sur le mappage de source d'événement et pour voir un exemple d'événement, consultez [Événements Amazon Cognito](#) dans le Manuel du développeur Amazon Cognito. Pour voir un autre exemple d'événement, consultez la section [Exemple d'événement de déclencheur de synchronisation Amazon Cognito \(p. 96\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon Cognito est configuré de manière à appeler une fonction Lambda de façon synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

AWS CloudFormation

Dans le cadre du déploiement des piles AWS CloudFormation, vous pouvez spécifier une fonction Lambda comme ressource personnalisée afin d'exécuter des commandes personnalisées. L'association d'une

fonction Lambda à une ressource personnalisée vous permet d'appeler cette fonction chaque fois que vous créez, mettez à jour ou supprimez des piles AWS CloudFormation.

Vous configurez le mappage de source d'événement dans AWS CloudFormation à l'aide de la définition de la pile. Pour plus d'informations, consultez [Ressources personnalisées basées sur AWS Lambda](#) dans le AWS CloudFormation Guide de l'utilisateur.

Pour voir un exemple d'événement, consultez la section [Exemple d'événement de création de requête AWS CloudFormation \(p. 92\)](#). Notez que cet événement est en fait un message AWS CloudFormation dans un événement Amazon SNS.

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. AWS CloudFormation appelle la fonction Lambda de manière synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon CloudWatch Logs

Vous pouvez utiliser des fonctions AWS Lambda pour procéder à des analyses personnalisées dans Amazon CloudWatch Logs via des abonnements CloudWatch Logs. Les abonnements CloudWatch Logs donnent accès à un flux en temps réel d'événements de journal à partir de CloudWatch Logs et l'envoie à votre fonction AWS Lambda pour un traitement personnalisé, une analyse ou le chargement sur d'autres systèmes. Pour plus d'informations sur CloudWatch Logs, consultez [Surveillance des fichiers journaux](#).

Vous gérez le mappage de source d'événement dans Amazon CloudWatch Logs à l'aide de la configuration des abonnements aux journaux. Pour en savoir plus, consultez [Traitement en temps réel des données de journaux avec les abonnements \(exemple 2 : AWS Lambda\)](#) dans le Guide de l'utilisateur Amazon CloudWatch.

Pour voir un exemple d'événement, consultez la section [Exemple d'événement Amazon CloudWatch Logs \(p. 94\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon CloudWatch Logs appelle la fonction Lambda de manière asynchrone (l'appel d'une fonction Lambda ne bloque pas l'opération d'écriture dans les journaux). Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon CloudWatch Events

Les [événements Amazon CloudWatch](#) vous permettent de réagir aux modifications d'état de vos ressources AWS. Lorsque l'état de vos ressources change, des événements sont automatiquement envoyés à un flux d'événements. Vous pouvez créer des règles en fonction de certains événements du flux et les transmettre à la fonction AWS Lambda afin d'agir en conséquence. Par exemple, vous pouvez appeler automatiquement une fonction AWS Lambda pour consigner l'état d'une [instance EC2](#) ou d'un [groupe AutoScaling](#).

Vous gérez le mappage de source d'événement dans Amazon CloudWatch Events en utilisant une définition de règle cible. Pour plus d'informations, consultez l'opération [PutTargets](#) dans le document Référence d'API Amazon CloudWatch Events.

Pour voir des exemples d'événements, consultez [Types d'événements pris en charge](#) dans le Guide de l'utilisateur Amazon CloudWatch.

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon CloudWatch Events appelle la fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

AWS CodeCommit

Vous pouvez créer un déclencheur pour un référentiel AWS CodeCommit afin que des événements dans le référentiel appellent une fonction Lambda. Par exemple, vous pouvez appeler une fonction Lambda lorsqu'une branche ou une balise est créée, ou lorsqu'un push est effectué sur une branche existante. Pour plus d'informations, consultez [Gestion des déclencheurs pour un référentiel AWS CodeCommit](#).

Vous gérez le mappage de source d'événement dans AWS CodeCommit en utilisant un déclencheur de référentiel. Pour plus d'informations, consultez l'opération [PutRepositoryTriggers](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. AWS CodeCommit appelle la fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Événements planifiés (via Amazon CloudWatch Events)

Vous pouvez également configurer AWS Lambda pour appeler votre code sur une base régulière avec la fonctionnalité de planification d'événements d'Amazon CloudWatch Events. Pour configurer une planification, vous pouvez spécifier un taux fixe (nombre d'heures, de jours ou de semaines) ou une expression cron (voir [Syntaxe d'expression de planification des règles](#) dans le Guide de l'utilisateur Amazon CloudWatch).

Vous gérez le mappage de source d'événement dans Amazon CloudWatch Events en utilisant une définition de règle cible. Pour plus d'informations, consultez l'opération [PutTargets](#) dans le document Référence d'API Amazon CloudWatch Events.

Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation de AWS Lambda avec Amazon CloudWatch Events \(p. 209\)](#).

Pour voir un exemple d'événement, consultez la section [Exemple d'événement planifié \(p. 93\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon CloudWatch Events est configuré pour appeler une fonction Lambda de manière asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

AWS Config

Vous pouvez utiliser des fonctions AWS Lambda pour évaluer si les configurations des ressources AWS respectent vos règles de configuration personnalisées. Lorsque les ressources sont créées, supprimées ou modifiées, AWS Config enregistre ces modifications et envoie les informations aux fonctions Lambda. Ces dernières évaluent alors les modifications et transmettent les résultats à AWS Config. Vous pouvez ensuite utiliser AWS Config afin d'estimer la conformité globale des ressources : vous pouvez déterminer quelles ressources ne sont pas conformes et quels attributs de configuration sont à l'origine de cette non-conformité.

Vous gérez le mappage de source d'événement dans AWS Config en utilisant une définition de règle cible. Pour plus d'informations, consultez l'opération [PutConfigRule](#) dans la référence d'API d'AWS Config.

Pour plus d'informations, consultez [Évaluation des ressources avec les règles AWS Config](#). Pour obtenir un exemple de configuration d'une règle personnalisée, consultez [Élaboration d'une règle personnalisée pour AWS Config](#). Pour des exemples de fonctions Lambda, consultez [Exemples de fonctions AWS Lambda pour les règles AWS Config \(Node.js\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. AWS Config est configuré de manière à appeler une fonction Lambda de façon asynchrone. Pour en savoir

plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Alexa

Vous pouvez utiliser les fonctions Lambda pour concevoir des services qui confèrent de nouvelles compétences à Alexa, l'assistante vocale d'Amazon Echo. Le kit Alexa Skills fournit les API, les outils et la documentation nécessaires à la création de ces nouvelles compétences, sur la base de vos propres services exécutés en tant que fonctions Lambda. Les utilisateurs d'Amazon Echo peuvent accéder à ces nouvelles compétences en posant des questions à Alexa ou en soumettant des demandes. Pour en savoir plus, consultez :

- [Mise en route avec le kit Alexa Skills.](#)
- [alexa-skills-kit-sdk-for-nodejs](#)
- [alexa-skills-kit-java](#)

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon Echo est configuré pour appeler une fonction Lambda de manière synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Lex

Amazon Lex est un service AWS de création d'interfaces conversationnelles au sein des applications, qui utilise la voix et le texte. Amazon Lex offre une intégration préconçue avec AWS Lambda, ce qui vous permet de créer des fonctions Lambda que vous pourrez utiliser comme hook de code avec votre bot Amazon Lex. Vous pouvez, dans la configuration que vous souhaitez utiliser, identifier votre fonction Lambda afin d'effectuer l'initialisation/validation, l'exécution ou les deux.

Pour plus d'informations, consultez [Utilisation des fonctions Lambda](#). Pour voir un exemple de cas d'utilisation, consultez [Exercice 1 : Création d'un bot Amazon Lex à l'aide d'un modèle](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon Lex est configuré pour appeler une fonction Lambda de manière synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon API Gateway

Vous pouvez appeler une fonction Lambda sur HTTPS. Pour ce faire, définissez un point de terminaison d'API REST personnalisé via Amazon API Gateway. Vous mappez des opérations d'API individuelles, comme GET et PUT, avec des fonctions Lambda spécifiques. Lorsque vous envoyez une requête HTTPS au point de terminaison de l'API, le service Amazon API Gateway appelle la fonction Lambda correspondante.

Pour plus d'informations, consultez [Appels synchrones de fonctions Lambda](#). Pour voir un exemple de cas d'utilisation, consultez la page [Utilisation de AWS Lambda avec Amazon API Gateway \(p. 186\)](#).

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Amazon API Gateway est configuré de manière à appeler une fonction Lambda de façon synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

En outre, vous pouvez utiliser les fonctions Lambda avec d'autres services AWS qui publient des données dans l'une des sources d'événements AWS prises en charge et répertoriés dans cette rubrique. Par exemple, vous pouvez :

- Déclencher des fonctions Lambda en réponse à des mises à jour CloudTrail, car il enregistre tous les événements d'accès aux API dans un compartiment Amazon S3.
- Déclencher des fonctions Lambda en réponse à des alarmes CloudWatch, car il publie des événements d'alarme dans une rubrique Amazon SNS.

Bouton AWS IoT

Le bouton AWS IoT est un bouton programmable basé sur les composants du bouton Amazon Dash. Ce simple appareil Wi-Fi est facile à configurer et conçu pour permettre aux développeurs de commencer avec AWS Lambda, parmi de nombreux autres services AWS, sans avoir à écrire de code propre aux appareils.

Codez la logique du bouton dans le cloud pour que les pressions servent à compter ou suivre des éléments, à appeler ou alerter quelqu'un, à démarrer ou arrêter quelque chose, à commander des services ou même à donner un avis. Il est par exemple possible d'appuyer sur le bouton pour déverrouiller ou démarrer une voiture, ouvrir votre porte de garage, appeler un taxi, appeler votre conjoint ou un service client, suivre la réalisation des tâches ménagères, la prise de médicaments et l'utilisation de produits domestiques, ou contrôler vos appareils domestiques à distance.

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. AWS IoT est configuré de manière à appeler une fonction Lambda de façon asynchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon CloudFront

Lambda@Edge vous permet d'exécuter les fonctions Lambda dans les régions AWS et aux emplacements périphériques Amazon CloudFront en réponse à des événements CloudFront, sans mise en service ou gestion de serveurs. Vous pouvez utiliser les fonctions Lambda pour modifier les requêtes et réponses CloudFront aux stades suivants :

- Après la réception par CloudFront d'une demande provenant de l'utilisateur (demande de l'utilisateur)
- Avant la transmission par CloudFront d'une demande à l'origine (demande à l'origine)
- Après la réception par CloudFront de la réponse provenant de l'origine (réponse de l'origine)
- Avant la transmission par CloudFront de la réponse pour l'utilisateur (réponse à l'utilisateur)

Pour en savoir plus, consultez [Lambda@Edge \(p. 221\)](#)

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. CloudFront est configuré de manière à appeler une fonction Lambda de façon synchrone. Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose est la solution la plus simple pour charger les données de streaming dans AWS. Cette solution peut capturer, transformer et charger automatiquement les données de streaming dans des services en aval comme Kinesis Data Analytics ou Amazon S3, ce qui permet une analyse en temps quasi-réel à l'aide des outils d'aide à la décision existants et des tableaux de bord que vous utilisez déjà. Vous pouvez écrire des fonctions Lambda pour demander un traitement personnalisé supplémentaire des données avant que ces dernières ne soient envoyées en aval.

La gestion des erreurs pour une source d'événement donnée dépend de la façon dont Lambda est appelé. Kinesis Data Firehose est configuré de manière à appeler une fonction Lambda de façon synchrone.

Pour en savoir plus sur les nouvelles tentatives après des erreurs, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Autres sources d'événements : appel d'une fonction Lambda à la demande

En plus d'appeler des fonctions Lambda à l'aide de sources d'événements, vous pouvez également les appeler à la demande. Dans ce cas, vous n'avez pas besoin de préconfigurer de mappage de source d'événement. Cependant, assurez-vous que l'application personnalisée dispose des autorisations nécessaires pour appeler la fonction Lambda.

Par exemple, les applications utilisateur peuvent également générer des événements (créez vos propres sources d'événements personnalisées). Les applications utilisateur, telles que les applications client, mobiles ou web, peuvent publier des événements et appeler les fonctions Lambda via les kits SDK AWS ou AWS Mobile (AWS Mobile SDK pour Android, par exemple).

Pour plus d'informations, consultez [Outils pour Amazon web Services](#). Pour voir un exemple de didacticiel, consultez la section [Utilisation de AWS Lambda avec Amazon API Gateway \(p. 186\)](#).

Exemples d'événements publiés en fonction des sources

Voici une liste d'exemples d'événements publiés par les services AWS pris en charge. Pour plus d'informations sur les sources d'événement AWS prises en charge, consultez la section [Sources d'événements prises en charge \(p. 82\)](#).

Note

La casse des clés JSON peut varier entre des sources d'événements AWS.

Exemples d'événements

- [Exemple d'événement de création de requête AWS CloudFormation \(p. 92\)](#)
- [Exemple d'événement de réception d'e-mail Amazon SES \(p. 92\)](#)
- [Exemple d'événement planifié \(p. 93\)](#)
- [Exemple d'événement Amazon CloudWatch Logs \(p. 94\)](#)
- [Exemple d'événement Amazon SNS \(p. 94\)](#)
- [Exemple d'événement de mise à jour Amazon DynamoDB \(p. 94\)](#)
- [Exemple d'événement de déclencheur de synchronisation Amazon Cognito \(p. 96\)](#)
- [Exemple d'événement Amazon Kinesis Data Streams \(p. 96\)](#)
- [Exemple d'événement Put Amazon S3 \(p. 97\)](#)
- [Exemple d'événement de suppression Amazon S3 \(p. 97\)](#)
- [Exemple d'événement Amazon Lex \(p. 98\)](#)
- [Événement de demande de proxy API Gateway \(p. 98\)](#)
- [Événement de réponse de proxy API Gateway \(p. 100\)](#)
- [Événement Amazon SQS \(p. 99\)](#)
- [Événement CloudFront \(p. 100\)](#)
- [Événement AWS Config \(p. 101\)](#)
- [Événement de bouton AWS IoT \(p. 101\)](#)
- [Événement Kinesis Data Firehose \(p. 101\)](#)

Exemple d'événement de création de requête AWS CloudFormation

```
{  
  "StackId": "arn:aws:cloudformation:us-west-2:EXAMPLE/stack-name/guid",  
  "ResponseURL": "http://pre-signed-S3-url-for-response",  
  "ResourceProperties": {  
    "StackName": "stack-name",  
    "List": [  
      "1",  
      "2",  
      "3"  
    ]  
  },  
  "RequestType": "Create",  
  "ResourceType": "Custom::TestResource",  
  "RequestId": "unique id for this create request",  
  "LogicalResourceId": "MyTestResource"  
}
```

Exemple d'événement de réception d'e-mail Amazon SES

```
{  
  "Records": [  
    {  
      "eventVersion": "1.0",  
      "ses": {  
        "mail": {  
          "commonHeaders": {  
            "from": [  
              "Jane Doe <janedoe@example.com>"  
            ],  
            "to": [  
              "johndoe@example.com"  
            ],  
            "returnPath": "janedoe@example.com",  
            "messageId": "<0123456789example.com>",  
            "date": "Wed, 7 Oct 2015 12:34:56 -0700",  
            "subject": "Test Subject"  
          },  
          "source": "janedoe@example.com",  
          "timestamp": "1970-01-01T00:00:00.000Z",  
          "destination": [  
            "johndoe@example.com"  
          ],  
          "headers": [  
            {  
              "name": "Return-Path",  
              "value": "<janedoe@example.com>"  
            },  
            {  
              "name": "Received",  
              "value": "from mailer.example.com (mailer.example.com [203.0.113.1])  
by inbound-smtp.us-west-2.amazonaws.com with SMTP id o3vrnil0e2ic for  
johndoe@example.com; Wed, 07 Oct 2015 12:34:56 +0000 (UTC)"  
            },  
            {  
              "name": "DKIM-Signature",  
              "value": "v=1; a=rsa-sha256; c=relaxed/relaxed; d=example.com;  
s=example; h=mime-version:from:date:message-id:subject:to:content-type;  
bh=jX3F0bCAI7sIbkHyy3mLYO28ieDQz2R0P8HwQkklFj4=; b=sQwJ+LMe9RjkesGu  
+vqU56asvMhrLRRYrWCbV"  
            },  
            {  
              "name": "MIME-Version",  
              "value": "1.0"  
            }  
          ]  
        }  
      }  
    }  
  ]  
}
```

```
        "value": "1.0"
    },
    {
        "name": "From",
        "value": "Jane Doe <janedoe@example.com>"
    },
    {
        "name": "Date",
        "value": "Wed, 7 Oct 2015 12:34:56 -0700"
    },
    {
        "name": "Message-ID",
        "value": "<0123456789example.com>"
    },
    {
        "name": "Subject",
        "value": "Test Subject"
    },
    {
        "name": "To",
        "value": "johndoe@example.com"
    },
    {
        "name": "Content-Type",
        "value": "text/plain; charset=UTF-8"
    }
],
"headersTruncated": false,
"messageId": "o3vrnil0e2ic28tr"
},
"receipt": {
    "recipients": [
        "johndoe@example.com"
    ],
    "timestamp": "1970-01-01T00:00:00.000Z",
    "spamVerdict": {
        "status": "PASS"
    },
    "dkimVerdict": {
        "status": "PASS"
    },
    "processingTimeMillis": 574,
    "action": {
        "type": "Lambda",
        "invocationType": "Event",
        "functionArn": "arn:aws:lambda:us-west-2:012345678912:function:Example"
    },
    "spfVerdict": {
        "status": "PASS"
    },
    "virusVerdict": {
        "status": "PASS"
    }
},
"eventSource": "aws:ses"
}
]
}
```

Exemple d'événement planifié

```
{
    "account": "123456789012",
    "region": "us-east-1",
```

```
    "detail": {},
    "detail-type": "Scheduled Event",
    "source": "aws.events",
    "time": "1970-01-01T00:00:00Z",
    "id": "cdc73f9d-aea9-11e3-9d5a-835b769c0d9c",
    "resources": [
        "arn:aws:events:us-east-1:123456789012:rule/my-schedule"
    ]
}
```

Exemple d'événement Amazon CloudWatch Logs

```
{
  "awslogs": {
    "data": "H4sIAAAAAAAAHWPwQqCQBCGX0Xm7EFtK
+smZBEUgXoLCdMhFtKV3akI8d0bLYmibvPPN3wz00CJxmQnTO41whwWQRICtmEcB6sQbFC3CjW3XW8kxpOpP
+OC22d1Wml1qZkOGtoMsScxaczKN3plG8zlaHIta5KqWsozoTYw3/djzwhpLwivWFHGhpAFe7DL68JlBUk
+17KSN7tCOEJ4M3/qOI49vMHj+zCKdlFqLaU2ZHV2a4Ct/an0/ivdx8oYc1UVX860fQDQiMdxRQEAAA=="
  }
}
```

Exemple d'événement Amazon SNS

```
{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSubscriptionArn": eventsubscriptionarn,
      "EventSource": "aws:sns",
      "Sns": {
        "SignatureVersion": "1",
        "Timestamp": "1970-01-01T00:00:00.000Z",
        "Signature": "EXAMPLE",
        "SigningCertUrl": "EXAMPLE",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        "MessageAttributes": {
          "Test": {
            "Type": "String",
            "Value": "TestString"
          },
          "TestBinary": {
            "Type": "Binary",
            "Value": "TestBinary"
          }
        },
        "Type": "Notification",
        "UnsubscribeUrl": "EXAMPLE",
        "TopicArn": topicarn,
        "Subject": "TestInvoke"
      }
    }
  ]
}
```

Exemple d'événement de mise à jour Amazon DynamoDB

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "eventTime": "2018-04-17T22:27:52.600Z",
      "eventName": "aws:batch:inventories:PutInventoryItem",
      "awsRegion": "us-east-1",
      "eventSource": "aws:batch:inventories",
      "eventSourceARN": "arn:aws:batch:us-east-1:123456789012:inventories/MyInventories",
      "awsShardId": null,
      "approximateArrivalTimestamp": 1524183272600
    }
  ]
}
```

```
"dynamodb": {
    "Keys": {
        "Id": {
            "N": "101"
        }
    },
    "NewImage": {
        "Message": {
            "S": "New item!"
        },
        "Id": {
            "N": "101"
        }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "111",
    "SizeBytes": 26
},
"awsRegion": "us-west-2",
"eventName": "INSERT",
"eventSourceARN": eventsourcearn,
"eventSource": "aws:dynamodb"
},
{
    "eventID": "2",
    "eventVersion": "1.0",
    "dynamodb": {
        "OldImage": {
            "Message": {
                "S": "New item!"
            },
            "Id": {
                "N": "101"
            }
        },
        "SequenceNumber": "222",
        "Keys": {
            "Id": {
                "N": "101"
            }
        },
        "SizeBytes": 59,
        "NewImage": {
            "Message": {
                "S": "This item has changed"
            },
            "Id": {
                "N": "101"
            }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
},
"awsRegion": "us-west-2",
"eventName": "MODIFY",
"eventSourceARN": sourcearn,
"eventSource": "aws:dynamodb"
},
{
    "eventID": "3",
    "eventVersion": "1.0",
    "dynamodb": {
        "Keys": {
            "Id": {
                "N": "101"
            }
        },
        "SizeBytes": 26
}
```

```
        "SizeBytes": 38,
        "SequenceNumber": "333",
        "OldImage": {
            "Message": {
                "S": "This item has changed"
            },
            "Id": {
                "N": "101"
            }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "awsRegion": "us-west-2",
    "eventName": "REMOVE",
    "eventSourceARN": sourcearn,
    "eventSource": "aws:dynamodb"
}
]
```

Exemple d'événement de déclencheur de synchronisation Amazon Cognito

```
{
    "datasetName": "datasetName",
    "eventType": "SyncTrigger",
    "region": "us-east-1",
    "identityId": "identityId",
    "datasetRecords": {
        "SampleKey2": {
            "newValue": "newValue2",
            "oldValue": "oldValue2",
            "op": "replace"
        },
        "SampleKey1": {
            "newValue": "newValue1",
            "oldValue": "oldValue1",
            "op": "replace"
        }
    },
    "identityPoolId": "identityPoolId",
    "version": 2
}
```

Exemple d'événement Amazon Kinesis Data Streams

```
{
    "Records": [
        {
            "eventID": "shardId-000000000000:49545115243490985018280067714973144582180062593244200961",
            "eventVersion": "1.0",
            "kinesis": {
                "partitionKey": "partitionKey-3",
                "data": "SGVsbG8sIHRoaXMgYSB0ZXN0IDEyMy4=",
                "kinesisSchemaVersion": "1.0",
                "sequenceNumber": "49545115243490985018280067714973144582180062593244200961"
            },
            "invokeIdentityArn": identityarn,
            "eventName": "aws:kinesis:record",
            "approximateArrivalTimestamp": 15243490985018280067714973144582180062593244200961
        }
    ]
}
```

```
        "eventSourceARN": eventsourcearn,
        "eventSource": "aws:kinesis",
        "awsRegion": "us-east-1"
    }
]
}
```

Exemple d'événement Put Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901",
          "key": "HappyFace.jpg",
          "size": 1024
        },
        "bucket": {
          "arn": bucketarn,
          "name": "sourcebucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          }
        },
        "s3SchemaVersion": "1.0"
      },
      "responseElements": {
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmnaisawesome/mnopqrstuvwxyzABCDEFGH",
        "x-amz-request-id": "EXAMPLE123456789"
      },
      "awsRegion": "us-east-1",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "eventSource": "aws:s3"
    }
  ]
}
```

Exemple d'événement de suppression Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901",
          "key": "HappyFace.jpg",
          "size": 1024
        }
      }
    }
  ]
}
```

```
    "object": {
        "sequencer": "0A1B2C3D4E5F678901",
        "key": "HappyFace.jpg"
    },
    "bucket": {
        "arn": bucketarn,
        "name": "sourcebucket",
        "ownerIdentity": {
            "principalId": "EXAMPLE"
        }
    },
    "s3SchemaVersion": "1.0"
},
"responseElements": {
    "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmnaisawesome/
mnopqrstuvwxyzABCDEFGHI",
    "x-amz-request-id": "EXAMPLE123456789"
},
"awsRegion": "us-east-1",
"eventName": "ObjectRemoved:Delete",
"userIdentity": {
    "principalId": "EXAMPLE"
},
"eventSource": "aws:s3"
}
]
}
```

Exemple d'événement Amazon Lex

```
{
    "messageVersion": "1.0",
    "invocationSource": "FulfillmentCodeHook or DialogCodeHook",
    "userId": "user-id specified in the POST request to Amazon Lex.",
    "sessionAttributes": {
        "key1": "value1",
        "key2": "value2",
    },
    "bot": {
        "name": "bot-name",
        "alias": "bot-alias",
        "version": "bot-version"
    },
    "outputDialogMode": "Text or Voice, based on ContentType request header in runtime API request",
    "currentIntent": {
        "name": "intent-name",
        "slots": {
            "slot-name": "value",
            "slot-name": "value",
            "slot-name": "value"
        },
        "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if configured)"
    }
}
```

Événement de demande de proxy API Gateway

```
{
    "path": "/test/hello",
```

```

    "headers": {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
*q=0.8",
        "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
        "Accept-Language": "en-US,en;q=0.8",
        "CloudFront-Forwarded-Proto": "https",
        "CloudFront-Is-Desktop-Viewer": "true",
        "CloudFront-Is-Mobile-Viewer": "false",
        "CloudFront-Is-SmartTV-Viewer": "false",
        "CloudFront-Is-Tablet-Viewer": "false",
        "CloudFront-Viewer-Country": "US",
        "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
        "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
        "X-Amz-Cf-Id": "nBsWBOrSHMgnaROZJK1wGCZ9PcRcSpq_oSXZNQwQ10OTZL4cimZo3g==",
        "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "pathParameters": {
        "proxy": "hello"
    },
    "requestContext": {
        "accountId": "123456789012",
        "resourceId": "us4z18",
        "stage": "test",
        "requestId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9",
        "identity": {
            "cognitoIdentityPoolId": "",
            "accountId": "",
            "cognitoIdentityId": "",
            "caller": "",
            "apiKey": "",
            "sourceIp": "192.168.100.1",
            "cognitoAuthenticationType": "",
            "cognitoAuthenticationProvider": "",
            "userArn": "",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
            "user": ""
        },
        "resourcePath": "/{proxy+}",
        "httpMethod": "GET",
        "apiId": "wt6mne2s9k"
    },
    "resource": "/{proxy+}",
    "httpMethod": "GET",
    "queryStringParameters": {
        "name": "me"
    },
    "stageVariables": {
        "stageVarName": "stageVarValue"
    }
}

```

Événement Amazon SQS

```
{
    "Records": [
        {
            "messageId": "c80e8021-a70a-42c7-a470-796e1186f753",

```

```

    "receiptHandle": "AQEBJQ+/u6NsnT5t8Q/
VbVxgdU14TMKZ5FqhksRdIQvLBhwNvADoBxYSOVeCBXdnS9P
+erlTtwEALHsnBXynkfPLH3BOUqmgzP25U8kl8eHzq6RALzrSOfTO8ox9dcP6GLmW33YjO3zkq5VRYYQ1JgLCiAZUpY2D4UQcE5
+xtVaOctYeINjaQJ1u3mWx9T7tork3uAl0e1uyFjCWU5aPX/10HhWCGi2EPPZj6vchNqDOJC/
Y2k1gkivqCjz1CZ16F1Z7UVPOx3AMoszPuOYZ
+Nuqpx2uCE2MHTtMHD8PVjlsWirt56oUr6JPp9aRG06bitPIOmi4dX0FmuMKD6u/JnuZCp+AXtJVTmSHS8IXt/
twsKU7A+fiMK01NtD5msNgVPoe9JbFtlGwvTQ==",
    "body": "{\"foo\": \"bar\"}",
    "attributes": {
        "ApproximateReceiveCount": "3",
        "SentTimestamp": "1529104986221",
        "SenderId": "594035263019",
        "ApproximateFirstReceiveTimestamp": "1529104986230"
    },
    "messageAttributes": {},
    "md5OfBody": "9bb58f26192e4ba00f01e2e7b136bbd8",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-west-2:594035263019:NOTFIFOQUEUE",
    "awsRegion": "us-west-2"
}
]
}

```

Événement de réponse de proxy API Gateway

```
{
    "statusCode": 200,
    "headers": {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
        "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
        "Accept-Language": "en-US,en;q=0.8",
        "CloudFront-Forwarded-Proto": "https",
        "CloudFront-Is-Desktop-Viewer": "true",
        "CloudFront-Is-Mobile-Viewer": "false",
        "CloudFront-Is-SmartTV-Viewer": "false",
        "CloudFront-Is-Tablet-Viewer": "false",
        "CloudFront-Viewer-Country": "US",
        "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
        "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
        "X-Amz-Cf-Id": "nBsWBOrSHMgnAROZJK1wGCZ9PcRcSpq_oSXZNQwQ100TZL4cimZo3g==",
        "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "body": "Hello World"
}
```

Événement CloudFront

```
{
    "Records": [
        {
            "cf": {
                "config": {
                    "distributionId": "EDFDVBD6EXAMPLE"
                },
                "request": {
                    "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",
                    "method": "GET",

```

```
        "uri": "/picture.jpg",
        "headers": [
            "host": [
                {
                    "key": "Host",
                    "value": "d111111abcdef8.cloudfront.net"
                }
            ],
            "user-agent": [
                {
                    "key": "User-Agent",
                    "value": "curl/7.51.0"
                }
            ]
        }
    }
}
```

Événement AWS Config

```
{  
    "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\":\"2016-02-17T01:36:34.043Z\",\"awsAccountId\":\"000000000000\",\"configurationItemStatus\":\"OK\",\"resourceId\":\"i-00000000\",\"ARN\":\"arn:aws:ec2:us-east-1:000000000000:instance/i-00000000\",\"awsRegion\":\"us-east-1\",\"availabilityZone\":\"us-east-1a\",\"resourceType\":\"AWS::EC2::Instance\",\"tags\":{\"Foo\":\"Bar\"},\"relationships\":[{\"resourceId\":\"eipalloc-00000000\"},\"resourceType\":\"AWS::EC2::EIP\"],\"name\":\"Is attached to ElasticIp\"},\"configuration\":{\"foo\":\"bar\"},\"messageType\":\"ConfigurationItemChangeNotification\"},  
    "ruleParameters": {"myParameterKey\":\"myParameterValue\"},  
    "resultToken": "myResultToken",  
    "eventLeftScope": false,  
    "executionRoleArn": "arn:aws:iam::012345678912:role/config-role",  
    "configRuleArn": "arn:aws:config:us-east-1:012345678912:config-rule/config-rule-0123456",  
    "configRuleName": "change-triggered-config-rule",  
    "configRuleId": "config-rule-0123456",  
    "accountId": "012345678912",  
    "version": "1.0"  
}
```

Événement de bouton AWS IoT

```
{  
    "serialNumber": "ABCDEFG12345",  
    "clickType": "SINGLE",  
    "batteryVoltage": "2000 mV"  
}
```

Événement Kinesis Data Firehose

```
{  
  "invocationId": "invoked123",  
  "deliveryStreamArn": "aws:lambda:events",  
  "region": "us-west-2".
```

```
"records": [
  {
    "data": "SGVsbG8gV29ybGQ=",
    "recordId": "record1",
    "approximateArrivalTimestamp": 1510772160000,
    "kinesisRecordMetadata": {
      "shardId": "shardId-000000000000",
      "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c317a",
      "approximateArrivalTimestamp": "2012-04-23T18:25:43.511Z",
      "sequenceNumber": "49546986683135544286507457936321625675700192471156785154",
      "subsequenceNumber": ""
    }
  },
  {
    "data": "SGVsbG8gV29ybGQ=",
    "recordId": "record2",
    "approximateArrivalTimestamp": 151077216000,
    "kinesisRecordMetadata": {
      "shardId": "shardId-000000000001",
      "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",
      "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",
      "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",
      "subsequenceNumber": ""
    }
  }
]
```

Runtimes AWS Lambda

AWS Lambda prend en charge plusieurs langages grâce à l'utilisation des runtimes. Vous choisissez un runtime lorsque vous créez une fonction, et vous pouvez modifier les runtimes en mettant à jour la configuration de votre fonction. L'[environnement d'exécution \(p. 104\)](#) sous-jacent, qui est partagé par tous les runtimes, fournit des bibliothèques et des variables d'environnement supplémentaires auxquelles vous pouvez accéder depuis le code de votre fonction.

Lorsque votre fonction est appelée, Lambda tente de réutiliser l'environnement d'exécution à partir d'un appel précédent, le cas échéant. Cela permet d'économiser du temps pendant la préparation de l'environnement d'exécution, et vous permet d'économiser des ressources telles que des connexions de base de données et des fichiers temporaires dans le [contexte d'exécution \(p. 105\)](#) pour éviter de les créer chaque fois que votre fonction s'exécute.

Un runtime peut prendre en charge une seule version d'un langage, plusieurs versions d'un langage ou plusieurs langages. Les runtimes spécifiques à un langage ou à une version d'infrastructure sont [déconseillés \(p. 106\)](#) lorsque la version arrive en fin de vie.

Pour utiliser d'autres langages dans Lambda, vous pouvez implémenter un [runtime personnalisé \(p. 107\)](#). L'environnement d'exécution de Lambda fournit une [interface de runtime \(p. 109\)](#) pour obtenir les événements d'appels et envoyer les réponses. Vous pouvez déployer un runtime personnalisé en association avec le code de votre fonction, ou dans une [couche \(p. 56\)](#).

Runtimes Node.js

Nom	Identificateur
Node.js 8.10	<code>nodejs8.10</code>
Node.js 6.10	<code>nodejs6.10</code>

Runtimes Python

Nom	Identificateur
Python 3.6	<code>python3.6</code>
Python 3.7	<code>python3.7</code>
Python 2.7	<code>python2.7</code>

Runtimes Ruby

Nom	Identificateur
Ruby 2.5	<code>ruby2.5</code>

Runtimes Java

Nom	Identificateur	JDK
Java 8	<code>java8</code>	<code>java-1.8.0-openjdk</code>

Runtimes Go

Nom	Identificateur
Go 1.x	go1.x

Runtimes .NET

Nom	Identificateur	Langages
.NET Core 2.1	dotnetcore2.1	C#
		PowerShell Core 6.0
.NET Core 2.0	dotnetcore2.0	C#
.NET Core 1.0	dotnetcore1.0	C#

Rubriques

- [Environnement d'exécution Lambda et bibliothèques disponibles \(p. 104\)](#)
- [Contexte d'exécution d'AWS Lambda \(p. 105\)](#)
- [stratégie de prise en charge de l'exécution \(p. 106\)](#)
- [Runtimes AWS Lambda personnalisés \(p. 107\)](#)
- [Interface de runtime AWS Lambda \(p. 109\)](#)
- [Didacticiel – Publication d'un runtime personnalisé \(p. 111\)](#)

Environnement d'exécution Lambda et bibliothèques disponibles

L'environnement d'exécution sous-jacent AWS Lambda inclut les logiciels et les bibliothèques suivantes.

- Système d'exploitation : Amazon Linux
- AMI – [amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2](#)
- Noyau Linux – 4.14.77-70.59.amzn1.x86_64
- AWS SDK for JavaScript – 2.290.0
- Kit SDK pour Python (Boto3) : 3-1.7.74 botocore-1.10.74

Tous les langages d'exécution ne sont pas disponibles sur l'AMI Amazon Linux et ses référentiels yum. Si vous utilisez Amazon Linux pour le développement, vous devrez peut-être les télécharger et les installer manuellement depuis leurs sites publics respectifs.

Variables d'environnement disponibles pour les fonctions Lambda

Voici la liste des variables d'environnement qui font partie de l'environnement d'exécution AWS Lambda et qui sont rendues disponibles pour les fonctions Lambda. Le tableau ci-dessous indique les valeurs réservées par AWS Lambda qui ne peuvent pas être modifiées, ainsi que celles que vous pouvez définir lors de la création de votre fonction Lambda. Pour plus d'informations sur l'utilisation des variables d'environnement avec votre fonction Lambda, consultez [Variables d'environnement AWS Lambda \(p. 30\)](#).

Variables d'environnement Lambda

Clé	Instances réservées	Value
_HANDLER	Oui	L'emplacement du gestionnaire configuré sur la fonction.
AWS_REGION	Oui	La région AWS où la fonction Lambda est exécutée.
AWS_EXECUTION_ENV	Oui	L' identificateur de runtime (p. 103) doté du préfixe AWS_Lambda_. Par exemple, AWS_Lambda_java8.
AWS_LAMBDA_FUNCTION_NAME	Oui	Nom de la fonction.
AWS_LAMBDA_FUNCTION_MEMORY_SIZE	Oui	La quantité de mémoire disponible pour la fonction en Mo.
AWS_LAMBDA_FUNCTION_VERSION	Oui	La version de la fonction en cours d'exécution.
AWS_LAMBDA_LOG_GROUP_NAME	Oui	Le nom du groupe Amazon CloudWatch Logs et du flux pour la fonction.
AWS_LAMBDA_LOG_STREAM_NAME		
AWS_ACCESS_KEY_ID	Oui	Les clés d'accès obtenues à partir du rôle d'exécution (p. 261) de la fonction.
AWS_SECRET_ACCESS_KEY		
AWS_SESSION_TOKEN		
LANG	Non	en_US.UTF-8. Il s'agit de la locale de l'environnement d'exécution.
TZ	Oui	Le fuseau horaire (UTC) de l'environnement. L'environnement d'exécution utilise NTP pour synchroniser l'horloge système.
LAMBDA_TASK_ROOT	Oui	Le chemin vers le code de votre fonction Lambda.
LAMBDA_RUNTIME_DIR	Oui	Le chemin vers les bibliothèques d'exécution.
PATH	Non	/usr/local/bin:/usr/bin:/bin:/opt/bin
LD_LIBRARY_PATH	Non	/lib64:/usr/lib64:\$LAMBDA_RUNTIME_DIR:\$LAMBDA_RUNTIME_DIR/lib:\$LAMBDA_TASK_ROOT:\$LAMBDA_TASK_ROOT/lib:/opt/lib
NODE_PATH	Non	(Node.js) /opt/nodejs/node8/node_modules:/opt/nodejs/node_modules:\$LAMBDA_RUNTIME_DIR/node_modules
PYTHONPATH	Non	(Python) \$LAMBDA_RUNTIME_DIR.
AWS_LAMBDA_RUNTIME_API	Oui	(runtime personnalisé) Hôte et port de l' API du runtime (p. 109) .

Contexte d'exécution d'AWS Lambda

Quand AWS Lambda exécute votre fonction Lambda, il alloue et gère les ressources nécessaires à l'exécution de votre fonction Lambda. Lorsque vous créez une fonction Lambda, vous spécifiez les informations de configuration, telles que la quantité de mémoire et sa durée maximale d'exécution. Lorsqu'une fonction Lambda est appelée, AWS Lambda lance un contexte d'exécution basé sur les

paramètres de configuration que vous fournissez. Le contexte d'exécution est un environnement d'exécution temporaire qui initialise toutes les dépendances externes du code de votre fonction Lambda, telles que les connexions de base de données et les points de terminaison HTTP. Les appels suivants bénéficient ainsi de meilleures performances, car il n'y a pas besoin de « démarrage à froid » ou d'initialisation de ces dépendances externes, comme expliqué ci-dessous.

La configuration d'un contexte d'exécution et l'amorçage nécessaire prennent du temps, ce qui implique une certaine latence chaque fois que la fonction Lambda est appelée. En général, vous observez cette latence lorsqu'une fonction Lambda est appelée pour la première fois ou après sa mise à jour, car AWS Lambda essaie de réutiliser le contexte d'exécution pour les appels ultérieurs de la fonction Lambda.

Après l'exécution d'une fonction Lambda, AWS Lambda gère le contexte d'exécution pendant un certain temps en prévision d'un autre appel de la fonction Lambda. En effet, le service gèle le contexte d'exécution une fois l'exécution d'une fonction Lambda terminée, et le débloque si AWS Lambda choisit de le réutiliser lorsque la fonction Lambda est de nouveau appelée. Cette approche de réutilisation du contexte d'exécution présente les conséquences suivantes :

- Toutes les déclarations du code de votre fonction Lambda (en dehors du code `handler`, cf. [Modèle de programmation \(p. 24\)](#)) restent initialisées et fournissent ainsi une optimisation supplémentaire lorsque la fonction est appelée à nouveau. Par exemple, si la fonction Lambda établit une connexion de base de données, au lieu de rétablir la connexion, la connexion d'origine est utilisée dans les appels suivants. Vous pouvez ajouter une logique dans le code pour vérifier s'il existe une connexion avant d'en créer une.
- Chaque contexte d'exécution fournit 500 Mo d'espace disque supplémentaire dans le répertoire `/tmp`. Le contenu du répertoire est préservé lorsque le contexte d'exécution est gelé, fournissant ainsi un cache temporaire qui peut servir à plusieurs appels. Vous pouvez ajouter du code pour vérifier si le cache contient les données que vous avez stockées. Pour en savoir plus sur les limites appliquées aux déploiements, consultez [Limites AWS Lambda \(p. 288\)](#).
- Les processus en arrière-plan ou les rappels initiés par la fonction Lambda qui ne se terminent pas lorsque la fonction prend fin reprennent si AWS Lambda choisit de réutiliser le contexte d'exécution. Assurez-vous que les processus d'arrière-plan ou les rappels (en cas d'utilisation de Node.js) dans votre code se terminent avant que l'exécution du code ne prenne fin.

Lorsque vous écrivez le code de la fonction Lambda, ne partez pas du principe qu'AWS Lambda réutilise automatiquement le contexte d'exécution pour les appels de fonction suivants. D'autres facteurs peuvent obliger AWS Lambda à créer un nouveau contexte d'exécution, ce qui peut entraîner des résultats inattendus, tels que des échecs de connexion à la base de données.

stratégie de prise en charge de l'exécution

AWS Lambda rendra uniquement obsolètes les versions d'exécution qui sont marquées comme EOL (End of Life, fin de vie) à la fin de leur fenêtre de maintenance. Les versions marquées comme étant obsolètes cesseront de prendre en charge la création de nouvelles fonctions et les mises à jour de fonctions existantes créées dans l'exécution obsolète (sauf en cas de reconfiguration pour utiliser une version d'exécution prise en charge). AWS Lambda ne fournira pas non plus de mises à jour de sécurité, de support technique ou de correctifs logiciels pour les versions d'exécution obsolètes et se réserve le droit de désactiver les appels des fonctions configurées pour s'exécuter sur une exécution obsolète à tout moment.

Runtimes Node.js obsolètes

Nom	Identificateur	Fin de vie
Node.js 0.10	<code>nodejs</code>	31/10/2016

Nom	Identificateur	Fin de vie
Node.js 4.3	nodejs4.3 nodejs4.3-edge	30/04/2018

Runtimes AWS Lambda personnalisés

Vous pouvez implémenter un runtime AWS Lambda dans n'importe quel langage de programmation. Un runtime est un programme qui exécute la méthode de gestionnaire d'une fonction Lambda lorsque la fonction est appelée. Vous pouvez inclure un runtime dans le package de déploiement de votre fonction sous la forme d'un fichier exécutable nommé `bootstrap`.

Le runtime est responsable de l'exécution du code de configuration de la fonction, de la lecture du nom du gestionnaire dans la variable d'environnement et de la lecture des événements d'appels dans l'API de runtime de Lambda. Le runtime transmet les données d'événements au gestionnaire de la fonction et renvoie la réponse du gestionnaire à Lambda.

Votre runtime personnalisé s'exécute dans Lambda et, plus précisément, dans son [environnement d'exécution standard \(p. 104\)](#). Il peut s'agir d'un script shell, d'un script dans un langage qui est inclus dans Amazon Linux ou d'un fichier exécutable binaire compilé dans Amazon Linux.

Pour commencer à avec les runtimes personnalisés, consultez [Didacticiel – Publication d'un runtime personnalisé \(p. 111\)](#). Vous pouvez également explorer un runtime personnalisé implémenté en C++ à l'adresse [awslabs/aws-lambda-cpp](#) sur GitHub.

Rubriques

- [Utilisation d'un runtime personnalisé \(p. 107\)](#)
- [Création d'un runtime personnalisé \(p. 107\)](#)

Utilisation d'un runtime personnalisé

Pour utiliser un runtime personnalisé, définissez le runtime de votre fonction sur `provided`. Le runtime peut être inclus dans le package de déploiement de votre fonction ou dans une [couche \(p. 56\)](#).

Example function.zip

```
.\n### bootstrap\n### function.sh
```

S'il y a un fichier nommé `bootstrap` dans votre package de déploiement, Lambda exécute ce fichier. Dans le cas contraire, Lambda recherche un runtime dans les couches de la fonction. Si le fichier d'amorçage est introuvable ou n'est pas exécutable, votre fonction renvoie une erreur au moment de l'appel.

Création d'un runtime personnalisé

Le point d'entrée d'un runtime personnalisé est un fichier exécutable nommé `bootstrap`. Le fichier d'amorçage peut être le runtime, ou il peut appeler un autre fichier qui crée le runtime. L'exemple suivant utilise une version groupée de Node.js pour exécuter un runtime JavaScript dans un fichier séparé nommé `runtime.js`.

Example amorçage

```
#!/bin/sh
cd $LAMBDA_TASK_ROOT
./node-v11.1.0-linux-x64/bin/node runtime.js
```

Le code de votre runtime est responsable de l'exécution de certaines tâches d'initialisation. Ensuite, il traite les événements d'appels dans une boucle jusqu'à ce qu'à son arrêt. Les tâches d'initialisation sont exécutées une seule fois [par instance de la fonction \(p. 105\)](#) pour préparer l'environnement à la gestion des appels.

Tâches d'initialisation

- Récupérer les paramètres – Lecture des variables de l'environnement pour obtenir les détails sur la fonction et l'environnement.
- `_HANDLER` – Emplacement du gestionnaire, issu de la configuration de la fonction. Le format standard est `file.method`, où `file` est le nom du fichier sans extension et `method` est le nom d'une méthode ou fonction qui est définie dans le fichier.
- `LAMBDA_TASK_ROOT` – Répertoire qui contient le code de la fonction.
- `AWS_LAMBDA_RUNTIME_API` – Hôte et port de l'API du runtime.

Pour obtenir une liste complète des variables disponibles, consultez [Variables d'environnement disponibles pour les fonctions Lambda \(p. 104\)](#).

- Initialiser la fonction – Chargez le fichier de gestionnaire et exécutez tout code global ou statique qu'il contient. Les fonctions doivent créer des ressources statiques telles que des clients de kit SDK et des connexions de base de données une seule fois, puis les réutiliser pour plusieurs appels.
- Gérer les erreurs – Si une erreur se produit,appelez l'API [erreur d'initialisation \(p. 110\)](#) et quittez immédiatement.

Pendant son exécution, le runtime utilise l'[interface de runtime Lambda \(p. 109\)](#) pour gérer les événements entrants et signaler des erreurs. Après avoir terminé les tâches d'initialisation, le runtime traite les événements entrants dans une boucle.

Traitement des tâches

- Obtenir un événement – Appelez l'API [prochain appel \(p. 109\)](#) pour obtenir l'événement suivant. Le corps de la réponse contient les données de l'événement. Les en-têtes de la réponse contiennent l'ID de la demande et d'autres informations.
- Propager l'en-tête de suivi – Obtenez l'en-tête de suivi X-Ray à partir de l'en-tête `Lambda-Runtime-Trace-Id` dans la réponse de l'API. Définissez la variable d'environnement `_X_AMZN_TRACE_ID` avec la même valeur pour le kit SDK X-Ray à utiliser.
- Créer un objet de contexte – Créez un objet avec les informations de contexte à partir des variables d'environnement et les en-têtes de la réponse de l'API.
- Appeler le gestionnaire de fonctions – Transmettez l'événement et l'objet de contexte au gestionnaire.
- Gérer la réponse – Appelez l'API [réponse d'appel \(p. 110\)](#) pour afficher la réponse du gestionnaire.
- Gérer les erreurs – Si une erreur se produit,appelez l'API [erreur d'appel \(p. 110\)](#).
- Nettoyage – Libérez les ressources inutilisées, envoyez des données à d'autres services ou réalisez des tâches supplémentaires avant de passer à l'événement suivant.

Vous pouvez inclure le runtime dans le package de déploiement de votre fonction ou distribuer le runtime séparément dans une couche de fonction. Pour afficher un exemple, consultez [???](#) (p. 111).

Interface de runtime AWS Lambda

AWS Lambda fournit une API HTTP pour les [runtimes personnalisés \(p. 107\)](#) afin de recevoir les événements d'appels provenant de Lambda et d'envoyer des données de réponses au sein de Lambda et plus précisément, de l'[environnement d'exécution \(p. 104\)](#).

La spécification OpenAPI pour la version d'API de runtime 2018-06-01 est disponible ici : [runtime-api.zip](#)

Les runtimes obtiennent un point de terminaison de la variable d'environnement `AWS_LAMBDA_RUNTIME_API`, ajoutent la version de l'API et utilisent les chemins d'accès de ressources ci-après pour interagir avec l'API.

Example Requête

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Ressources

- [Appel suivant \(p. 109\)](#)
- [Réponse d'appel \(p. 110\)](#)
- [Erreur d'appel \(p. 110\)](#)
- [Erreur d'initialisation \(p. 110\)](#)

Appel suivant

Chemin d'accès – `/runtime/invocation/next`

Méthode – GET

Extrait un événement d'appel. Le corps de la réponse contient la charge utile provenant de l'appel, qui est un document JSON contenant les données d'événements du déclencheur de la fonction. Les en-têtes de la réponse contiennent des données supplémentaires sur l'appel.

En-têtes de réponse

- `Lambda-Runtime-Aws-Request-Id` – L'ID de demande, qui identifie la demande ayant déclenché l'appel de la fonction.

Par exemple, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.

- `Lambda-Runtime-Deadline-Ms` – Date à laquelle la fonction expire, exprimée en millisecondes au format horaire Unix.

Par exemple, `1542409706888`.

- `Lambda-Runtime-Invoked-Function-Arn` – ARN de la fonction Lambda, la version ou l'alias spécifié dans l'appel.

Par exemple, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.

- `Lambda-Runtime-Trace-Id` – [En-tête de suivi AWS X-Ray](#).

Par exemple, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.

- `Lambda-Runtime-Client-Context` – Pour les appels à partir du Kit SDK AWS Mobile, données sur l'application cliente et le périphérique.

- `Lambda-Runtime-Cognito-Identity` – Pour les appels à partir du Kit SDK AWS Mobile, données sur le fournisseur d'identités Amazon Cognito.

L'ID de demande permet de suivre l'appel au sein de Lambda. Utilisez-le pour spécifier l'appel lorsque vous envoyez la réponse.

L'en-tête de suivi contient l'ID de suivi, l'ID parent et la décision d'échantillonnage. Si la demande est échantillonnée, elle a été échantillonnée par Lambda ou par un service en amont. Le runtime doit définir l'`_X_AMZN_TRACE_ID` avec la valeur de l'en-tête. Le kit SDK X-Ray lit ceci pour obtenir les ID et déterminer si, oui ou non, suivre la demande.

Réponse d'appel

Chemin d'accès – `/runtime/invocation/AwsRequestId/response`

Méthode – POST

Envoie une réponse d'appel à Lambda. Une fois que le runtime a appelé le gestionnaire de fonctions, il publie la réponse de la fonction dans le chemin d'accès de la réponse d'appel. Pour les appels synchrones, Lambda renvoie alors la réponse au client.

Example Demande d'opération réussie

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "SUCCESS"
```

Erreur d'appel

Chemin d'accès – `/runtime/invocation/AwsRequestId/error`

Méthode – POST

Si la fonction renvoie une erreur, le runtime met en forme l'erreur dans un document JSON et le publie dans le chemin d'accès de l'erreur d'appel.

Example Corps de la demande

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException"
}
```

Example Demande d'erreur

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
ERROR={"\\"errorMessage\"\\" : \\"Error parsing event data.\\"", \\"errorType\\\"
: \\"InvalidEventDataException\\\"}"
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
error" -d "$ERROR"
```

Erreur d'initialisation

Chemin d'accès – `/runtime/init/error`

Méthode – POST

Si le runtime rencontre une erreur lors de l'initialisation, il publie un message d'erreur dans le chemin d'accès de l'erreur d'initialisation.

Example Demande d'erreur

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
#ERROR={"errorMessage": "Failed to load function.\\"", "errorType": "
  \"InvalidFunctionException\"}"
#curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR"
```

Didacticiel – Publication d'un runtime personnalisé

Dans ce didacticiel, vous allez créer une fonction Lambda avec un runtime personnalisé. Vous commencez par inclure le runtime dans le package de déploiement de la fonction. Ensuite, vous le migrez vers une couche que vous gérez indépendamment de la fonction. Enfin, vous partagez la couche du runtime en mettant à jour sa stratégie d'autorisations basée sur les ressources.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

You need an IAM role to create a Lambda function. The role needs permission to send logs to CloudWatch Logs and access the AWS services that your function uses. If you don't have a role for function development, create one now.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Autorisations – AWSLambdaBasicExecutionRole.
 - Nom de rôle – **lambda-role**.

La stratégie AWSLambdaBasicExecutionRole possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Créer une fonction

Créez une fonction Lambda avec un runtime personnalisé. Cet exemple comprend deux fichiers, un fichier `bootstrap` de runtime et un gestionnaire de fonctions. Tous deux sont mis en œuvre en Bash.

Le runtime charge un script de fonction à partir du package de déploiement. Il utilise deux variables pour localiser le script. `LAMBDA_TASK_ROOT` lui indique où le package a été extrait et `_HANDLER` inclut le nom du script.

Example amorçage

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event
    EVENT_DATA=$(curl -ss -LD "$HEADERS" -X GET "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")
    REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d '[:space:]' | cut -d: -f2)

    # Execute the handler function from the script
    RESPONSE=$(($(_HANDLER" | cut -d. -f2) "$EVENT_DATA"))

    # Send the response
    curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/response" -d "$RESPONSE"
done
```

Après avoir chargé le script, le runtime traite les événements dans une boucle. Il utilise l'API du runtime pour récupérer un événement d'appel dans Lambda, transmet l'événement au gestionnaire, puis renvoie la réponse à Lambda. Pour obtenir l'ID de la demande, le runtime enregistre les en-têtes à partir de la réponse de l'API dans un fichier temporaire et lit l'en-tête `Lambda-Runtime-Aws-Request-Id` à partir du fichier.

Note

Les runtimes ont d'autres responsabilités, notamment la gestion des erreurs et la fourniture d'informations de contexte au gestionnaire. Pour plus d'informations, consultez [Création d'un runtime personnalisé \(p. 107\)](#).

Le script définit une fonction de gestionnaire qui accepte les données des événements, la consigne dans `stderr`, puis la renvoie.

Example function.sh

```
function handler () {
    EVENT_DATA=$1
    echo "$EVENT_DATA" 1>&2;
    RESPONSE="Echoing request: '$EVENT_DATA'

    echo $RESPONSE
}
```

Enregistrez les deux fichiers dans un répertoire de projet nommé `runtime-tutorial`.

```
runtime-tutorial
# bootstrap
# function.sh
```

Rendez les fichiers exécutables et ajoutez-les dans une archive ZIP.

```
runtime-tutorial$ chmod 755 function.sh bootstrap
runtime-tutorial$ zip function.zip function.sh bootstrap
adding: function.sh (deflated 24%)
adding: bootstrap (deflated 39%)
```

Créez une fonction nommée bash-runtime.

```
runtime-tutorial$ aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime provided \
--role arn:aws:iam::123456789012:role/lambda-role
{
    "FunctionName": "bash-runtime",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:bash-runtime",
    "Runtime": "provided",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 831,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-28T06:57:31.095+0000",
    "CodeSha256": "mv/xRv84LPCxdpcbKvmwuuFzwo7sLwUO1VxcUv3wKlM=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "2e1d51b0-6144-4763-8e5c-7d5672a01713"
}
```

Appelez la fonction et vérifiez la réponse.

```
runtime-tutorial$ aws lambda invoke --function-name bash-runtime --payload
'{"text":"Hello"}' response.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
runtime-tutorial$ cat response.txt
Echoing request: '{"text":"Hello"}'
```

Créer une couche

Pour séparer le code du runtime du code de la fonction, créez une couche qui contient uniquement le runtime. Les couches vous permettent de développer les dépendances de votre fonction de manière indépendante et peuvent réduire l'utilisation du stockage lorsque vous utilisez la même couche avec plusieurs fonctions.

Créez une archive de couche qui contient le fichier bootstrap.

```
runtime-tutorial$ zip runtime.zip bootstrap
adding: bootstrap (deflated 39%)
```

Créez une couche à l'aide de la commande publish-layer-version.

```
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
fileb://runtime.zip
```

```
{  
    "Content": {  
        "Location": "https://awslambda-us-west-2-layers.s3.us-west-2.amazonaws.com/  
snapshots/123456789012/bash-runtime-018c209b...",  
        "CodeSha256": "bXVLhHi+D3H1QbDARUVPrDwlC7bssPxySQqt1QZqusE=",  
        "CodeSize": 584,  
        "UncompressedCodeSize": 0  
    },  
    "LayerArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime",  
    "LayerVersionArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1",  
    "Description": "",  
    "CreatedDate": "2018-11-28T07:49:14.476+0000",  
    "Version": 1  
}
```

Cela crée la première version de la couche.

Mise à jour de la fonction

Pour utiliser la couche de runtime avec la fonction, configurez la fonction pour utiliser la couche et supprimez le code du runtime de la fonction.

Mettez à jour la configuration de la fonction pour extraire la couche.

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \  
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1  
{  
    "FunctionName": "bash-runtime",  
    "Layers": [  
        {  
            "Arn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1",  
            "CodeSize": 584,  
            "UncompressedCodeSize": 679  
        }  
    ]  
    ...  
}
```

Cela ajoute le runtime à la fonction dans le répertoire /opt. Lambda utilise ce runtime, mais uniquement si vous le supprimez du package de déploiement de la fonction. Mettez à jour le code de la fonction de façon à inclure uniquement le script du gestionnaire.

```
runtime-tutorial$ zip function-only.zip function.sh  
adding: function.sh (deflated 24%)  
runtime-tutorial$ aws lambda update-function-code --function-name bash-runtime --zip-file  
fileb://function-only.zip  
{  
    "FunctionName": "bash-runtime",  
    "CodeSize": 270,  
    "Layers": [  
        {  
            "Arn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:7",  
            "CodeSize": 584,  
            "UncompressedCodeSize": 679  
        }  
    ]  
    ...  
}
```

Appelez la fonction pour vérifier qu'elle fonctionne avec la couche du runtime.

```
runtime-tutorial$ aws lambda invoke --function-name bash-runtime --payload
'{"text":"Hello"})' response.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
runtime-tutorial$ cat response.txt
Echoing request: '{"text":"Hello")'
```

Mise à jour du runtime

Pour enregistrer des informations sur l'environnement d'exécution, mettez à jour le script du runtime pour générer les variables d'environnement.

Example amorçage

```
#!/bin/sh

set -euo pipefail

echo "## Environment variables:"
env

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"
...
```

Créez une deuxième version de la couche avec le nouveau code.

```
runtime-tutorial$ zip runtime.zip bootstrap
updating: bootstrap (deflated 39%)
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
file:///runtime.zip
```

Configurez la fonction pour utiliser la nouvelle version de la couche.

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:2
```

Partage de la couche

Ajoutez une instruction d'autorisation dans la couche de votre runtime afin de la partager avec d'autres comptes.

```
runtime-tutorial$ aws lambda add-layer-version-permission --layer-name bash-runtime --
version-number 2 \
--principal "*" --statement-id publish --action lambda:GetLayerVersion
{
    "Statement": "{\"Sid\":\"publish\",\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\":
\"lambda:GetLayerVersion\", \"Resource\":\"arn:aws:lambda:us-west-2:123456789012:layer:bash-
runtime:2\"}",
    "RevisionId": "9d5fe08e-2a1e-4981-b783-37ab551247ff"
}
```

Vous pouvez ajouter plusieurs instructions qui accordent, chacune, une autorisation à un compte unique, aux comptes d'une organisation ou à tous les comptes.

Nettoyage

Supprimez chaque version de la couche.

```
runtime-tutorial$ aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
runtime-tutorial$ aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Étant donné que la fonction contient une référence à la version 2 de la couche, elle existe toujours dans Lambda. La fonction continue de fonctionner, mais les fonctions ne peuvent plus être configurées pour utiliser la version supprimée. Si vous modifiez ensuite la liste des couches sur la fonction, vous devez spécifier une nouvelle version ou omettre la couche supprimée.

Supprimez la fonction de didacticiel à l'aide de la commande `delete-function`.

```
runtime-tutorial$ aws lambda delete-function --function-name bash-runtime
```

Applications AWS Lambda

Une application AWS Lambda est une combinaison de fonctions Lambda, de sources d'événements et d'autres ressources qui fonctionnent ensemble pour effectuer des tâches. Vous pouvez utiliser AWS CloudFormation et d'autres outils pour collecter les composants de votre application dans un seul package pouvant être déployé et géré comme une seule ressource. Les applications rendent vos projets Lambda mobiles et vous permettent d'intégrer des outils de développement supplémentaires, tels qu'AWS CodePipeline, AWS CodeBuild et l'interface de ligne de commande Modèle d'application sans serveur AWS (interface de ligne de commande SAM).

Le [AWS Serverless Application Repository](#) fournit un ensemble d'applications Lambda que vous pouvez déployer dans votre compte en quelques clics. Le référentiel inclut des applications et des exemples prêts à l'emploi que vous pouvez utiliser comme point de départ pour vos propres projets. Vous pouvez également soumettre vos propres projets pour l'inclusion.

[AWS CloudFormation](#) vous permet de créer un modèle qui définit les ressources de votre application et vous permet de gérer l'application en tant que pile. Vous pouvez ajouter ou modifier des ressources de manière plus sûre dans votre pile d'applications. Si une partie quelconque d'une mise à jour échoue, AWS CloudFormation restaure automatiquement la configuration précédente. Avec les paramètres AWS CloudFormation, vous pouvez créer plusieurs environnements pour votre application à partir du même modèle.

Le [Modèle d'application sans serveur AWS \(p. 120\)](#) (AWS SAM) est une extension du langage de modèle AWS CloudFormation qui vous permet de définir des applications sans serveur à un niveau plus élevé. Il fait abstraction des tâches courantes telles que la création de rôle de fonction, ce qui facilite l'écriture de modèles. AWS SAM est pris en charge directement par AWS CloudFormation et inclut des fonctionnalités supplémentaires via l'AWS CLI et l'interface de ligne de commande AWS SAM.

L'[AWS CLI](#) et l'[interface de ligne de commande AWS SAM](#) sont des outils de ligne de commande pour la gestion des piles d'applications Lambda. En plus des commandes pour la gestion des piles d'applications avec l'API AWS CloudFormation, l'AWS CLI prend en charge des commandes de plus haut niveau qui simplifient les tâches telles que le chargement de packages de déploiement et la mise à jour de modèles. L'interface de ligne de commande AWS SAM fournit des fonctionnalités supplémentaires, y compris la validation des modèles et les tests locaux.

Rubriques

- [Gestion des applications dans la console AWS Lambda \(p. 117\)](#)
- [Utilisation du modèle d'application sans serveur AWS \(AWS SAM\) \(p. 120\)](#)
- [Création d'un pipeline pour votre application sans serveur \(p. 120\)](#)

Gestion des applications dans la console AWS Lambda

La console AWS Lambda vous permet de surveiller et de gérer vos [applications Lambda \(p. 117\)](#). Le menu Applications répertorie les piles AWS CloudFormation avec des fonctions Lambda. Le menu inclut les piles que vous lancez dans AWS CloudFormation à l'aide de la console AWS CloudFormation, du AWS Serverless Application Repository, de l'AWS CLI ou de l'interface de ligne de commande AWS SAM.

Pour afficher une application Lambda

1. Open the [Lambda console](#).
2. Choose Applications.

3. Choisissez une application.

Name	Description	Last modified	Status
scorekeep-random-name	Create a Lambda function with permission to use SNS and X-Ray	20 days ago	UPDATE_COMPLETE
lambda-web-page	Set up API Gateway endpoint connected to Lambda function that returns HTML	23 days ago	CREATE_COMPLETE

La présentation affiche les informations suivantes sur votre application.

- Modèle AWS CloudFormation ou modèle SAM – Le modèle qui définit votre application.
- Ressources – Les ressources AWS qui sont définies dans le modèle de votre application. Pour gérer les fonctions Lambda de votre application, choisissez un nom de fonction dans la liste.

Surveillance des applications

L'onglet Surveillance affiche un tableau de bord Amazon CloudWatch avec des mesures agrégées pour les ressources de votre application.

Pour surveiller une application Lambda

1. Open the [Lambda console](#).
2. Choose Applications.
3. Choisissez Surveillance.

Par défaut, la console Lambda affiche un tableau de bord de base. Vous pouvez personnaliser cette page en définissant des tableaux de bord personnalisés dans votre modèle d'application. Lorsque votre modèle inclut un ou plusieurs tableaux de bord, la page affiche vos tableaux de bord à la place du tableau de bord par défaut. Vous pouvez basculer entre les tableaux de bord avec le menu déroulant en haut à droite de la page.

Tableaux de bord de surveillance personnalisés

Personnalisez votre page de surveillance d'applications en ajoutant un ou plusieurs tableaux de bord Amazon CloudWatch à votre modèle d'application avec le type de ressource [AWS::CloudWatch::Dashboard](#). L'exemple suivant permet de créer un tableau de bord avec un seul widget qui représente graphiquement le nombre d'appels d'une fonction nommée `my-function`.

Example Modèle de tableau de bord de fonction

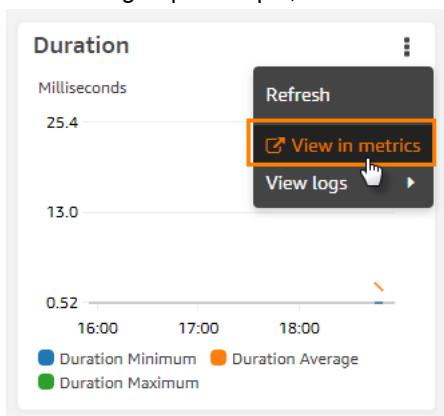
```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
          "widgets": [
            {
              "type": "metric",
              "name": "my-function-metric",
              "x": 100,
              "y": 100,
              "width": 200,
              "height": 100,
              "region": "us-east-1"
            }
          ]
        }
```

```
"width": 12,
"height": 6,
"properties": {
    "metrics": [
        [
            "AWS/Lambda",
            "Invocations",
            "FunctionName",
            "my-function",
            {
                "stat": "Sum",
                "label": "MyFunction"
            }
        ],
        [
            {
                "expression": "SUM(METRICS())",
                "label": "Total Invocations"
            }
        ]
    ],
    "region": "us-east-1",
    "title": "Invocations",
    "view": "timeSeries",
    "stacked": false
}
```

Vous pouvez obtenir la définition pour un widget quelconque dans le tableau de bord de surveillance par défaut à partir de la console CloudWatch.

Pour afficher la définition d'un widget

1. Open the [Lambda console](#).
2. Choose Applications.
3. Choisissez une application qui possède le tableau de bord standard.
4. Choisissez Surveillance.
5. Sur un widget quelconque, choisissez Afficher dans les métriques dans le menu déroulant.



6. Choisissez Source.

Pour plus d'informations sur la création des widgets et des tableaux de bord CloudWatch, consultez [Syntaxe et structure du corps d'un tableau de bord](#) dans la référence de l'API Amazon CloudWatch.

Utilisation du modèle d'application sans serveur AWS (AWS SAM)

Le modèle Modèle d'application sans serveur AWS (AWS SAM) est une infrastructure open source que vous pouvez utiliser pour construire des applications sans serveur sur AWS. Il se compose de la spécification de modèle AWS SAM que vous utilisez pour définir vos applications sans serveur et de l'interface de ligne de commande AWS SAM (CLI AWS SAM) que vous utilisez pour générer, tester et déployer vos applications sans serveur.

Pour plus d'informations sur AWS SAM, consultez [Qu'est-ce qu'AWS SAM ?](#).

Pour plus d'informations sur les modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#).

Pour plus d'informations sur les tests locaux des fonctions Lambda, consultez [Test et débogage des applications sans serveur](#).

Pour plus d'informations sur l'automatisation des déploiements d'applications sans serveur, consultez [Déploiement d'applications sans serveur](#).

Création d'un pipeline pour votre application sans serveur

Dans le didacticiel suivant, vous allez créer un AWS CodePipeline qui automatisé le déploiement de votre application sans serveur. Tout d'abord, vous devez mettre en place une étape source pour déclencher votre pipeline. Dans le cadre de didacticiel :

- Nous utilisons GitHub. Pour plus d'informations sur la création d'un référentiel GitHub, consultez [Création d'un référentiel dans GitHub](#).
- Vous devez créer un rôle AWS CloudFormation et ajouter la stratégie AWSLambdaExecute à ce rôle, comme indiqué dans les étapes suivantes :
 1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
 2. Pour créer un rôle IAM (rôle d'exécution), suivez les étapes fournies dans [Création d'un rôle pour la délégation d'autorisations à un service AWS](#), dans le IAM Guide de l'utilisateur. Ensuite, consultez la section Pour créer un rôle pour un service AWS. Lors de la création d'un rôle, prenez note des points suivants :
 - Dans Select Role Type (Sélectionner un type de rôle), sélectionnez AWS Service Roles (Rôles de service AWS), puis choisissez CloudFormation. Sélectionnez Étape suivante : autorisations.
 - Dans Attach permissions policies (Attacher les stratégies d'autorisation), utilisez la barre de recherche pour trouver AWSLambdaExecute et le sélectionner. Choisissez Next: Review.
 - Dans Nom du rôle, indiquez un nom unique dans votre compte AWS (par exemple, cloudformation-lambda-execution-role), puis choisissez Créer un rôle.
 - Ouvrez le rôle que vous venez de créer. Sous l'onglet Permissions (Autorisations), choisissez Add inline policy (Ajouter une stratégie en ligne).
 - Dans Créer une stratégie, choisissez l'onglet JSON et entrez ce qui suit :

Note

Veillez à substituer votre région et votre ID de compte aux espaces réservés *région* et *id*.

```
{  
  "Statement": [  
    {  
      "Action": [  
        "s3:GetObject",  
        "s3:GetObjectVersion",  
        "s3:GetBucketVersioning"  
      ],  
      "Resource": "*",  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "s3:PutObject"  
      ],  
      "Resource": [  
        "arn:aws:s3:::codepipeline*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "lambda:*"  
      ],  
      "Resource": [  
        "arn:aws:lambda:region:id:function:/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "apigateway:/*"  
      ],  
      "Resource": [  
        "arn:aws:apigateway:region::/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "iam:GetRole",  
        "iam>CreateRole",  
        "iam>DeleteRole",  
        "iam:PutRolePolicy"  
      ],  
      "Resource": [  
        "arn:aws:iam::id:role/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "iam:AttachRolePolicy",  
        "iam:DeleteRolePolicy",  
        "iam:DetachRolePolicy"  
      ],  
      "Resource": [  
        "arn:aws:iam::id:role/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "iam:PassRole"  
      ]  
    }  
  ]  
}
```

```
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "cloudformation>CreateChangeSet"
        ],
        "Resource": [
            "arn:aws:cloudformation:region:aws:transform/Serverless-2016-10-31"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "codedeploy>CreateApplication",
            "codedeploy>DeleteApplication",
            "codedeploy:RegisterApplicationRevision"
        ],
        "Resource": [
            "arn:aws:codedeploy:region:id:application:/*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "codedeploy>CreateDeploymentGroup",
            "codedeploy>CreateDeployment",
            "codedeploy:GetDeployment"
        ],
        "Resource": [
            "arn:aws:codedeploy:region:id:deploymentgroup:/*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "codedeploy:GetDeploymentConfig"
        ],
        "Resource": [
            "arn:aws:codedeploy:region:id:deploymentconfig:/*"
        ],
        "Effect": "Allow"
    }
],
"Version": "2012-10-17"
}
```

- Choisissez Valider la stratégie, puis Appliquer la stratégie.

Étape 1 : Configuration de votre référentiel

Vous pouvez utiliser n'importe quel runtime pris en charge par Lambda lorsque vous configurez un référentiel. L'exemple suivant utilise Node.js.

Pour configurer votre référentiel, procédez comme suit :

- Ajoutez un fichier index.js contenant le code suivant :

```
var time = require('time');
```

```
exports.handler = (event, context, callback) => {
    var currentTime = new Date();
    currentTime.settimezone("America/Los_Angeles");
    callback(null, {
        statusCode: '200',
        body: 'The time in Los Angeles is: ' + currentTime.toString(),
    });
};
```

- Ajoutez un fichier `samTemplate.yaml` présentant le contenu suivant. Il s'agit du modèle AWS SAM qui définit les ressources dans votre application. Ce modèle AWS SAM définit une fonction Lambda déclenchée par API Gateway. Notez que le paramètre `runtime` utilise `nodejs6.10`, mais que vous pouvez également spécifier `nodejs8.10`. Pour plus d'informations sur AWS SAM, consultez [Modèle d'application sans serveur AWS](#).

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Outputs the time
Resources:
  TimeFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      CodeUri: .
    Events:
      MyTimeApi:
        Type: Api
        Properties:
          Path: /TimeResource
          Method: GET
```

- Ajoutez un fichier `buildspec.yml`. Une spécification de génération est un ensemble de commandes de génération et de paramètres associés, au format YAML, qu'AWS CodeBuild utilise pour exécuter une génération. Pour plus d'informations, consultez [Référence des spécifications de génération pour AWS CodeBuild](#). Dans cet exemple, l'action de génération est la suivante :
 - Utilisez `npm` pour installer le package de durée.
 - Exécutez la commande `Package` pour préparer votre package de déploiement pour les étapes de déploiement suivantes dans votre pipeline. Pour plus d'informations sur la commande de mise en package, consultez [Chargement d'artefacts locaux dans un compartiment S3](#).

```
version: 0.2
phases:
  install:
    commands:
      - npm install time
      - aws cloudformation package --template-file samTemplate.yaml --kms-key-id kms-key-id --s3-bucket bucket-name --output-template-file outputSamTemplate.yaml
  artifacts:
    type: zip
    files:
      - samTemplate.yaml
      - outputSamTemplate.yaml
```

Étape 2 : Créez votre pipeline

Exécutez la procédure ci-dessous pour créer votre instance AWS CodePipeline.

1. Connectez-vous à AWS Management Console et ouvrez la console AWS CodePipeline.
2. Sélectionnez Pour commencer.
3. Dans Nom du pipeline, entrez un nom pour votre pipeline, puis choisissez Étape suivante.
4. Dans Fournisseur de source, choisissez GitHub.
5. Choisissez Se connecter à GitHub, puis choisissez le référentiel et la branche auxquels vous souhaitez vous connecter. Chaque git push sur la branche que vous sélectionnez déclenche votre pipeline. Choisissez Next step.
6. Choisissez AWS CodeBuild comme Fournisseur de génération.
7. Choisissez Create a new build project et entrez un nom de projet.
8. Choisissez Ubuntu comme système d'exploitation.
9. Choisissez Node.js en tant qu'environnement d'exécution.
10. Dans Version, choisissez aws/codebuild/nodejs:*version*.
11. Dans Spécification de génération, choisissez Use the buildspec.yml in the source code root directory.
12. Choisissez Save build project.

Note

Un rôle de service pour AWS CodeBuild est créé automatiquement en votre nom.

Choisissez Next step.

13. Dans Fournisseur du déploiement, choisissez AWS CloudFormation.

Lorsque vous choisissez cette option, les commandes AWS CloudFormation sont utilisées pour déployer le modèle AWS SAM. Pour plus d'informations, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

14. Dans Mode d'action, choisissez Créer ou remplacer un jeu de modifications.
15. Dans Nom de la pile, entrez MyBetaStack.
16. Dans Nom du jeu de modifications, entrez MyChangeSet.
17. Dans Fichier de modèle, entrez outputSamTemplate.yaml.
18. Dans Capacités, choisissez CAPABILITY_IAM.
19. Dans Rôle, sélectionnez le rôle AWS CloudFormation que vous avez créé au début de ce didacticiel, puis choisissez Étape suivante.
20. Choisissez Créer un rôle, Suivant, Autoriser, puis Étape suivante.
21. Passez en revue votre pipeline, puis choisissez Create pipeline.

Étape 3 : Mettre à jour la stratégie de service générée

Exécutez les étapes ci-après pour autoriser AWS CodeBuild à charger des artefacts de génération dans votre compartiment Amazon S3.

1. Accédez à la console IAM.
2. Sélectionnez Roles.
3. Ouvrez le rôle de service généré pour votre projet. Il s'agit généralement de code-build-*nom-du-projet*-service-role.

4. Sous l'onglet Permissions (Autorisations), choisissez Add inline policy (Ajouter une stratégie en ligne).
5. Dans service, choisissez Choose a service (Choisir un service).
6. Dans Select a service below (Sélectionner un service ci-dessous), choisissez S3.
7. Dans Actions, choisissez Select actions (Sélectionner des actions).
8. Développez Write (Écrire) sous Access level groups (Groupes de niveaux d'accès), puis choisissez PutObject.
9. Choisissez Resources (Ressources), puis cochez la case Any (Tout).
10. Choisissez Examiner une stratégie.
11. Entrez un nom, puis choisissez Créer une stratégie. Ensuite, revenez au pipeline que vous avez créé dans la section précédente.

Étape 4 : Exécuter votre étape de déploiement bêta

Utilisez les étapes suivantes pour terminer votre étape bêta.

1. Choisissez Edit.
2. Choisissez l'icône + en regard de MyBetaStack.
3. Dans Catégorie d'action, si cette option n'est pas déjà sélectionnée, choisissez Déployer.
4. Dans Fournisseur du déploiement, si cette option n'est pas déjà sélectionnée, choisissez AWS CloudFormation.
5. Dans Mode d'action, choisissez Exécuter un jeu de modifications. CreateChangeSet transforme le modèle AWS SAM au format AWS CloudFormation complet et deployChangeSet déploie le modèle AWS CloudFormation.
6. Dans Nom de la pile, entrez ou choisissez MyBetaStack.
7. Dans Nom du jeu de modifications, entrez MyChangeSet.
8. Choisissez Add action.
9. Choisissez Save pipeline changes.
10. Choisissez Save and continue.

Votre pipeline est prêt. Tout `git push` sur la branche que vous avez connectée à ce pipeline déclenchera un déploiement. Pour tester votre pipeline et déployer votre application pour la première fois, effectuez l'une des actions suivantes :

- Exédez un `git push` sur la branche connectée à votre pipeline.
- Accédez à la console AWS CodePipeline, choisissez le nom du pipeline que vous avez créé, puis choisissez Publier une modification.

Exemples d'utilisation d'AWS Lambda

Les cas d'utilisation d'AWS Lambda peuvent être regroupés dans les catégories suivantes :

- Utilisation d'AWS Lambda avec les services AWS en tant que sources d'événements – Les sources d'événements publient des événements qui entraînent l'appel de la fonction Lambda. Il peut s'agir de services AWS comme Amazon S3. Pour plus d'informations et de didacticiels, consultez les rubriques suivantes :

[Utilisation de AWS Lambda avec Amazon S3 \(p. 126\)](#)

[Utilisation de AWS Lambda avec Amazon Kinesis \(p. 142\)](#)

[Utilisation de AWS Lambda avec Amazon SQS \(p. 153\)](#)

[Utilisation de AWS Lambda avec Amazon DynamoDB \(p. 162\)](#)

[Utilisation de AWS Lambda avec AWS CloudTrail \(p. 173\)](#)

[Utilisation d'AWS Lambda avec Amazon SNS à partir de différents comptes \(p. 181\)](#)

- Appel de la fonction Lambda à la demande via HTTPS (Amazon API Gateway) – Outre les sources d'événements, vous pouvez également utiliser HTTPS pour appeler votre fonction Lambda. Pour ce faire, définissez une API REST personnalisée et un point de terminaison via API Gateway. Pour obtenir plus d'informations et pour voir le didacticiel correspondant, consultez la section [Utilisation de AWS Lambda avec Amazon API Gateway \(p. 186\)](#).
- Appel de la fonction Lambda à la demande (création de vos propres sources d'événements à l'aide d'applications personnalisées) – Les applications utilisateur telles que les applications web, mobiles ou clientes peuvent publier des événements et appeler des fonctions Lambda à l'aide des kits SDK AWS ou AWS Mobile (AWS Mobile SDK pour Android, par exemple). Pour obtenir plus d'informations et pour voir le didacticiel correspondant, consultez les sections [Mise en route \(p. 3\)](#) et [Utilisation d'AWS Lambda en tant que backend pour applications mobiles Android \(p. 200\)](#).
- Événements planifiés – Vous pouvez également configurer AWS Lambda pour appeler régulièrement le code à l'aide de la console AWS Lambda. Vous pouvez spécifier un taux fixe (nombre d'heures, de jours ou de semaines) ou vous pouvez spécifier une expression cron. Pour obtenir plus d'informations et pour voir le didacticiel correspondant, consultez la section [Utilisation de AWS Lambda avec Amazon CloudWatch Events \(p. 209\)](#).

En outre, vous pouvez également utiliser une machine d'état Lambda. Pour plus d'informations, consultez [Utilisation d'une machine d'état](#).

Utilisation de AWS Lambda avec Amazon S3

Amazon S3 peut publier des événements (par exemple, lorsqu'un objet est créé dans un compartiment) dans AWS Lambda et transmettre des données d'événement comme paramètre pour appeler votre fonction Lambda. Cette intégration vous permet d'écrire des fonctions Lambda qui traitent les événements Amazon S3. Dans Amazon S3, vous ajoutez la configuration des notifications de compartiment qui identifie le type d'événement que vous souhaitez qu'Amazon S3 publie et la fonction Lambda que vous voulez appeler.

Notez les considérations suivantes concernant l'intégration d'Amazon S3 et d'AWS Lambda :

- Modèle (asynchrone) non basé sur les flux – Il s'agit d'un modèle (consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#)), où Amazon S3 surveille un compartiment et transmet les données d'événement comme paramètre pour appeler la fonction Lambda. Dans un modèle push, vous gérez le mappage de source d'événement dans Amazon S3 à l'aide de la configuration de notifications de

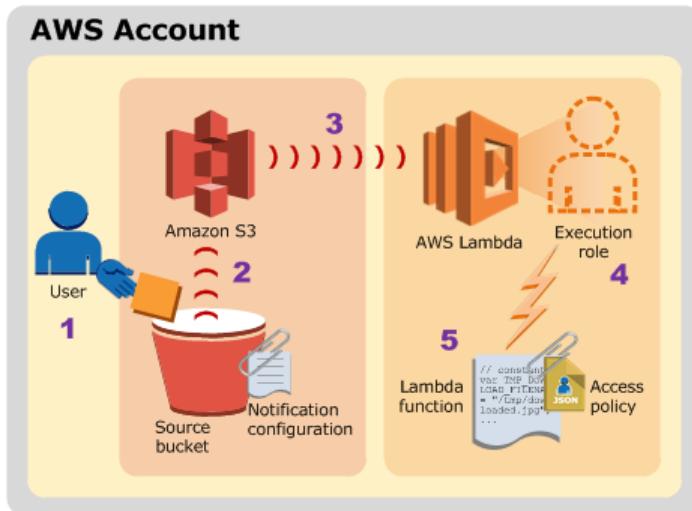
compartiment. Dans cette configuration, vous spécifiez les types d'événements suivis par Amazon S3 et la fonction AWS Lambda appelée par Amazon S3. Pour plus d'informations, consultez [Configuration des notifications d'événement Amazon S3](#) dans le Amazon Simple Storage Service Manuel du développeur.

- Appel asynchrone – AWS Lambda appelle une fonction Lambda avec le type d'appel Event (appel asynchrone). Pour plus d'informations sur les types d'appel, consultez la section [Types d'appel \(p. 74\)](#).
- Structure de l'événement – L'événement que la fonction Lambda reçoit concerne un seul objet et fournit des informations, telles que le nom de compartiment et le nom de clé de l'objet.

Notez que deux types de stratégies d'autorisations sont à votre disposition lorsque vous configurez l'environnement complet :

- Autorisations de la fonction Lambda – Quelle que soit l'origine de l'appel de la fonction Lambda, AWS Lambda exécute cette fonction en assumant le rôle IAM (rôle d'exécution) que vous spécifiez lors de sa création. Avec la stratégie d'autorisations associée à ce rôle, vous accordez à la fonction Lambda les autorisations dont elle a besoin. Par exemple, si la fonction Lambda doit lire un objet, vous accordez les autorisations requises pour les actions Amazon S3 correspondantes dans la stratégie d'autorisations. Pour plus d'informations, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).
- Autorisations permettant à Amazon S3 d'appeler la fonction Lambda – Amazon S3 ne peut pas appeler la fonction Lambda sans votre autorisation. Vous devez donc mettre à jour en conséquence la stratégie d'autorisations associée à cette fonction.

Le diagramme suivant résume ce processus :



1. L'utilisateur importe un objet dans un compartiment S3 (événement de création d'objet).
2. Amazon S3 détecte l'événement de création d'objet.
3. Amazon S3 appelle une fonction Lambda qui est spécifiée dans la configuration des notifications de compartiment.
4. AWS Lambda exécute la fonction Lambda en assumant le rôle d'exécution que vous avez spécifié au moment de la création de cette fonction.
5. Le fonction Lambda est exécutée.

Rubriques

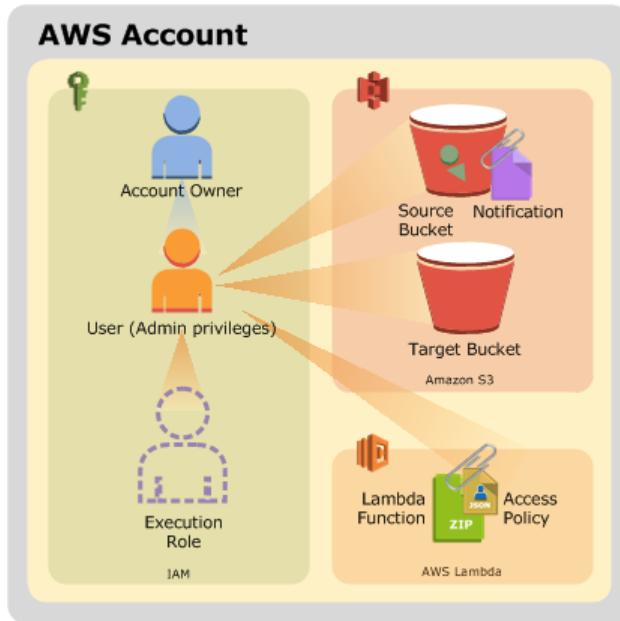
- [Didacticiel : Utilisation d'AWS Lambda avec Amazon S3 \(p. 128\)](#)
- [Exemple de code de fonction Amazon Simple Storage Service \(p. 135\)](#)

- Modèle AWS SAM pour une application Amazon S3 (p. 141)

Didacticiel : Utilisation d'AWS Lambda avec Amazon S3

Supposons que vous souhaitez créer une miniature pour chaque fichier image qui est chargé dans un compartiment. Vous pouvez créer une fonction Lambda (`CreateThumbnail`) qu'Amazon S3 appelle lorsque des objets sont créés. Ensuite, la fonction Lambda peut lire l'objet image à partir du compartiment source et créer une image miniature dans le compartiment cible.

À la fin de ce didacticiel, votre compte comportera les ressources Amazon S3, Lambda et IAM suivantes :



Ressources Lambda

- Une fonction Lambda.
- Une stratégie d'accès associée à votre fonction Lambda qui accorde à Amazon S3 l'autorisation d'appeler la fonction Lambda.

Ressources IAM

- Un rôle d'exécution qui accorde les autorisations dont votre fonction Lambda a besoin par le biais de la stratégie d'autorisations associée à ce rôle.

Ressources Amazon S3

- Un compartiment source doté d'une configuration de notification qui appelle la fonction Lambda.
- Un compartiment cible où la fonction enregistre les images redimensionnées.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaExecute.
 - Nom de rôle – **lambda-s3-role**.

La stratégie AWSLambdaExecute possède les autorisations dont la fonction a besoin pour gérer les objets dans Amazon S3 et écrire des journaux dans CloudWatch Logs.

Création des compartiments et chargement d'un exemple d'objet

Suivez les étapes requises pour créer des compartiments et importer un objet.

1. Ouvrez la console Amazon S3.
2. Créez deux compartiments. Le nom du compartiment cible correspond à celui du compartiment **source**, suivi de la mention **resized**. Par exemple : **mybucket** et **mybucketresized**.
3. Dans le compartiment source, importez l'objet **.jpg HappyFace.jpg**.

Lorsque vous appelez la fonction Lambda manuellement avant de vous connecter à Amazon S3, vous lui transmettez un échantillon de données d'événement qui spécifie le compartiment source et **HappyFace.jpg** en tant que nouvel objet créé. Il est donc essentiel de créer cet échantillon en premier lieu.

Créer la fonction

L'exemple de code suivant reçoit une entrée d'événement Amazon S3 et traite le message qu'elle contient. Il redimensionne une image dans le compartiment source et enregistre la sortie dans le compartiment cible.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction Amazon Simple Storage Service \(p. 135\)](#).

Example index.js

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();

exports.handler = function(event, context, callback) {
    // Read options from the event.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    var srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    var srcKey    =
        decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    var dstBucket = srcBucket + "resized";
    var dstKey    = "resized-" + srcKey;

    // Sanity check: validate that source and destination are different buckets.
    if (srcBucket == dstBucket) {
        callback("Source and destination buckets are the same.");
        return;
    }

    // Infer the image type.
    var typeMatch = srcKey.match(/^(.*\.(?:[a-zA-Z0-9]+))$/);
    if (!typeMatch) {
        callback("Could not determine the image type.");
        return;
    }
    var imageType = typeMatch[1];
    if (imageType != "jpg" && imageType != "png") {
        callback('Unsupported image type: ${imageType}');
        return;
    }

    // Download the image from S3, transform, and upload to a different S3 bucket.
    async.waterfall([
        function download(next) {
            // Download the image from S3 into a buffer.
            s3.getObject({
                Bucket: srcBucket,
                Key: srcKey
            },
            next);
        },
        function transform(response, next) {
            gm(response.Body).size(function(err, size) {
                // Infer the scaling factor to avoid stretching the image unnaturally.
                var scalingFactor = Math.min(
                    MAX_WIDTH / size.width,
```

```
        MAX_HEIGHT / size.height
    );
    var width = scalingFactor * size.width;
    var height = scalingFactor * size.height;

    // Transform the image buffer in memory.
    this.resize(width, height)
        .toBuffer(imageType, function(err, buffer) {
            if (err) {
                next(err);
            } else {
                next(null, response.ContentType, buffer);
            }
        });
},
function upload(contentType, data, next) {
    // Stream the transformed image to a different S3 bucket.
    s3.putObject({
        Bucket: dstBucket,
        Key: dstKey,
        Body: data,
        ContentType: contentType
    },
    next);
}
], function (err) {
    if (err) {
        console.error(
            'Unable to resize ' + srcBucket + '/' + srcKey +
            ' and upload to ' + dstBucket + '/' + dstKey +
            ' due to an error: ' + err
        );
    } else {
        console.log(
            'Successfully resized ' + srcBucket + '/' + srcKey +
            ' and uploaded to ' + dstBucket + '/' + dstKey
        );
    }
    callback(null, "message");
});
);
};
```

Le package de déploiement est un fichier .zip contenant le code de la fonction Lambda et les dépendances.

Pour créer un package de déploiement

1. Créez un dossier (`examplefolder`), puis un sous-dossier (`node_modules`).
2. Installez la plateforme Node.js. Pour plus d'informations, consultez le site web de [Node.js](#).
3. Installez les dépendances. Les exemples de code utilisent les bibliothèques suivantes :
 - Kit SDK AWS pour JavaScript dans Node.js
 - gm, GraphicsMagick pour node.js
 - Module utilitaire asynchrone

L'environnement d'exécution d'AWS Lambda intègre déjà le kit AWS SDK pour JavaScript en Node.js. Donc, vous devez uniquement installer les autres bibliothèques. Ouvrez une invite de commande, accédez au dossier `examplefolder` et installez les bibliothèques à l'aide de la commande `npm`, qui fait partie de Node.js.

```
npm install async gm
```

4. Enregistrez l'exemple de code dans un fichier nommé index.js.
5. Passez en revue le code précédent et notez les points suivants :
 - La fonction identifie le nom du compartiment source et le nom de clé de l'objet grâce aux données d'événement qu'elle reçoit comme paramètres. Si l'objet est un fichier .jpg, le code crée une miniature et l'enregistre dans le compartiment cible.
 - Le code suppose que le compartiment de destination existe et que son nom est une concaténation du nom du compartiment source, suivi de la chaîne `resized`. Par exemple, si le compartiment source identifié dans les données d'événement s'appelle `examplebucket`, le code suppose que le compartiment de destination porte le nom `examplebucketresized`.
 - Pour la miniature qu'il crée, le code dérive son nom de clé comme étant la concaténation de la chaîne `resized-`, suivie du nom de clé de l'objet source. Par exemple, si la clé de l'objet source s'appelle `sample.jpg`, le code crée un objet miniature avec la clé `resized-sample.jpg`.
6. Enregistrez le fichier sous le nom `index.js` dans `examplefolder`. Une fois que vous avez terminé cette étape, vous disposez de la structure de dossier suivante :

```
index.js
/node_modules/gm
/node_modules/async
```

7. Comptez le fichier `index.js` et le dossier `node_modules` dans `CreateThumbnail.zip`.

Pour créer la fonction

- Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn \
--timeout 10 --memory-size 1024
```

La commande précédente spécifie un délai d'attente de 10 secondes comme paramètre de configuration de la fonction. En fonction de la taille des objets que vous importez, vous devrez peut-être augmenter la valeur d'expiration à l'aide de la commande suivante de l'AWS CLI.

```
$ aws lambda update-function-configuration --function-name CreateThumbnail --timeout 30
```

Test de la fonction Lambda

Dans cette étape, vous appelez manuellement la fonction Lambda à l'aide d'un échantillon de données d'événement Amazon S3.

Pour tester la fonction Lambda

1. Enregistrez l'échantillon de données d'événement Amazon S3 suivant sous le nom de fichier `inputFile.txt`. Vous devez mettre à jour le fichier JSON en fournissant le nom de votre `compartiment source` et la clé de l'objet `.jpg`.

```
{
```

```
"Records": [
    {
        "eventVersion": "2.0",
        "eventSource": "aws:s3",
        "awsRegion": "us-west-2",
        "eventTime": "1970-01-01T00:00:00.000Z",
        "eventName": "ObjectCreated:Put",
        "userIdentity": {
            "principalId": "AIDAJDPLRKLG7UEXAMPLE"
        },
        "requestParameters": {
            "sourceIPAddress": "127.0.0.1"
        },
        "responseElements": {
            "x-amz-request-id": "C3D13FE58DE4C810",
            "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/
JRWeUWerMUE5JgHvANOjpD"
        },
        "s3": {
            "s3SchemaVersion": "1.0",
            "configurationId": "testConfigRule",
            "bucket": {
                "name": "sourcebucket",
                "ownerIdentity": {
                    "principalId": "A3NL1KOZZKExample"
                },
                "arn": "arn:aws:s3:::sourcebucket"
            },
            "object": {
                "key": "HappyFace.jpg",
                "size": 1024,
                "eTag": "d41d8cd98f00b204e9800998ecf8427e",
                "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"
            }
        }
    }
]
```

- Exécutez la commande Lambda CLI `invoke` suivante pour appeler la fonction. Notez que cette commande demande l'exécution asynchrone. Pour l'appeler de façon synchrone, spécifiez `RequestResponse` en tant que valeur de paramètre `invocation-type`.

```
$ aws lambda invoke --function-name CreateThumbnail --invocation-type Event \
--payload file://inputfile.txt outputfile.txt
```

- Vérifiez que la miniature a été créée dans le compartiment cible.

Configuration d'Amazon S3 pour publier les événements

Au cours de cette étape, vous ajoutez la configuration restante de sorte qu'Amazon S3 puisse publier des événements de création d'objet dans AWS Lambda et qu'il puisse appeler la fonction Lambda. Dans ce cas, vous effectuez les opérations suivantes :

- Vous ajoutez des autorisations à la stratégie d'accès de la fonction Lambda pour permettre à Amazon S3 de l'appeler.
- Vous ajoutez une configuration de notifications à votre compartiment source. Dans la configuration des notifications, vous renseignez les éléments suivants :
 - Type d'événement pour lequel vous souhaitez qu'Amazon S3 publie des événements. Pour ce didacticiel, vous spécifiez le type d'événement `s3:ObjectCreated:*` de sorte qu'Amazon S3 publie des événements lorsque des objets sont créés.

- Fonction Lambda à appeler.

Pour ajouter des autorisations dans la stratégie de la fonction

1. Exécutez la commande Lambda CLI add-permission suivante pour accorder au service Amazon S3 les autorisations principales (`s3.amazonaws.com`) requises pour effectuer l'action `lambda:InvokeFunction`. Notez que cette autorisation permet uniquement à Amazon S3 d'appeler la fonction si les conditions suivantes sont remplies :
 - Un événement de création d'objet est détecté dans un compartiment spécifique.
 - Ce compartiment appartient à un compte AWS spécifique. Si le propriétaire d'un compartiment supprime ce dernier, certains autres compte AWS pourront créer un compartiment portant le même nom. Cette condition garantit que seul un compte AWS spécifique peut appeler votre fonction Lambda.

```
$ aws lambda add-permission --function-name CreateThumbnail --principal s3.amazonaws.com \
--statement-id some-unique-id --action "lambda:InvokeFunction" \
--source-arn arn:aws:s3:::sourcebucket \
--source-account bucket-owner-account-id
```

2. Vérifiez la stratégie d'accès de la fonction en exécutant la commande AWS CLI get-policy.

```
$ aws lambda get-policy --function-name function-name
```

Ajoutez la configuration des notifications au niveau du compartiment source pour demander à Amazon S3 de publier les événements de création d'objet dans Lambda.

Pour configurer des notifications

1. Ouvrez la [console Amazon S3](#).
2. Choisissez le compartiment source.
3. Choisissez Properties.
4. Sous Événements, configurez une notification avec les paramètres suivants.
 - Nom – **lambda-trigger**.
 - Événements – **ObjectCreate (All)**.
 - Envoyer à – **Lambda function**.
 - Lambda – **CreateThumbnail**.

Pour plus d'informations sur la configuration d'événements, consultez [Activation des notifications d'événement](#) dans le Amazon Simple Storage Service Guide de l'utilisateur de la console.

Testez la configuration

Maintenant, vous pouvez tester la configuration comme suit :

1. Importez les objets .jpg ou .png dans le compartiment source à l'aide de la console Amazon S3.
2. Vérifiez que la miniature a été créée dans le compartiment cible à l'aide de la fonction `CreateThumbnail`.
3. Affichez les journaux dans la console CloudWatch.

Exemple de code de fonction Amazon Simple Storage Service

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js 8 \(p. 135\)](#)
- [Java 8 \(p. 137\)](#)
- [Python 3 \(p. 140\)](#)

Node.js 8

L'exemple de code suivant reçoit une entrée d'événement Amazon S3 et traite le message qu'elle contient. Il redimensionne une image dans le compartiment source et enregistre la sortie dans le compartiment cible.

Example index.js

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();

exports.handler = function(event, context, callback) {
    // Read options from the event.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    var srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    var srcKey   =
        decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    var dstBucket = srcBucket + "resized";
    var dstKey   = "resized-" + srcKey;

    // Sanity check: validate that source and destination are different buckets.
    if (srcBucket == dstBucket) {
        callback("Source and destination buckets are the same.");
        return;
    }

    // Infer the image type.
    var typeMatch = srcKey.match(/\.([^.]*$)/);
    if (!typeMatch) {
        callback("Could not determine the image type.");
        return;
    }
    var imageType = typeMatch[1];
    if (imageType != "jpg" && imageType != "png") {
        callback('Unsupported image type: ${imageType}');
        return;
    }
}
```

```
// Download the image from S3, transform, and upload to a different S3 bucket.
async.waterfall([
    function download(next) {
        // Download the image from S3 into a buffer.
        s3.getObject({
            Bucket: srcBucket,
            Key: srcKey
        },
        next);
    },
    function transform(response, next) {
        gm(response.Body).size(function(err, size) {
            // Infer the scaling factor to avoid stretching the image unnaturally.
            var scalingFactor = Math.min(
                MAX_WIDTH / size.width,
                MAX_HEIGHT / size.height
            );
            var width  = scalingFactor * size.width;
            var height = scalingFactor * size.height;

            // Transform the image buffer in memory.
            this.resize(width, height)
                .toBuffer(imageType, function(err, buffer) {
                    if (err) {
                        next(err);
                    } else {
                        next(null, response.ContentType, buffer);
                    }
                });
        });
    },
    function upload(contentType, data, next) {
        // Stream the transformed image to a different S3 bucket.
        s3.putObject({
            Bucket: dstBucket,
            Key: dstKey,
            Body: data,
            ContentType: contentType
        },
        next);
    }
], function (err) {
    if (err) {
        console.error(
            'Unable to resize ' + srcBucket + '/' + srcKey +
            ' and upload to ' + dstBucket + '/' + dstKey +
            ' due to an error: ' + err
        );
    } else {
        console.log(
            'Successfully resized ' + srcBucket + '/' + srcKey +
            ' and uploaded to ' + dstBucket + '/' + dstKey
        );
    }
    callback(null, "message");
});
});
```

Le package de déploiement est un fichier .zip contenant le code de la fonction Lambda et les dépendances.

Pour créer un package de déploiement

1. Créez un dossier (`examplefolder`), puis un sous-dossier (`node_modules`).

2. Installez la plateforme Node.js. Pour plus d'informations, consultez le site web de [Node.js](#).
3. Installez les dépendances. Les exemples de code utilisent les bibliothèques suivantes :
 - Kit SDK AWS pour JavaScript dans Node.js
 - gm, GraphicsMagick pour node.js
 - Module utilitaire asynchrone

L'environnement d'exécution d'AWS Lambda intègre déjà le kit AWS SDK pour JavaScript en Node.js. Donc, vous devez uniquement installer les autres bibliothèques. Ouvrez une invite de commande, accédez au dossier `examplefolder` et installez les bibliothèques à l'aide de la commande `npm`, qui fait partie de Node.js.

```
npm install async gm
```

4. Enregistrez l'exemple de code dans un fichier nommé `index.js`.
5. Passez en revue le code précédent et notez les points suivants :
 - La fonction identifie le nom du compartiment source et le nom de clé de l'objet grâce aux données d'événement qu'elle reçoit comme paramètres. Si l'objet est un fichier `.jpg`, le code crée une miniature et l'enregistre dans le compartiment cible.
 - Le code suppose que le compartiment de destination existe et que son nom est une concaténation du nom du compartiment source, suivi de la chaîne `resized`. Par exemple, si le compartiment source identifié dans les données d'événement s'appelle `examplebucket`, le code suppose que le compartiment de destination porte le nom `examplebucketresized`.
 - Pour la miniature qu'il crée, le code dérive son nom de clé comme étant la concaténation de la chaîne `resized-`, suivie du nom de clé de l'objet source. Par exemple, si la clé de l'objet source s'appelle `sample.jpg`, le code crée un objet miniature avec la clé `resized-sample.jpg`.
6. Enregistrez le fichier sous le nom `index.js` dans `examplefolder`. Une fois que vous avez terminé cette étape, vous disposez de la structure de dossier suivante :

```
index.js
/node_modules/gm
/node_modules/async
```

7. Comprimez le fichier `index.js` et le dossier `node_modules` dans `CreateThumbnail.zip`.

Java 8

Voici maintenant un exemple de code Java qui lit les événements Amazon S3 entrants et crée une miniature. Notez qu'il implémente l'interface `RequestHandler` fournie dans la bibliothèque `aws-lambda-java-core`. Par conséquent, au moment où vous créez une fonction Lambda, vous spécifiez la classe en tant que gestionnaire (c'est-à-dire, `example.S3EventProcessorCreateThumbnail`). Pour plus d'informations sur l'utilisation des interfaces pour créer un gestionnaire, consultez la page [Utilisation des interfaces prédéfinies pour la création d'un gestionnaire \(Java\) \(p. 329\)](#).

Le type `S3Event` que le gestionnaire utilise comme type d'entrée est l'une des classes prédéfinies dans la bibliothèque `aws-lambda-java-events`. Il offre des méthodes pour vous aider à lire les informations à partir de l'événement Amazon S3 entrant. Le gestionnaire retourne une chaîne en tant que sortie.

Example S3EventProcessorCreateThumbnail.java

```
package example;
```

```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLDecoder;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.event.S3EventNotification.S3EventNotificationRecord;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3Object;

public class S3EventProcessorCreateThumbnail implements
    RequestHandler<S3Event, String> {
    private static final float MAX_WIDTH = 100;
    private static final float MAX_HEIGHT = 100;
    private final String JPG_TYPE = (String) "jpg";
    private final String JPG_MIME = (String) "image/jpeg";
    private final String PNG_TYPE = (String) "png";
    private final String PNG_MIME = (String) "image/png";

    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);

            String srcBucket = record.getS3().getBucket().getName();
            // Object key may have spaces or unicode non-ASCII characters.
            String srcKey = record.getS3().getObject().getKey()
                .replace('+', ' ');
            srcKey = URLDecoder.decode(srcKey, "UTF-8");

            String dstBucket = srcBucket + "resized";
            String dstKey = "resized-" + srcKey;

            // Sanity check: validate that source and destination are different
            // buckets.
            if (srcBucket.equals(dstBucket)) {
                System.out
                    .println("Destination bucket must not match source bucket.");
                return "";
            }

            // Infer the image type.
            Matcher matcher = Pattern.compile(".*\\\\.([^\n\\.]*).").matcher(srcKey);
            if (!matcher.matches()) {
                System.out.println("Unable to infer image type for key "
                    + srcKey);
                return "";
            }
            String imageType = matcher.group(1);
            if (!(JPG_TYPE.equals(imageType)) && !(PNG_TYPE.equals(imageType))) {
                System.out.println("Skipping non-image " + srcKey);
                return "";
            }
        }
    }
}
```

```
// Download the image from S3 into a stream
AmazonS3 s3Client = new AmazonS3Client();
S3Object s3Object = s3Client.getObject(new GetObjectRequest(
    srcBucket, srcKey));
InputStream objectData = s3Object.getObjectContent();

// Read the source image
BufferedImage srcImage = ImageIO.read(objectData);
int srcHeight = srcImage.getHeight();
int srcWidth = srcImage.getWidth();
// Infer the scaling factor to avoid stretching the image
// unnaturally
float scalingFactor = Math.min(MAX_WIDTH / srcWidth, MAX_HEIGHT
    / srcHeight);
int width = (int) (scalingFactor * srcWidth);
int height = (int) (scalingFactor * srcHeight);

BufferedImage resizedImage = new BufferedImage(width, height,
    BufferedImage.TYPE_INT_RGB);
Graphics2D g = resizedImage.createGraphics();
// Fill with white before applying semi-transparent (alpha) images
g.setPaint(Color.white);
g.fillRect(0, 0, width, height);
// Simple bilinear resize
// If you want higher quality algorithms, check this link:
// https://today.java.net/pub/a/today/2007/04/03/perils-of-image-
getscaledinstance.html
g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
g.drawImage(srcImage, 0, 0, width, height, null);
g.dispose();

// Re-encode image to target format
ByteArrayOutputStream os = new ByteArrayOutputStream();
ImageIO.write(resizedImage, imageType, os);
InputStream is = new ByteArrayInputStream(os.toByteArray());
// Set Content-Length and Content-Type
ObjectMetadata meta = new ObjectMetadata();
meta.setContentLength(os.size());
if (JPG_TYPE.equals(imageType)) {
    meta.setContentType(JPG_MIME);
}
if (PNG_TYPE.equals(imageType)) {
    meta.setContentType(PNG_MIME);
}

// Uploading to S3 destination bucket
System.out.println("Writing to: " + dstBucket + "/" + dstKey);
s3Client.putObject(dstBucket, dstKey, is, meta);
System.out.println("Successfully resized " + srcBucket + "/"
    + srcKey + " and uploaded to " + dstBucket + "/" + dstKey);
return "Ok";
} catch (IOException e) {
    throw new RuntimeException(e);
}
}
}
```

Amazon S3 appelle la fonction Lambda avec le type d'appel `Event`, où AWS Lambda exécute le code de façon asynchrone. Ce que vous renvoyez n'a pas d'importance. Toutefois, dans le cas présent, nous mettons en œuvre une interface qui nécessite la spécification d'un type de retour. Donc, dans cet exemple, le gestionnaire utilise `String` comme type de retour.

Dependencies

- aws-lambda-java-core
- aws-lambda-java-events
- aws-java-sdk-s3

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Python 3

L'exemple de code suivant reçoit une entrée d'événement Amazon S3 et traite le message qu'elle contient. Il redimensionne une image dans le compartiment source et enregistre la sortie dans le compartiment cible.

Example CreateThumbnail.py

```
from __future__ import print_function
import boto3
import os
import sys
import uuid
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
        upload_path = '/tmp/resized-{}'.format(key)

        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}resized'.format(bucket), key)
```

Pour créer un package de déploiement

1. Copiez l'exemple de code dans un fichier nommé `CreateThumbnail.py`.
2. Créez un environnement virtuel.

```
virtualenv ~/shrink_venv
```

```
source ~/shrink_venv/bin/activate
```

3. Installez les bibliothèques dans l'environnement virtuel

```
pip install Pillow
```

```
pip install boto3
```

Note

AWS Lambda comprend le kit AWS SDK pour Python (Boto 3). Dès lors, vous n'avez pas besoin de l'inclure dans votre package de déploiement, mais vous avez la possibilité de l'ajouter pour les tests en local.

4. Ajoutez le contenu des répertoires site-packages lib et lib64 à votre fichier .zip.

```
cd $VIRTUAL_ENV/lib/python3.7/site-packages
```

```
zip -r9 ~/CreateThumbnail.zip .
```

5. Ajoutez le code Python dans le fichier .zip.

```
cd ~
```

```
zip -g CreateThumbnail.zip CreateThumbnail.py
```

Modèle AWS SAM pour une application Amazon S3

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda à partir du [didacticiel \(p. 128\)](#). Copiez le texte ci-dessous dans un fichier .yaml et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre Handler et Runtime doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  CreateThumbnail:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Timeout: 60
      Policies: AWSLambdaExecute
      Events:
        CreateThumbnailEvent:
          Type: S3
          Properties:
            Bucket: !Ref SrcBucket
            Events: s3:ObjectCreated:*
  SrcBucket:
    Type: AWS::S3::Bucket
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Utilisation de AWS Lambda avec Amazon Kinesis

Vous pouvez utiliser une fonction AWS Lambda pour traiter des enregistrements dans un [flux de données Amazon Kinesis](#). Avec Kinesis, vous pouvez collecter des données à partir de nombreuses sources et les traiter avec plusieurs consommateurs. Lambda prend en charge les itérateurs de flux de données standard et les consommateurs de flux HTTP/2.

Lambda lit les enregistrements à partir du flux de données et appelle votre fonction [de façon synchrone \(p. 74\)](#) avec un événement qui contient des enregistrements de flux. Lambda lit les enregistrements sous forme de lots et appelle votre fonction pour traiter les enregistrements d'un lot.

Example Événement d'enregistrement Kinesis

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "partitionKey": "partitionKey-3",  
                "kinesisSchemaVersion": "1.0",  
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0IDEyMy4=",  
                "sequenceNumber":  
                    "49545115243490985018280067714973144582180062593244200961"  
            },  
            "eventSource": "aws:kinesis",  
            "eventID":  
                "shardId-000000000000:49545115243490985018280067714973144582180062593244200961",  
            "invokeIdentityArn": "arn:aws:iam::account-id:role/testLEBRole",  
            "eventVersion": "1.0",  
            "eventName": "aws:kinesis:record",  
            "eventSourceARN": "arn:aws:kinesis:us-west-2:35667example:stream/  
examplestream",  
            "awsRegion": "us-west-2"  
        }  
    ]  
}
```

Si vous avez plusieurs applications qui lisent des enregistrements à partir du même flux, vous pouvez utiliser des consommateurs de flux Kinesis au lieu d'itérateurs standard. Les consommateurs ont un débit de lecture dédié afin de ne pas avoir à rivaliser avec d'autres consommateurs des mêmes données. Avec les consommateurs, Kinesis transmet les enregistrements à Lambda via une connexion HTTP/2, qui peut également réduire la latence entre l'ajout d'un enregistrement et de l'appel de la fonction.

Si votre fonction renvoie une erreur, Lambda retente de traiter le lot jusqu'à ce que le traitement réussisse ou que les données expirent. Tant que le problème n'est pas résolu, aucune donnée de la partition n'est traitée. Pour éviter le blocage des partitions et d'éventuelles pertes de données, assurez-vous de gérer et d'enregistrer les erreurs de traitement dans votre code.

Rubriques

- [Configuration de votre fonction et de votre flux de données \(p. 143\)](#)
- [Création d'un mappage de source d'événement \(p. 143\)](#)
- [API de mappage de la source d'événement \(p. 144\)](#)
- [Autorisations du rôle d'exécution \(p. 145\)](#)
- [Indicateurs Amazon CloudWatch \(p. 146\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec Amazon Kinesis \(p. 146\)](#)
- [Exemple de code de fonction \(p. 149\)](#)
- [Modèle AWS SAM pour une application Kinesis \(p. 152\)](#)

Configuration de votre fonction et de votre flux de données

Votre fonction Lambda est une application consommateur pour votre flux de données. Elle traite un lot d'enregistrements à la fois à partir de chaque partition. Vous pouvez mapper une fonction Lambda à un flux de données (itérateur standard) ou à un consommateur d'un flux ([diffusion améliorée](#)).

Pour les itérateurs standard, Lambda interroge chaque partition de votre flux Kinesis afin d'obtenir des enregistrements à une fréquence de base d'une fois par seconde. Lorsque plusieurs enregistrements sont disponibles, Lambda continue à traiter les lots jusqu'à ce qu'il reçoive un lot dont la taille est inférieure à la taille de lot maximale configurée. La fonction partage le débit en lecture avec d'autres consommateurs de la partition.

Pour réduire la latence et d'optimiser le débit en lecture, vous pouvez créer un consommateur de flux de données. Les consommateurs de flux obtiennent une connexion dédiée pour chaque partition qui n'a pas d'impact sur les autres applications lisant sur le flux. Le débit dédié peut aider si vous avez de nombreuses applications lisant les mêmes données, ou si vous retraitez un flux avec de gros enregistrements.

Les consommateurs de flux utilisent HTTP/2 afin de réduire la latence en transférant les enregistrements à Lambda via une connexion longue durée et en comprimant les en-têtes de requête. Vous pouvez créer un consommateur de flux avec l'API [RegisterStreamConsumer](#) d'Kinesis.

```
$ aws kinesis register-stream-consumer --consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/mystream
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream/consumer/
con1:1540591608",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

Pour augmenter la vitesse à laquelle votre fonction traite les enregistrements, ajoutez des partitions à votre flux de données. Lambda traite dans l'ordre les enregistrements de chaque partition, et arrête de traiter des enregistrements supplémentaires d'une partition si votre fonction renvoie une erreur. Plus de partitions signifient plus de lots traités en une seule fois, ce qui réduit l'impact des erreurs sur la simultanéité.

Note

Si la taille totale des enregistrements dans un lot dépasse la [limite de charge utile \(p. 288\)](#), Lambda divise le lot en lots plus petits à traiter.

Si votre fonction ne peut pas augmenter sa capacité pour gérer une exécution simultanée par partition, [demandez une augmentation de limite \(p. 288\)](#) ou [réservez la simultanéité \(p. 279\)](#) pour votre fonction. La simultanéité disponible pour votre fonction doit correspondre au nombre de partitions dans votre flux de données Kinesis ou dépasser ce nombre.

Création d'un mappage de source d'événement

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des enregistrements à partir de votre flux de données à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter les mêmes données avec plusieurs fonctions Lambda, ou traiter des éléments en provenance de plusieurs flux de données avec une seule fonction.

Pour configurer votre fonction afin de lire à partir de Kinesis dans la console Lambda, créez un déclencheur Kinesis.

Pour créer un déclencheur

1. Open the Lambda console [Functions page](#).
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez un type de déclencheur pour ajouter un déclencheur à votre fonction.
4. Sous Configurer les déclencheurs, configurez les options de sources d'événements, puis choisissez Ajouter.
5. Choisissez Enregistrer.

Lambda prend en charge les options suivantes pour les sources d'événements Kinesis.

Options de source d'événement

- Flux Kinesis : le flux Kinesis à partir duquel les enregistrements sont lus.
- Consommateur (facultatif) : utilisez un flux consommateur pour lire depuis le flux sur une connexion dédiée.
- Taille de lot : nombre d'enregistrements à lire à partir d'une partition dans chaque lot, jusqu'à 10 000. Lambda transmet tous les enregistrements dans le lot à la fonction en un seul appel, tant que la taille totale des événements ne dépasse pas la limite de charge utile de 6 MB.
- Position de départ : traitez uniquement les nouveaux enregistrements, tous les enregistrements existants ou les enregistrements créés après une certaine date.
 - Dernier : traitez les nouveaux enregistrements ajoutés au flux.
 - Trim horizon (Supprimer l'horizon) : traitez tous les enregistrements du flux.
 - At timestamp (À l'horodatage) : traitez les enregistrements à partir d'un moment spécifique.

Après le traitement de tous les enregistrements existants, la fonction est à jour et continue à traiter les nouveaux enregistrements.

- Activé : désactivez la source d'événement pour arrêter le traitement des enregistrements. Lambda assure le suivi du dernier enregistrement traité et reprend le traitement à ce point lorsqu'il est réactivé.

Pour gérer ultérieurement la configuration de la source d'événement, choisissez le déclencheur dans le concepteur.

API de mappage de la source d'événement

Pour créer le mappage de la source d'événement avec l'AWS CLI, utilisez l'API [CreateEventSourceMapping \(p. 406\)](#). L'exemple suivant utilise l'AWS CLI pour mapper une fonction nommée `my-function` à un flux de données Kinesis. Le flux de données est spécifié par un Amazon Resource Name (ARN), avec une taille de lot de 500, à partir de l'horodatage en temps unix.

```
$ aws lambda create-event-source-mapping --function-name my-function --no-enabled \
--batch-size 500 --starting-position AT_TIMESTAMP --starting-position-timestamp 1541139109
\
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/my-stream
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-kinesis-
stream",
```

```
"FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
"LastModified": 1541139209.351,
"LastProcessingResult": "No records processed",
"State": "Creating",
"StateTransitionReason": "User action"
}
```

Pour utiliser un consommateur, spécifiez le consommateur l'ARN au lieu de l'ARN du flux. L'option --no-enabled crée le mappage de source d'événement sans l'activer. Pour l'activer, utilisez update-event-source-mapping.

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b1128 --enabled
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b1128",
    "BatchSize": 500,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-kinesis-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541190239.996,
    "LastProcessingResult": "No records processed",
    "State": "Enabling",
    "StateTransitionReason": "User action"
}
```

Pour supprimer le mappage de source d'événement, utilisez delete-event-source-mapping.

```
$ aws lambda delete-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-kinesis-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541190240.0,
    "LastProcessingResult": "No records processed",
    "State": "Deleting",
    "StateTransitionReason": "User action"
}
```

Autorisations du rôle d'exécution

Lambda a besoin des autorisations suivantes pour gérer les ressources liées à votre flux de données Kinesis. Ajoutez-les au [rôle d'exécution \(p. 261\)](#) de la fonction.

- [kinesis:DescribeStream](#)
- [kinesis:DescribeStreamSummary](#)
- [kinesis:GetRecords](#)
- [kinesis:GetShardIterator](#)
- [kinesis>ListShards](#)
- [kinesis>ListStreams](#)
- [kinesis:SubscribeToShard](#)

La stratégie gérée `AWSLambdaKinesisExecutionRole` inclut ces autorisations. Pour en savoir plus, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).

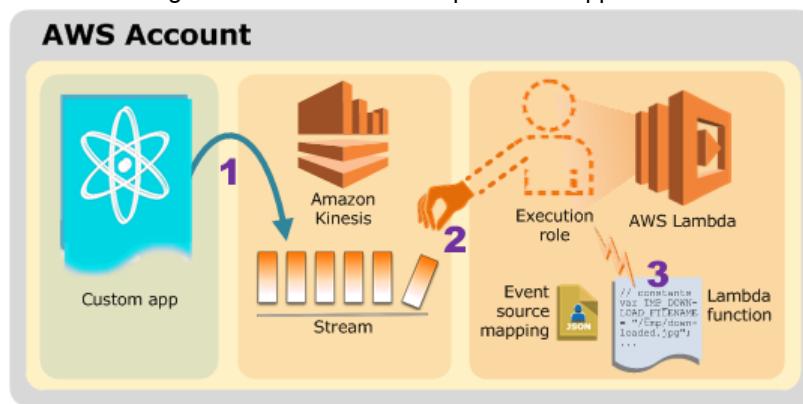
Indicateurs Amazon CloudWatch

Lambda émet la métrique `IteratorAge` lorsque votre fonction termine le traitement d'un lot d'enregistrements. Cette métrique indique l'âge du dernier enregistrement du lot à la fin du traitement. Si votre fonction traite de nouveaux événements, vous pouvez utiliser l'âge de l'itérateur pour estimer la latence entre le moment où un enregistrement est ajouté et celui où la fonction le traite.

Une tendance à la hausse de l'âge de l'itérateur peut indiquer des problèmes liés à votre fonction. Pour plus d'informations, consultez [Utiliser Amazon CloudWatch \(p. 223\)](#).

Didacticiel : Utilisation d'AWS Lambda avec Amazon Kinesis

Dans ce didacticiel, vous allez créer une fonction Lambda afin d'utiliser les événements à partir d'un flux Kinesis. Le diagramme suivant illustre le processus applicatif :



1. L'application personnalisée écrit les enregistrements dans le flux.
2. AWS Lambda interroge le flux et appelle votre fonction Lambda quand il y détecte de nouveaux enregistrements.
3. AWS Lambda exécute la fonction Lambda en assumant le rôle d'exécution que vous avez spécifié au moment de la création de cette fonction.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaKinesisExecutionRole.
 - Nom de rôle – **lambda-kinesis-role**.

La stratégie AWSLambdaKinesisExecutionRole possède les autorisations dont la fonction a besoin pour lire les éléments dans Kinesis et écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple de code suivant reçoit une entrée d'événement Kinesis et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction \(p. 149\)](#).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context) {
    //console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
};
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name ProcessKinesisRecord \
--zip-file file:///function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn
```

Test de la fonction Lambda

Appelez manuellement la fonction Lambda à l'aide de la commande CLI `invoke` AWS Lambda et d'un exemple d'événement Kinesis.

Pour tester la fonction Lambda

1. Copiez le code JSON suivant dans un fichier et enregistrez-le sous le nom `input.txt`.

```
{
    "Records": [
        {
            "kinesis": {
                "partitionKey": "partitionKey-3",
                "kinesisSchemaVersion": "1.0",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0IDEyMy4=",
                "sequenceNumber": "49545115243490985018280067714973144582180062593244200961"
            },
            "eventSource": "aws:kinesis",
            "eventID": "shardId-000000000000:49545115243490985018280067714973144582180062593244200961",
            "invokeIdentityArn": "arn:aws:iam::account-id:role/testLEBRole",
            "eventVersion": "1.0",
            "eventName": "aws:kinesis:record",
            "eventSourceARN": "arn:aws:kinesis:us-west-2:35667example:stream/examplestream",
            "awsRegion": "us-west-2"
        }
    ]
}
```

2. Utilisez la commande `invoke` pour envoyer l'événement à la fonction.

```
$ aws lambda invoke --invocation-type RequestResponse \
--function-name ProcessKinesisRecords --payload file://input.txt out.txt
```

La réponse est enregistrée dans out.txt.

Créer un flux Kinesis

Pour créer un flux, utilisez la commande `create-stream` .

```
$ aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Exécutez la commande `describe-stream` suivante pour obtenir l'ARN du flux.

```
$ aws kinesis describe-stream --stream-name examplestream  
arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Vous utilisez l'ARN du flux à l'étape suivante pour associer le flux à la fonction Lambda.

Ajout d'une source d'événement dans AWS Lambda

Exécutez la commande `add-event-source` AWS CLI suivante.

```
$ aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \
```

```
--event-source arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \
--batch-size 100 --starting-position TRIM_HORIZON
```

Notez l'ID de mappage pour une utilisation ultérieure. Pour obtenir une liste des mappages de source d'événement, exécutez la commande suivante `list-event-source-mappings`.

```
$ aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Dans la réponse, vous pouvez vérifier que la valeur d'état indique `enabled`. Les mappages d'événement source peut être désactivés pour suspendre temporairement l'interrogation, ce qui entraîne la perte d'enregistrements.

Testez la configuration

Pour tester le mappage de source d'événement, ajoutez des enregistrements d'événements à votre Kinesis. La valeur `--data` est une valeur codée en base64 de la chaîne "Hello, this is a test.". Vous pouvez exécuter la même commande plus d'une fois pour ajouter plusieurs enregistrements dans le flux.

```
$ aws kinesis put-record --stream-name examplestream \
--data "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==" --partition-key shardId-000000000000
```

Lambda utilise le rôle d'exécution pour lire les enregistrements du flux. Ensuite, il appelle la fonction Lambda en transmettant des lots d'enregistrements. La fonction décode les données de chaque enregistrement et les consigne, en envoyant le résultat à CloudWatch Logs. Affichez les journaux dans la [console CloudWatch](#).

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js 8 \(p. 149\)](#)
- [Java 8 \(p. 150\)](#)
- [C# \(p. 150\)](#)
- [Python 3 \(p. 151\)](#)
- [Go \(p. 152\)](#)

Node.js 8

L'exemple de code suivant reçoit une entrée d'événement Kinesis et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context) {
    //console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
```

```
// Kinesis data is base64 encoded so decode here
var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
console.log('Decoded payload:', payload);
});
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Java 8

Voici un exemple de code Java qui reçoit des données d'enregistrements d'événement Kinesis en tant qu'entrée et qui les traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Dans le code, `recordHandler` est le gestionnaire. Le gestionnaire utilise la classe `KinesisEvent` prédéfinie dans la bibliothèque `aws-lambda-java-events`.

Example `ProcessKinesisEvents.java`

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.KinesisEventRecord;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent, Void>{
    @Override
    public Void recordHandler(KinesisEvent event, Context context)
    {
        for(KinesisEventRecord rec : event.getRecords()) {
            System.out.println(new String(rec.getKinesis().getData().array()));
        }
        return null;
    }
}
```

Si le gestionnaire ne renvoie aucune exception, Lambda considère le lot d'enregistrements entrants comme traité avec succès et commence à lire les nouveaux enregistrements dans le flux. Si le gestionnaire renvoie une exception, Lambda considère le lot d'enregistrements entrants comme non traité et appelle à nouveau la fonction avec le même lot d'enregistrements.

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`
- `aws-java-sdk-s3`

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

C#

Voici un exemple de code C# qui reçoit des données d'enregistrement d'événement Kinesis en tant qu'entrée et qui les traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Dans le code, `HandleKinesisRecord` est le gestionnaire. Le gestionnaire utilise la classe `KinesisEvent` prédéfinie dans la bibliothèque `Amazon.Lambda.KinesisEvents`.

Example ProcessingKinesisEvents.cs

```
using System;
using System.IO;
using System.Text;

using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;

namespace KinesisStreams
{
    public class KinesisSample
    {
        [LambdaSerializer(typeof(JsonSerializer))]
        public void HandleKinesisRecord(KinesisEvent kinesisEvent)
        {
            Console.WriteLine($"Beginning to process {kinesisEvent.Records.Count} records...");

            foreach (var record in kinesisEvent.Records)
            {
                Console.WriteLine($"Event ID: {record.EventId}");
                Console.WriteLine($"Event Name: {record.EventName}");

                string recordData = GetRecordContents(record.Kinesis);
                Console.WriteLine($"Record Data:");
                Console.WriteLine(recordData);
            }
            Console.WriteLine("Stream processing complete.");
        }

        private string GetRecordContents(KinesisEvent.Record streamRecord)
        {
            using (var reader = new StreamReader(streamRecord.Data, Encoding.ASCII))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Replace the `Program.cs` in a .NET Core project with the above sample. For instructions, see [Interface de ligne de commande .NET Core \(p. 358\)](#).

Python 3

Voici un exemple de code Python qui reçoit des données d'enregistrements d'événement Kinesis en tant qu'entrée et qui les traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Example ProcessKinesisRecords.py

```
from __future__ import print_function
import json
import base64
def lambda_handler(event, context):
    for record in event['Records']:
        #Kinesis data is base64 encoded so decode here
        payload=base64.b64decode(record["kinesis"]["data"])
```

```
    print("Decoded payload: " + str(payload))
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Go

Voici un exemple de code Go qui reçoit les données d'enregistrement d'événement Kinesis en tant qu'entrée et qui les traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Example ProcessKinesisRecords.go

```
import (
    "strings"
    "github.com/aws/aws-lambda-go/events"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) {
    for _, record := range kinesisEvent.Records {
        kinesisRecord := record.Kinesis
        dataBytes := kinesisRecord.Data
        dataText := string(dataBytes)

        fmt.Printf("%s Data = %s \n", record.EventName, dataText)
    }
}
```

Build the executable with `go build` and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Go \(p. 344\)](#).

Modèle AWS SAM pour une application Kinesis

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda à partir du [didacticiel \(p. 146\)](#). Copiez le texte ci-dessous dans un fichier YAML et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre `Handler` et `Runtime` doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

Example kinesis-trigger.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  ProcessKinesisRecords:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Policies: AWSLambdaKinesisExecutionRole
      Events:
        Stream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt ExampleStream.Arn
            BatchSize: 100
            StartingPosition: TRIM_HORIZON
```

```
ExampleStream:  
  Type: AWS::Kinesis::Stream  
  Properties:  
    ShardCount: 1
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Utilisation de AWS Lambda avec Amazon SQS

Vous pouvez utiliser une fonction AWS Lambda pour traiter les messages d'une [file d'attente Amazon Simple Queue Service \(Amazon SQS\) standard](#). Avec Amazon SQS, vous pouvez décharger des tâches depuis un composant de votre application en les envoyant vers une file d'attente et en les traitant de manière asynchrone.

Lambda interroge la file d'attente et appelle votre fonction [de façon synchrone \(p. 74\)](#) avec un événement contenant les messages de la file d'attente. Lambda lit les messages par lots et appelle votre fonction une fois pour chaque lot. Lorsque la fonction traite un lot avec succès, Lambda supprime ses messages depuis la file d'attente.

Example Événement de message Amazon SQS

```
{  
  "Records": [  
    {  
      "messageId": "c80e8021-a70a-42c7-a470-796e1186f753",  
      "receiptHandle": "AQEBJQ+/u6NsnT5t8Q/  
VbVxgdU14TMKZ5FqhksRDIQvLBhwNvADoBxYSOVeCBXdnS9P  
+erlTtwEALHsnBXynkfPLH3BOUqmgzP25U8k18eHzq6RALzrSOfT08ox9dcp6GLmW33YjO3zkq5VRYYq1JgLCiAZUpY2D4UQcE5D1Vm  
+xtVaOctYeInjaQJ1u3mWx9T7tork3uAlOeluyFjCWU5aPX/10HhWCGi2EPFZj6vchNqDOJC/  
Y2k1gkvqCjz1CZ16F1Z7UVPOx3AMoszPuOYZ  
+Nuqpx2uCE2MHTtMHD8PVjlsWirt56oUr6JPp9aRG06bitPIOmi4dX0FmuMKD6u/JnuZCp+AxtJVTmSHS8Ixt/  
twsKU7A+fiMK01NtD5msNgVPoe9JbFtlGwvTQ==",  
      "body": "{\"foo\":\"bar\"}",  
      "attributes": {  
        "ApproximateReceiveCount": "3",  
        "SentTimestamp": "1529104986221",  
        "SenderId": "594035263019",  
        "ApproximateFirstReceiveTimestamp": "1529104986230"  
      },  
      "messageAttributes": {},  
      "md5OfBody": "9bb58f26192e4ba00f01e2e7b136bbd8",  
      "eventSource": "aws:sqs",  
      "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:MyQueue",  
      "awsRegion": "us-west-2"  
    }  
  ]  
}
```

Lambda utilise l'[interrogation longue](#) pour interroger une file d'attente jusqu'à ce qu'elle devienne active. Lorsque les messages sont disponibles, Lambda augmente la vitesse à laquelle il lit les lots et appelle votre fonction jusqu'à atteindre une limite de simultanéité. Pour plus d'informations sur la façon dont Lambda est mis à l'échelle pour traiter les messages dans votre file d'attente Amazon SQS, consultez [Présentation du comportement de dimensionnement \(p. 79\)](#).

Lorsqu'Lambda lit un message depuis la file d'attente, ce dernier reste dans la file d'attente mais devient masqué jusqu'à ce qu'Lambda le supprime. Si votre fonction renvoie une erreur ou n'arrête pas de s'exécuter avant l'expiration du [délai de visibilité](#) de la file d'attente, elle redevient visible. Ensuite, Lambda

l'envoie à nouveau à votre fonction Lambda. Tous les messages d'un lot en échec retournent dans la fil d'attente, ainsi le code de la fonction peut traiter le même message plusieurs fois sans effets secondaires.

Sections

- [Configuration d'une file d'attente à utiliser avec Lambda \(p. 154\)](#)
- [Configuration d'une file d'attente en tant que source d'événement \(p. 154\)](#)
- [Autorisations du rôle d'exécution \(p. 155\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec Amazon Simple Queue Service \(p. 155\)](#)
- [Exemple de code de fonction Amazon SQS \(p. 158\)](#)
- [Modèle AWS SAM pour une application Amazon SQS \(p. 161\)](#)

Configuration d'une file d'attente à utiliser avec Lambda

Créez une file d'attente [Amazon SQS standard](#) à utiliser en tant que source d'événement pour votre fonction Lambda. Ensuite, configurez cette file d'attente pour laisser le temps à votre fonction Lambda de traiter chaque lot d'événements, et pour que Lambda réessaie le traitement en réponse aux erreurs de limitation lors de sa mise à l'échelle.

Pour laisser à votre fonction le temps nécessaire pour traiter chaque lot d'enregistrements, définissez le délai de visibilité de la file d'attente source sur au moins 6 fois le [délai d'attente \(p. 29\)](#) que vous configurez sur votre fonction. Le délai supplémentaire permet à Lambda de réessayer si l'exécution de la fonction est limitée lors du traitement d'un lot précédent.

En cas d'échec répété du traitement d'un message, Amazon SQS peut l'envoyer à une [file d'attente de lettres mortes](#). Configurez une file d'attente de lettres mortes sur votre file d'attente source pour conserver les messages dont le traitement a échoué pour le dépannage. Définissez le paramètre `maxReceiveCount` de la stratégie de redirection de la file d'attente sur au moins 5, afin d'éviter d'envoyer des messages à la file d'attente de lettres mortes en raison d'une limitation.

Configuration d'une file d'attente en tant que source d'événement

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des éléments de votre file d'attente à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter des éléments de plusieurs files d'attente avec une seule fonction. Lorsque Lambda appelle la fonction cible, l'événement peut contenir plusieurs éléments, jusqu'à une taille de lot maximale configurable.

Pour ajouter un mappage de source d'événement pour une file d'attente Amazon SQS

1. Open the Lambda console [Functions page](#).
2. Choisissez une fonction.
3. Sous Ajouter des déclencheurs, choisissez SQS.
4. Sous Configurer les déclencheurs, configurez l'événement source.
 - File d'attente SQS – Spécifiez la file d'attente source.
 - Taille de lot – Spécifiez le nombre maximal d'éléments à lire depuis la file d'attente et à envoyer à votre fonction, en un seul appel.
 - Activé – Décochez la case pour désactiver la source d'événement.
5. Choisissez Ajouter.
6. Choisissez Enregistrer.

Configurez le délai d'attente de la fonction, afin de laisser suffisamment de temps pour traiter le lot entier d'éléments. Si les éléments sont longs à traiter, choisissez une taille de lot plus petite. Une grande taille de lot peut améliorer l'efficacité pour des charges de travail qui sont très rapides ou qui induisent beaucoup d'efforts supplémentaires. Toutefois, si votre fonction renvoie une erreur, tous les éléments du lot retournent dans la file d'attente. Si vous configurez une [simultanéité réservée \(p. 280\)](#) sur votre fonction, définissez un minimum de 5 exécutions simultanées pour réduire le risque d'erreurs de limitation lorsque Lambda appelle votre fonction.

Pour configurer une source de l'événement avec l'API Lambda ou le kit AWS SDK, utilisez les actions [UpdateEventSourceMapping \(p. 504\)](#) et [CreateEventSourceMapping \(p. 406\)](#).

Autorisations du rôle d'exécution

Lambda exige les autorisations suivantes pour gérer les messages dans la file d'attente Amazon SQS. Ajoutez-les au [rôle d'exécution \(p. 261\)](#) de la fonction.

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

Pour plus d'informations, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).

Didacticiel : Utilisation d'AWS Lambda avec Amazon Simple Queue Service

Dans ce didacticiel, vous allez créer une fonction Lambda, afin de consommer les messages à partir d'une file d'attente [Amazon SQS](#).

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.

2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaSQSQueueExecutionRole.
 - Nom de rôle – **lambda-sqs-role**.

La stratégie AWSLambdaSQSQueueExecutionRole possède les autorisations dont la fonction a besoin pour lire les éléments dans Amazon SQS et écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple de code suivant reçoit une entrée d'événement Amazon SQS et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction Amazon SQS \(p. 158\)](#).

Example index.js

```
exports.handler = async function(event, context) {  
    event.Records.forEach(record => {  
        const { body } = record;  
        console.log(body);  
    });  
    return {};  
}
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name ProcessSQSRecord \  
--zip-file file:///function.zip --handler index.handler --runtime nodejs8.10 \  
--role role-arn
```

Test de la fonction

Appelez manuellement la fonction Lambda à l'aide de la commande CLI `invoke` AWS Lambda et d'un exemple d'événement Amazon Simple Queue Service.

Si le gestionnaire ne renvoie aucune exception, Lambda considère le message comme traité avec succès et commence à lire les nouveaux messages dans la file d'attente. Une fois qu'un message est traité avec succès, il est automatiquement supprimé de la file d'attente. Si le gestionnaire renvoie une exception, Lambda considère les messages entrants comme non traités et appelle à nouveau la fonction avec le même lot de messages.

1. Copiez le code JSON suivant dans un fichier et enregistrez-le sous le nom `input.txt`.

```
{  
    "Records": [  
        {  
            "messageId": "c80e8021-a70a-42c7-a470-796e1186f753",  
            "receiptHandle": "AQEBJQ+/u6NsnT5t8Q/  
VbVxgdU14TMKZ5FqhksRdiQvLBhwNvADoBxYSOVeCBXdnS9P  
+erlTtwEALHsnBXynkfPLH3BOUqmzP25U8kl8eHzg6RALzrSOfTO8ox9dcP6GLmW33YjO3zkq5VRYYQ1JgLCiAZUpY2D4UqcE5  
+xtVaOctYeINjaQJ1u3mWx9T7tork3uAloeluyFjCWU5aPX/10HhWCGi2EPbzj6vchNqDOJC/  
Y2k1gkvqCjz1CZ16FlZ7UVPOx3AMoszPuOYZ  
+Nuqpx2uCE2MHTtMHD8PVjlsWirt56oUr6JPp9aRG06bitPIOmi4dXOFmuMKD6u/JnuZCp+AxtJVTmSHS8IXt/  
twsKU7A+fiMK01NtD5msNgVPoe9JbFtlGwvTQ==",  
            "body": "{\"foo\":\"bar\"}",  
            "attributes": {  
                "ApproximateReceiveCount": "3",  
                "SentTimestamp": "1529104986221",  
                "SenderId": "594035263019",  
                "ApproximateFirstReceiveTimestamp": "1529104986230"  
            },  
            "messageAttributes": {},  
            "md5OfBody": "9bb58f26192e4ba00f01e2e7b136bbd8",  
            "eventSource": "aws:sqs",  
            "eventSourceARN": "arn:aws:sqs:us-west-2:594035263019:NOTFIFOQUEUE",  
            "awsRegion": "us-west-2"  
        }  
    ]  
}
```

2. Exécutez la commande `invoke` suivante.

```
$ aws lambda invoke --invocation-type RequestResponse --function-name ProcessSQSRecord  
\  
--payload file://input.txt outputfile.txt
```

La commande `invoke` spécifie `RequestResponse` comme type d'appel, qui demande l'exécution synchrone. Pour plus d'informations, consultez [Types d'appel \(p. 74\)](#).

3. Vérifiez la sortie dans le fichier `outputfile.txt`.

Créer une file d'attente Amazon SQS

Créez une file d'attente Amazon SQS que la fonction Lambda peut utiliser comme une source de l'événement.

Pour créer une file d'attente

1. Connectez-vous à la AWS Management Console et ouvrez la console Amazon SQS à l'adresse <https://console.aws.amazon.com/sqs/>.
2. Dans la console Amazon SQS, créez une file d'attente.
3. Notez ou enregistrez l'ARN (Amazon Resource Name) de la file d'attente. Vous en aurez besoin à l'étape suivante lorsque vous associerez la file d'attente à la fonction Lambda.

Créez un mappage de source d'événement dans AWS Lambda. Ce mappage de source d'événement associe la file d'attente Amazon SQS avec votre fonction Lambda. Une fois que vous créez ce mappage de source d'événement, AWS Lambda commence à interroger la file d'attente.

Testez l'environnement complet. Lorsque vous effectuez les mises à jour de la file d'attente, Amazon Simple Queue Service écrit des messages dans la file d'attente. AWS Lambda interroge la file d'attente,

déetecte les nouveaux enregistrements et exécute la fonction Lambda en votre nom en transmettant les événements (ici, des messages Amazon SQS à la fonction).

Configurer la source de l'événement

Pour créer un mappage entre la file d'attente Amazon SQS spécifiée et la fonction Lambda, exécutez la commande AWS CLI `create-event-source-mapping` suivante. Une fois la commande exécutée, notez ou enregistrez l'UUID. Vous aurez besoin de l'UUID pour faire référence au mappage de source d'événement dans les autres commandes (par exemple, si vous décidez de supprimer le mappage).

```
$ aws lambda create-event-source-mapping --function-name ProcessSQSRecord \
--event-source SQS-queue-arn --batch-size 1
```

Pour obtenir la liste des mappages de source d'événement, exécutez la commande suivante.

```
$ aws lambda list-event-source-mappings --function-name ProcessSQSRecord \
--event-source SQS-queue-arn
```

Cette liste renvoie tous les mappages de source d'événement que vous avez créés et indique la valeur `LastProcessingResult` pour chacun d'eux, entre autres. Ce champ est utilisé pour fournir un message d'information en cas de problème. Les valeurs comme `No records processed` (signale qu'AWS Lambda n'a pas commencé l'interrogation ou que la file d'attente ne contient aucun enregistrement) et `OK` (signifie qu'AWS Lambda est parvenu à lire les enregistrements à partir de la file d'attente et à appeler la fonction Lambda) indiquent qu'il n'y pas de problèmes. Dans le cas contraire, vous recevez un message d'erreur.

Testez la configuration

Maintenant, vous pouvez tester la configuration comme suit :

1. Dans la console Amazon SQS, envoyez des messages à la file d'attente. Amazon SQS écrit les enregistrements de ces actions dans la file d'attente.
2. AWS Lambda interroge la file d'attente et appelle la fonction Lambda quand il détecte une mise à jour de la file, en transmettant les données d'événement qu'il trouve dans cette dernière.
3. Votre fonction s'exécute et crée des journaux dans Amazon CloudWatch. Vous pouvez également vérifier les journaux signalés dans la console Amazon CloudWatch.

Exemple de code de fonction Amazon SQS

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 158\)](#)
- [Java \(p. 159\)](#)
- [C# \(p. 160\)](#)
- [Go \(p. 160\)](#)
- [Python \(p. 161\)](#)

Node.js

Voici un exemple de code Go qui reçoit un message d'événement Amazon SQS en tant qu'entrée et qui le traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Example index.js (Node.js 8)

```
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Example index.js (Node.js 6)

```
event.Records.forEach(function(record) {
  var body = record.body;
  console.log(body);
});
callback(null, "message");
};
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Java

Voici un exemple de code Java qui reçoit un message d'événement Amazon SQS en tant qu'entrée et qui le traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Dans le code, `handleRequest` est le gestionnaire. Le gestionnaire utilise la classe `SQSEvent` prédéfinie dans la bibliothèque `aws-lambda-java-events`.

Example ProcessSQSRecord.java

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class ProcessSQSEvents implements RequestHandler<SQSEvent, Void>{
    @Override
    public Void handleRequest(SQSEvent event, Context context)
    {
        for(SQSMessage msg : event.getRecords()){
            System.out.println(new String(msg.getSQS().getBody()));
        }
        return null;
    }
}
```

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

C#

Voici un exemple de code C# qui reçoit un message d'événement Amazon SQS en tant qu'entrée et qui le traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans la console.

Dans le code, `handleRequest` est le gestionnaire. Le gestionnaire utilise la classe `SQSEvent` prédéfinie dans la bibliothèque `AWS.Lambda.SQSEvents`.

Example `ProcessingSQSRecords.cs`

```
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace SQSLambdaFunction
{
    public class SQSLambdaFunction
    {
        public string HandlesSQSEvent(SQSEvent sqsEvent, ILambdaContext context)
        {
            Console.WriteLine($"Beginning to process {sqsEvent.Records.Count} records...");

            foreach (var record in sqsEvent.Records)
            {
                Console.WriteLine($"Message ID: {record.MessageId}");
                Console.WriteLine($"Event Source: {record.EventSource}");

                Console.WriteLine($"Record Body:");
                Console.WriteLine(record.Body);
            }

            Console.WriteLine("Processing complete.");

            return $"Processed {sqsEvent.Records.Count} records.";
        }
    }
}
```

Replace the `Program.cs` in a .NET Core project with the above sample. For instructions, see [Interface de ligne de commande .NET Core \(p. 358\)](#).

Go

Voici un exemple de code Go qui reçoit un message d'événement Amazon SQS en tant qu'entrée et qui le traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Dans le code, `handler` est le gestionnaire. Le gestionnaire utilise la classe `SQSEvent` prédéfinie dans la bibliothèque `aws-lambda-go-events`.

Example `ProcessSQSRecords.go`

```
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent) error {
    for _, message := range sqsEvent.Records {
```

```
    fmt.Printf("The message %s for event source %s = %s \n", message.MessageId,
message.EventSource, message.Body)
}

return nil
}

func main() {
    lambda.Start(handler)
}
```

Build the executable with `go build` and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Go \(p. 344\)](#).

Python

Voici un exemple de code Python qui accepte un enregistrement Amazon SQS en tant qu'entrée et qui le traite. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Suivez les instructions pour créer le package de déploiement d'une fonction AWS Lambda.

Example ProcessSQSRecords.py

```
from __future__ import print_function

def lambda_handler(event, context):
    for record in event['Records']:
        print ("test")
        payload=record["body"]
        print(str(payload))
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Modèle AWS SAM pour une application Amazon SQS

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda à partir du [didacticiel \(p. 155\)](#). Copiez le texte ci-dessous dans un fichier `.yaml` et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre `Handler` et `Runtime` doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Example of processing messages on an SQS queue with Lambda
Resources:
  MySQSQueueFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Events:
        MySQSEvent:
          Type: SQS
          Properties:
            Queue: !GetAtt MySqsQueue.Arn
```

```
BatchSize: 10
MySqsQueue:
  Type: AWS::SQS::Queue
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Utilisation de AWS Lambda avec Amazon DynamoDB

Vous pouvez utiliser une fonction AWS Lambda pour traiter des enregistrements dans un [flux Amazon DynamoDB Flux](#). Avec Flux DynamoDB, vous pouvez déclencher une fonction Lambda pour effectuer des tâches supplémentaires chaque fois qu'une table DynamoDB est mise à jour.

Lambda lit les enregistrements à partir du flux et appelle votre fonction [de façon synchrone \(p. 74\)](#) avec un événement qui contient des enregistrements de flux. Lambda lit les enregistrements sous forme de lots et appelle votre fonction pour traiter les enregistrements d'un lot.

Example Événement d'enregistrement Flux DynamoDB

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "SizeBytes": 26
      },
      "awsRegion": "us-west-2",
      "eventName": "INSERT",
      "eventSourceARN": eventsourcearn,
      "eventSource": "aws:dynamodb"
    },
    {
      "eventID": "2",
      "eventVersion": "1.0",
      "dynamodb": {
        "OldImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        }
      }
    }
  ]
}
```

```
        }
    },
    "SequenceNumber": "222",
    "Keys": {
        "Id": {
            "N": "101"
        }
    },
    "SizeBytes": 59,
    "NewImage": {
        "Message": {
            "S": "This item has changed"
        },
        "Id": {
            "N": "101"
        }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES"
},
"awsRegion": "us-west-2",
"eventName": "MODIFY",
"eventSourceARN": sourcearn,
"eventSource": "aws:dynamodb"
}
```

Lambda interroge les partitions de votre flux DynamoDB pour obtenir des enregistrements à une fréquence de base de quatre fois par seconde. Lorsque des enregistrements sont disponibles, Lambda appelle votre fonction et attend le résultat. Si le traitement réussit, Lambda reprend l'interrogation jusqu'à ce qu'il reçoive plus d'enregistrements.

Si votre fonction renvoie une erreur, Lambda retente de traiter le lot jusqu'à ce que le traitement réussisse ou que les données expirent. Tant que le problème n'est pas résolu, aucune donnée de la partition n'est traitée. Gérez et enregistrez les erreurs de traitement dans votre code pour éviter le blocage des partitions et d'éventuelles pertes de données.

Rubriques

- [Création d'un mappage de source d'événement \(p. 163\)](#)
- [Autorisations du rôle d'exécution \(p. 164\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB \(p. 164\)](#)
- [Exemple de code de fonction \(p. 169\)](#)
- [Modèle AWS SAM pour une application DynamoDB \(p. 172\)](#)

Création d'un mappage de source d'événement

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des enregistrements à partir de votre flux à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter les mêmes données avec plusieurs fonctions Lambda, ou traiter des éléments à partir de plusieurs flux avec une seule fonction.

Pour configurer votre fonction afin de lire à partir de Flux DynamoDB dans la console Lambda, créez un déclencheur DynamoDB.

Pour créer un déclencheur

1. Open the Lambda console [Functions page](#).
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez un type de déclencheur pour ajouter un déclencheur à votre fonction.

4. Sous Configurer les déclencheurs, configurez les options requises, puis choisissez Ajouter.
5. Choisissez Enregistrer.

Options de source d'événement

- Table DynamoDB : table DynamoDB à partir de laquelle a lieu la lecture des enregistrements.
- Taille de lot : nombre d'enregistrements à lire à partir d'une partition dans chaque lot, jusqu'à 10 000. Lambda transmet tous les enregistrements du lot à la fonction en un seul appel, tant que la taille totale des événements ne dépasse pas la limite de charge utile de 6 MB.
- Starting position (Position de départ) : traitez uniquement les nouveaux enregistrements, ou tous enregistrement existants.
 - Dernière : traitez les nouveaux enregistrements ajoutés au flux.
 - Trim horizon (Supprimer l'horizon) : traitez tous les enregistrements du flux.

Après le traitement de tous les enregistrements existants, la fonction est à jour et continue à traiter les nouveaux enregistrements.

- Activé : désactivez la source d'événement pour arrêter le traitement des enregistrements. Lambda assure le suivi du dernier enregistrement traité et reprend le traitement à ce point lorsqu'il est réactivé.

Pour gérer ultérieurement la configuration de la source d'événement, choisissez le déclencheur dans le concepteur.

Autorisations du rôle d'exécution

Lambda a besoin des autorisations suivantes pour gérer les ressources liées à votre flux Flux DynamoDB. Ajoutez-les au [rôle d'exécution \(p. 261\)](#) de la fonction.

- [dynamodb:DescribeStream](#)
- [dynamodb:GetRecords](#)
- [dynamodb:GetShardIterator](#)
- [dynamodb>ListStreams](#)

La stratégie gérée `AWSLambdaDynamoDBExecutionRole` inclut ces autorisations. Pour en savoir plus, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).

Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB

Dans ce didacticiel, vous allez créer une fonction Lambda afin d'utiliser les événements à partir d'un flux Amazon DynamoDB.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command
```

```
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution](#) (p. 261) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Autorisations – AWSLambdaDynamoDBExecutionRole.
 - Nom de rôle – **lambda-dynamodb-role**.

Le rôle AWSLambdaDynamoDBExecutionRole possède les autorisations dont la fonction a besoin pour lire les éléments de DynamoDB et écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple de code suivant reçoit une entrée d'événement DynamoDB et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction](#) (p. 169).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
    });
    callback(null, "message");
};
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name ProcessDynamoDBRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn
```

Test de la fonction Lambda

Dans cette étape, vous appelez la fonction Lambda manuellement via la commande CLI `invoke` AWS Lambda et l'exemple d'événement DynamoDB suivant.

Example input.txt

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "111",
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "stream-ARN"
    },
    {
      "eventID": "2",
      "eventName": "MODIFY",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "This item has changed"
          },
          "Id": {
            "N": "101"
          }
        },
        "OldImage": {

```

```
        "Message":{  
            "S":"New item!"  
        },  
        "Id":{  
            "N":"101"  
        }  
    },  
    "SequenceNumber":"222",  
    "SizeBytes":59,  
    "StreamViewType":"NEW_AND_OLD_IMAGES"  
},  
"eventSourceARN":"stream-ARN"  
},  
{  
    "eventID":"3",  
    "eventName":"REMOVE",  
    "eventVersion":"1.0",  
    "eventSource":"aws:dynamodb",  
    "awsRegion":"us-east-1",  
    "dynamodb":{  
        "Keys":{  
            "Id":{  
                "N":"101"  
            }  
        },  
        "OldImage":{  
            "Message":{  
                "S":"This item has changed"  
            },  
            "Id":{  
                "N":"101"  
            }  
        },  
        "SequenceNumber":"333",  
        "SizeBytes":38,  
        "StreamViewType":"NEW_AND_OLD_IMAGES"  
},  
"eventSourceARN":"stream-ARN"  
}  
]  
}
```

Exécutez la commande invoke suivante.

```
$ aws lambda invoke --function-name ProcessDynamoDBRecords --payload file://input.txt  
outputfile.txt
```

La fonction renvoie le message de type chaîne (message de la méthode context.succeed() dans le code) dans le corps de la réponse.

Vérifiez la sortie dans le fichier outputfile.txt.

Créer une table DynamoDB avec un flux activé

Créez une table Amazon DynamoDB en y activant un flux.

Pour créer une table DynamoDB

1. Ouvrez la [console DynamoDB](#).
2. Choisissez Create table (Créer une table).
3. Créez une table avec les paramètres suivants.

- Nom de la table – **lambda-dynamodb-stream**
 - Clé primaire – **id** (chaîne)
4. Sélectionnez Create.

Pour activer les flux

1. Ouvrez la [console DynamoDB](#).
2. Choisissez Tables.
3. Choisissez la table lambda-dynamodb-stream.
4. Sous Présentation, choisissez Gérer le flux.
5. Choisissez Enable.

Notez l'ARN du flux. Vous en aurez besoin à l'étape suivante lorsque vous associerez le flux à la fonction Lambda. Pour plus d'informations sur l'activation des flux, consultez [Capture d'activité Table avec flux DynamoDB](#).

Ajout d'une source d'événement dans AWS Lambda

Créez un mappage de source d'événement dans AWS Lambda. Ce mappage de source d'événement associe le flux DynamoDB avec votre fonction Lambda. Une fois que vous créez ce mappage de source d'événement, AWS Lambda commence à interroger le flux.

Exécutez la commande `create-event-source-mapping` AWS CLI suivante. Une fois que la commande s'exécute, notez l'UUID. Vous aurez besoin de l'UUID pour faire référence au mappage de source d'événement dans les commandes (par exemple, lors de la suppression du mappage).

```
$ aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Cette opération crée un mappage entre le flux DynamoDB spécifié et la fonction Lambda. Vous pouvez associer un flux DynamoDB à plusieurs fonctions Lambda et associer la même fonction Lambda à plusieurs flux. Toutefois, les fonctions Lambda partageront le débit de lecture du flux qui leur est commun.

Pour obtenir la liste des mappages de source d'événement, exécutez la commande suivante.

```
$ aws lambda list-event-source-mappings
```

Cette liste renvoie tous les mappages de source d'événement que vous avez créés et indique la valeur `LastProcessingResult` pour chacun d'eux, entre autres. Ce champ est utilisé pour fournir un message d'information en cas de problème. Les valeurs comme `No records processed` (signale qu'AWS Lambda n'a pas commencé l'interrogation ou que le flux ne contient aucun enregistrement) et `OK` (signifie qu'AWS Lambda est parvenu à lire les enregistrements à partir du flux et à appeler la fonction Lambda) indiquent qu'il n'y pas de problèmes. Dans le cas contraire, vous recevez un message d'erreur.

Si vous avez un grand nombre de mappages de source d'événement, utilisez le paramètre du nom de la fonction pour affiner les résultats.

```
$ aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Testez la configuration

Testez l'environnement complet. A mesure que vous effectuez des mises à jour de la table, DynamoDB écrit les enregistrements d'événement dans le flux. Quand AWS Lambda interroge le flux, il y détecte les

nouveaux enregistrements et exécute la fonction Lambda en votre nom en transmettant les événements à la fonction.

1. Dans la console DynamoDB, vous pouvez ajouter, mettre à jour et supprimer des éléments de la table. DynamoDB écrit les enregistrements de ces actions dans le flux.
2. AWS Lambda interroge le flux et appelle la fonction Lambda quand il détecte une mise à jour du flux, en transmettant les données d'événement qu'il trouve dans ce dernier.
3. Votre fonction s'exécute et crée des journaux dans Amazon CloudWatch. Vous pouvez également vérifier les journaux signalés dans la console Amazon CloudWatch.

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 169\)](#)
- [Java 8 \(p. 169\)](#)
- [C# \(p. 170\)](#)
- [Python 3 \(p. 171\)](#)
- [Go \(p. 171\)](#)

Node.js

L'exemple suivant traite les messages depuis DynamoDB et enregistre leur contenu.

Example ProcessDynamoDBStream.js

```
console.log('Loading function');

exports.lambda_handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
    });
    callback(null, "message");
};
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Java 8

L'exemple suivant traite les messages de DynamoDB, et enregistre leur contenu. `handleRequest` est le gestionnaire qu'AWS Lambda appelle et auquel il fournit les données d'événements. Le gestionnaire utilise la classe `DynamodbEvent` qui est prédéfinie dans la bibliothèque `aws-lambda-java-events`.

Example DDBEventProcessor.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.Logger;
```

```
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;

public class DDBEventProcessor implements
    RequestHandler<DynamodbEvent, String> {

    public String handleRequest(DynamodbEvent ddbEvent, Context context) {
        for (DynamodbStreamRecord record : ddbEvent.getRecords()){
            System.out.println(record.getEventID());
            System.out.println(record.geteventName());
            System.out.println(record.getDynamodb().toString());

        }
        return "Successfully processed " + ddbEvent.getRecords().size() + " records.";
    }
}
```

Si le gestionnaire ne renvoie aucune exception, Lambda considère le lot d'enregistrements entrants comme traité avec succès et commence à lire les nouveaux enregistrements dans le flux. Si le gestionnaire renvoie une exception, Lambda considère le lot d'enregistrements entrants comme non traité et appelle à nouveau la fonction avec le même lot d'enregistrements.

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

C#

L'exemple suivant traite les messages de DynamoDB, et enregistre leur contenu. `ProcessDynamoEvent` est le gestionnaire qu'AWS Lambda appelle et auquel il fournit les données d'événements. Le gestionnaire utilise la classe `DynamoDbEvent` qui est prédéfinie dans la bibliothèque `Amazon.Lambda.DynamoDBEvents`.

Example ProcessingDynamoDBStreams.cs

```
using System;
using System.IO;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

using Amazon.Lambda.Serialization.Json;

namespace DynamoDBStreams
{
    public class DdbSample
    {
        private static readonly JsonSerializer _jsonSerializer = new JsonSerializer();

        public void ProcessDynamoEvent(DynamoDBEvent dynamoEvent)
        {
            Console.WriteLine($"Beginning to process {dynamoEvent.Records.Count} records...");

            foreach (var record in dynamoEvent.Records)
```

```
{  
    Console.WriteLine($"Event ID: {record.EventID}");  
    Console.WriteLine($"Event Name: {record.EventName}");  
  
    string streamRecordJson = SerializeObject(record.Dynamodb);  
    Console.WriteLine($"DynamoDB Record:");  
    Console.WriteLine(streamRecordJson);  
}  
  
Console.WriteLine("Stream processing complete.");  
}  
  
private string SerializeObject(object streamRecord)  
{  
    using (var ms = new MemoryStream())  
    {  
        _jsonSerializer.Serialize(streamRecord, ms);  
        return Encoding.UTF8.GetString(ms.ToArray());  
    }  
}
```

Replace the `Program.cs` in a .NET Core project with the above sample. For instructions, see [Interface de ligne de commande .NET Core \(p. 358\)](#).

Python 3

L'exemple suivant traite les messages depuis DynamoDB et enregistre leur contenu.

Example ProcessDynamoDBStream.py

```
from __future__ import print_function  
  
def lambda_handler(event, context):  
    for record in event['Records']:  
        print(record['eventID'])  
        print(record['eventName'])  
    print('Successfully processed %s records.' % str(len(event['Records'])))
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Go

L'exemple suivant traite les messages depuis DynamoDB et enregistre leur contenu.

Example

```
import (  
    "strings"  
  
    "github.com/aws/aws-lambda-go/events"  
)  
  
func handleRequest(ctx context.Context, e events.DynamoDBEvent) {  
  
    for _, record := range e.Records {  
        fmt.Printf("Processing request data for event ID %s, type %s.\n", record.EventID,  
        record.EventName)
```

```
// Print new values for attributes of type String
for name, value := range record.Change.NewImage {
    if value.DataType() == events.DataTypeString {
        fmt.Printf("Attribute name: %s, value: %s\n", name, value.String())
    }
}
}
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Modèle AWS SAM pour une application DynamoDB

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'[application du didacticiel \(p. 164\)](#). Copiez le texte ci-dessous dans un fichier .yaml et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre Handler et Runtime doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  ProcessDynamoDBStream:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Policies: AWSLambdaDynamoDBExecutionRole
      Events:
        Stream:
          Type: DynamoDB
          Properties:
            Stream: !GetAtt DynamoDBTable.StreamArn
            BatchSize: 100
            StartingPosition: TRIM_HORIZON

  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        -AttributeName: id
        AttributeType: S
      KeySchema:
        -AttributeName: id
        KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
      StreamSpecification:
        StreamViewType: NEW_IMAGE
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Utilisation de AWS Lambda avec AWS CloudTrail

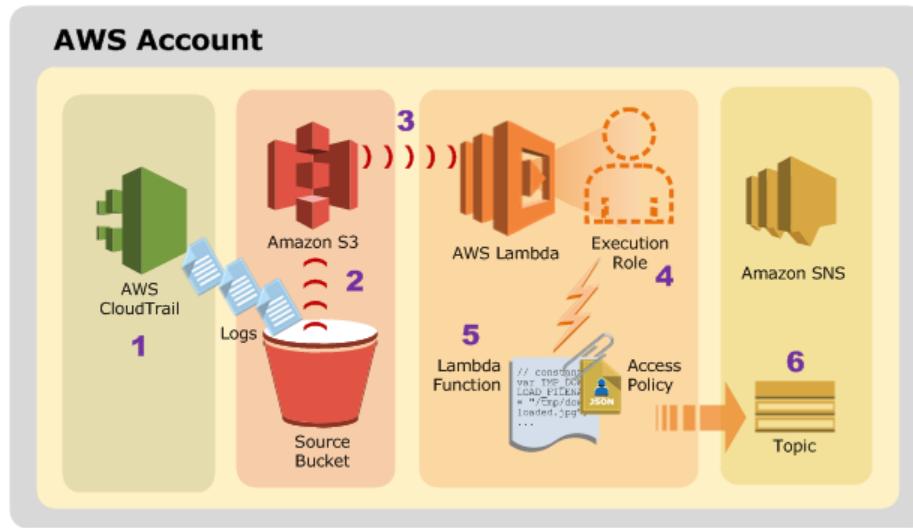
AWS CloudTrail est un service qui enregistre les actions effectuées par un utilisateur, un rôle ou un service AWS. CloudTrail capture tous les appels d'API en tant qu'événements. Pour un enregistrement continu des événements dans votre compte AWS, créez un journal de suivi. Un journal d'activité permet à CloudTrail de livrer les fichiers journaux d'événements dans un compartiment Amazon S3.

Vous pouvez tirer parti de la fonctionnalité des notifications de compartiment d'Amazon S3 et demander à ce service de publier les événements de création d'objet dans AWS Lambda. Chaque fois qu'CloudTrail écrit des journaux dans votre compartiment S3, Amazon S3 peut appeler la fonction Lambda en transmettant l'événement de création d'objet Amazon S3 comme paramètre. L'événement S3 fournit des informations, y compris le nom du compartiment et le nom de clé de l'objet de journal créé par CloudTrail. Le code de la fonction Lambda peut lire l'objet de journal et traiter les enregistrements d'accès consignés par CloudTrail. Par exemple, vous pouvez écrire le code de la fonction Lambda pour vous avertir si un appel d'API spécifique a été effectué dans votre compte.

Dans ce scénario, CloudTrail écrit les journaux d'accès dans votre compartiment S3. En ce qui concerne AWS Lambda, Amazon S3 est la source d'événement. Dès lors Amazon S3 publie les événements dans AWS Lambda et appelle la fonction Lambda.

Pour obtenir des informations détaillées sur la configuration d'Amazon S3 en tant que source d'événement, consultez [Utilisation de AWS Lambda avec Amazon S3 \(p. 126\)](#).

Le diagramme suivant résume ce processus :



1. AWS CloudTrail enregistre les journaux dans un compartiment S3 (événement de création d'objet).
2. Amazon S3 détecte l'événement de création d'objet.
3. Amazon S3 publie l'événement `s3:ObjectCreated:*` dans AWS Lambda en appelant la fonction Lambda, conformément à la configuration des notifications de compartiment. Étant donné que la stratégie d'autorisations d'accès de la fonction Lambda permet à Amazon S3 d'appeler la fonction, cela est possible.
4. AWS Lambda exécute la fonction Lambda en assumant le rôle d'exécution que vous avez spécifié au moment de la création de cette fonction.
5. La fonction Lambda lit l'événement Amazon S3 qu'elle reçoit comme paramètre, détermine où se trouve l'objet CloudTrail, le lit, puis traite les enregistrements de journal dans l'objet CloudTrail.
6. Si le journal comprend un enregistrement avec des valeurs `eventType` et `eventSource` spécifiques, il publie l'événement dans la rubrique Amazon SNS. Dans la section [Didacticiel : Utilisation d'AWS](#)

[Lambda avec AWS CloudTrail \(p. 174\)](#), vous vous abonnez à la rubrique SNS à l'aide du protocole de messagerie, afin de recevoir des notifications par e-mail.

Rubriques

- [Didacticiel : Utilisation d'AWS Lambda avec AWS CloudTrail \(p. 174\)](#)
- [Exemple de code de fonction \(p. 180\)](#)

Didacticiel : Utilisation d'AWS Lambda avec AWS CloudTrail

Dans ce scénario, AWS CloudTrail conservera des enregistrements (journaux) des appels d'API AWS effectués sur votre compte et vous informera à tout moment lorsqu'un appel d'API est effectué pour créer une rubrique SNS. Comme les appels d'API sont effectués dans votre compte, CloudTrail écrit les journaux dans un compartiment Amazon S3 que vous avez configuré. Dans ce scénario, vous voulez qu'Amazon S3 publie les événements de création d'objet dans AWS Lambda et qu'il appelle la fonction Lambda quand CloudTrail crée des objets de journaux.

Quand Amazon S3 appelle la fonction Lambda , il transmet un événement S3 qui identifie, entre autres, le nom du compartiment et le nom de clé de l'objet créé par CloudTrail. La fonction Lambda peut lire l'objet de journal et identifier les appels d'API qui ont été signalés dans le journal.

Chaque objet créé par CloudTrail dans votre compartiment S3 est un objet JSON, avec un ou plusieurs enregistrements d'événement. Chaque enregistrement, entre autres, fournit les valeurs `eventSource` et `eventName`.

```
{  
    "Records": [  
        {  
            "eventVersion": "1.02",  
            "userIdentity": {  
                ...  
            },  
            "eventTime": "2014-12-16T19:17:43Z",  
            "eventSource": "sns.amazonaws.com",  
            "eventName": "CreateTopic",  
            "awsRegion": "us-west-2",  
            "sourceIPAddress": "72.21.198.64",  
            ...  
        },  
        {  
            ...  
        },  
        ...  
    ]  
}
```

Pour illustrer ce propos, la fonction Lambda vous informe par e-mail si un appel d'API pour créer une rubrique Amazon SNS est indiqué dans le journal. Autrement dit, quand la fonction Lambda analyse le journal, elle recherche les enregistrements avec les éléments suivants :

- `eventSource = "sns.amazonaws.com"`
- `eventName = "CreateTopic"`

Si elle trouve un événement de ce type, elle le publie dans votre rubrique Amazon SNS (que vous pouvez configurer pour être averti par e-mail).

La fonction Lambda utilise un événement S3 qui fournit le nom du compartiment et le nom de clé de l'objet créé par CloudTrail. La fonction Lambda lit ensuite cet objet pour traiter les enregistrements CloudTrail.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Activer CloudTrail

Dans la console AWS CloudTrail, activez la piste en spécifiant `examplebucket` dans la région `us-west-2` pour qu'CloudTrail enregistre les journaux. Lors de la configuration du journal de suivi, n'activez pas les notifications SNS.

Pour obtenir des instructions, consultez [Création et mise à jour de votre journal de suivi](#) dans le AWS CloudTrail User Guide.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Nom de rôle – `lambda-cloudtrail-role`.
 - Autorisations – Stratégie personnalisée.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:*"  
            ],  
            "Resource": "arn:aws:logs:*:/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:PutLogEvents"  
            ],  
            "Resource": "arn:aws:logs:*:/*"  
        }  
    ]  
}
```

```
    "Action": [
        "s3:GetObject"
    ],
    "Resource": "arn:aws:s3:::examplebucket/*"
},
{
    "Effect": "Allow",
    "Action": [
        "sns:Publish"
    ],
    "Resource": "your sns topic ARN"
}
]
```

La stratégie possède les autorisations dont la fonction a besoin pour lire les éléments d'Amazon S3 et écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple suivant traite les journaux CloudTrail et envoie une notification lorsqu'une rubrique Amazon SNS a été créée.

Example index.js

```
var aws = require('aws-sdk');
var zlib = require('zlib');
var async = require('async');

var EVENT_SOURCE_TO_TRACK = /sns.amazonaws.com/;
var EVENT_NAME_TO_TRACK = /CreateTopic/;
var DEFAULT_SNS_REGION = 'us-west-2';
var SNS_TOPIC_ARN = 'The ARN of your SNS topic';

var s3 = new aws.S3();
var sns = new aws.SNS({
    apiVersion: '2010-03-31',
    region: DEFAULT_SNS_REGION
});

exports.handler = function(event, context, callback) {
    var srcBucket = event.Records[0].s3.bucket.name;
    var srcKey = event.Records[0].s3.object.key;

    async.waterfall([
        function fetchLogFromS3(next){
            console.log('Fetching compressed log from S3...');
            s3.getObject({
                Bucket: srcBucket,
                Key: srcKey
            },
            next);
        },
        function uncompressLog(response, next){
            console.log("Uncompressing log...");
            zlib.gunzip(response.Body, next);
        },
        function publishNotifications(jsonBuffer, next) {
            console.log('Filtering log...');
            var json = jsonBuffer.toString();
            console.log('CloudTrail JSON from S3:', json);
            var records;
```

```

        try {
            records = JSON.parse(json);
        } catch (err) {
            next('Unable to parse CloudTrail JSON: ' + err);
            return;
        }
        var matchingRecords = records
            .Records
            .filter(function(record) {
                return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
                    && record.eventName.match(EVENT_NAME_TO_TRACK);
            });

        console.log('Publishing ' + matchingRecords.length + ' notification(s) in parallel...');

        async.each(
            matchingRecords,
            function(record, publishComplete) {
                console.log('Publishing notification: ', record);
                sns.publish({
                    Message:
                        'Alert... SNS topic created: \n TopicARN=' +
                    record.responseElements.topicArn + '\n\n' +
                        JSON.stringify(record),
                    TopicArn: SNS_TOPIC_ARN
                }, publishComplete);
            },
            next
        );
    }
], function (err) {
    if (err) {
        console.error('Failed to publish notifications: ', err);
    } else {
        console.log('Successfully published all notifications.');
    }
    callback(null, "message");
});
};

}

```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
 2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name CloudTrailEventProcessing \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn
--timeout 10 --memory-size 1024
```

Test de la fonction Lambda

Dans cette section, vous appelez manuellement la fonction Lambda à l'aide d'un échantillon de données d'événement Amazon S3. Lorsque la fonction Lambda s'exécute, elle lit l'objet S3 (un exemple de journal CloudTrail) à partir du compartiment identifié dans les données d'événement SNS, puis elle publie un événement dans votre rubrique SNS si l'exemple de journal CloudTrail indique l'utilisation d'une API.

spécifique. Pour ce didacticiel, l'API SNS est utilisée pour créer une rubrique. Autrement dit, le journal CloudTrail signale un enregistrement identifiant sns.amazonaws.com comme eventSource et CreateTopic comme eventName.

Notez que l'un des événements de ce journal indique sns.amazonaws.com comme eventSource et CreateTopic comme eventName. La fonction Lambda lit les journaux et, si elle détecte un événement de ce type, le publie dans la rubrique Amazon SNS que vous avez créée. Vous recevez ensuite un e-mail lorsque vousappelez la fonction Lambda manuellement.

Example input.txt

```
{  
    "Records": [  
        {  
            "eventVersion": "1.02",  
            "userIdentity": {  
                "type": "Root",  
                "principalId": "account-id",  
                "arn": "arn:aws:iam::account-id:root",  
                "accountId": "account-id",  
                "accessKeyId": "access-key-id",  
                "sessionContext": {  
                    "attributes": {  
                        "mfaAuthenticated": "false",  
                        "creationDate": "2015-01-24T22:41:54Z"  
                    }  
                }  
            },  
            "eventTime": "2015-01-24T23:26:50Z",  
            "eventSource": "sns.amazonaws.com",  
            "eventName": "CreateTopic",  
            "awsRegion": "us-west-2",  
            "sourceIPAddress": "205.251.233.176",  
            "userAgent": "console.amazonaws.com",  
            "requestParameters": {  
                "name": "dropmeplease"  
            },  
            "responseElements": {  
                "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"  
            },  
            "requestID": "3fdb7834-9079-557e-8ef2-350abc03536b",  
            "eventID": "17b46459-dada-4278-b8e2-5a4ca9ff1a9c",  
            "eventType": "AwsApiCall",  
            "recipientAccountId": "account-id"  
        },  
        {  
            "eventVersion": "1.02",  
            "userIdentity": {  
                "type": "Root",  
                "principalId": "account-id",  
                "arn": "arn:aws:iam::account-id:root",  
                "accountId": "account-id",  
                "accessKeyId": "access key id",  
                "sessionContext": {  
                    "attributes": {  
                        "mfaAuthenticated": "false",  
                        "creationDate": "2015-01-24T22:41:54Z"  
                    }  
                }  
            },  
            "eventTime": "2015-01-24T23:27:02Z",  
            "eventSource": "sns.amazonaws.com",  
            "eventName": "GetTopicAttributes",  
            "awsRegion": "us-west-2",  
        }  
    ]  
}
```

```
        "sourceIPAddress": "205.251.233.176",
        "userAgent": "console.amazonaws.com",
        "requestParameters": {
            "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"
        },
        "responseElements": null,
        "requestID": "4a0388f7-a0af-5df9-9587-c5c98c29cbe",
        "eventID": "ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
        "eventType": "AwsApiCall",
        "recipientAccountId": "account-id"
    }
}
```

Appelez manuellement la fonction Lambda via l'AWS CLI, comme suit :

```
$ aws lambda invoke --function-name CloudTrailEventProcessing \
--payload file://input.txt outputfile.txt
```

Ajouter des autorisations dans la stratégie de la fonction

Ajoutez des autorisations à la stratégie de ressources de la fonction Lambda pour permettre à Amazon S3 d'appeler la fonction.

1. Exécutez la commande `add-permission` suivante pour accorder au mandataire de service Amazon S3 (`s3.amazonaws.com`) les autorisations requises pour effectuer l'action `lambda:InvokeFunction`. Notez que cette autorisation permet uniquement à Amazon S3 d'appeler la fonction si les conditions suivantes sont remplies :
 - Un événement de création d'objet est détecté dans un compartiment spécifique.
 - Ce compartiment appartient à un compte AWS spécifique. Si le propriétaire d'un compartiment supprime ce dernier, certains autres compte AWS pourront créer un compartiment portant le même nom. Cette condition garantit que seul un compte AWS spécifique peut appeler votre fonction Lambda.

```
$ aws lambda add-permission --function-name CloudTrailEventProcessing \
--statement-id Id-1 --action "lambda:InvokeFunction" --principal s3.amazonaws.com \
--source-arn arn:aws:s3:::examplebucket \
--source-account examplebucket-owner-account-id
```

2. Vérifiez la stratégie d'accès de la fonction à l'aide de la commande `get-policy`.

```
$ aws lambda get-policy --function-name function-name
```

Configurer les notifications au niveau du compartiment

Ajoutez la configuration des notifications au niveau du compartiment `examplebucket` pour demander à Amazon S3 de publier les événements de création d'objet dans Lambda. Dans cette configuration, spécifiez les éléments suivants :

- Type d'événement – Tous les types d'événement qui créent des objets.
- ARN de la fonction Lambda – Il s'agit de la fonction Lambda qu'Amazon S3 doit appeler.

```
arn:aws:lambda:us-west-2:123456789012:function:CloudTrailEventProcessing
```

Pour plus d'informations sur l'ajout d'une configuration de notifications à un compartiment, consultez [Activation des notifications d'événement](#) dans le Amazon Simple Storage Service Guide de l'utilisateur de la console.

Testez la configuration

Maintenant, vous pouvez tester la configuration comme suit :

1. Créez une rubrique Amazon SNS.
2. AWS CloudTrail crée un objet de journal dans votre compartiment.
3. Amazon S3 appelle votre fonction Lambda en transmettant l'emplacement de l'objet de journal comme données d'événement.
4. Lambda exécute votre fonction. La fonction récupère le journal, trouve un événement `CreateTopic` et envoie une notification.

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 180\)](#)

Node.js

L'exemple suivant traite les journaux CloudTrail et envoie une notification lorsqu'une rubrique Amazon SNS a été créée.

Example index.js

```
var aws = require('aws-sdk');
var zlib = require('zlib');
var async = require('async');

var EVENT_SOURCE_TO_TRACK = /sns.amazonaws.com/;
var EVENT_NAME_TO_TRACK = /CreateTopic/;
var DEFAULT_SNS_REGION = 'us-west-2';
var SNS_TOPIC_ARN = 'The ARN of your SNS topic';

var s3 = new aws.S3();
var sns = new aws.SNS({
    apiVersion: '2010-03-31',
    region: DEFAULT_SNS_REGION
});

exports.handler = function(event, context, callback) {
    var srcBucket = event.Records[0].s3.bucket.name;
    var srcKey = event.Records[0].s3.object.key;

    async.waterfall([
        function fetchLogFromS3(next){
            console.log('Fetching compressed log from S3...');
            s3.getObject({
                Bucket: srcBucket,
                Key: srcKey
            },
            next);
        },
    ],
    function(err, result) {
        if (err) {
            return callback(err);
        }
        var decompressedLog = zlib.inflateSync(result.body);
        var log = decompressedLog.toString();
        var logLines = log.split(/\r\n|\n/);

        var message = {
            TopicArn: SNS_TOPIC_ARN,
            Subject: 'CloudTrail Log for ' + srcBucket + ' - ' + srcKey,
            Message: log
        };

        sns.publish(message, function(err, response) {
            if (err) {
                return callback(err);
            }
            callback(null, response);
        });
    });
};
```

```
function uncompressLog(response, next){
    console.log("Uncompressing log...");
    zlib.gunzip(response.Body, next);
},
function publishNotifications(jsonBuffer, next) {
    console.log('Filtering log...');
    var json = jsonBuffer.toString();
    console.log('CloudTrail JSON from S3:', json);
    var records;
    try {
        records = JSON.parse(json);
    } catch (err) {
        next('Unable to parse CloudTrail JSON: ' + err);
        return;
    }
    var matchingRecords = records
        .Records
        .filter(function(record) {
            return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
                && record.eventName.match(EVENT_NAME_TO_TRACK);
        });

    console.log('Publishing ' + matchingRecords.length + ' notification(s) in
parallel...');
    async.each(
        matchingRecords,
        function(record, publishComplete) {
            console.log('Publishing notification: ', record);
            sns.publish({
                Message:
                    'Alert... SNS topic created: \n TopicARN=' +
                record.responseElements.topicArn + '\n\n' +
                    JSON.stringify(record),
                TopicArn: SNS_TOPIC_ARN
            }, publishComplete);
        },
        next
    );
},
function (err) {
    if (err) {
        console.error('Failed to publish notifications: ', err);
    } else {
        console.log('Successfully published all notifications.');
    }
    callback(null,"message");
});
};
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Utilisation d'AWS Lambda avec Amazon SNS à partir de différents comptes

Pour réaliser des livraisons Amazon SNS entre comptes vers Lambda, vous devez autoriser l'appel de la fonction Lambda à partir d'Amazon SNS. De même, Amazon SNS doit autoriser le compte Lambda à s'abonner à la rubrique Amazon SNS. Par exemple, si la rubrique Amazon SNS se trouve dans un compte A et la fonction Lambda dans un compte B, les deux comptes doivent s'accorder des autorisations mutuelles pour accéder à leurs ressources respectives. Dans la mesure où les options de configuration des

autorisations entre comptes ne sont pas toutes disponibles à partir de la console AWS, utilisez AWS CLI pour configurer l'ensemble du processus.

Rubriques

- [Didacticiel : Utilisation d'AWS Lambda avec Amazon Simple Notification Service \(p. 182\)](#)
- [Exemple de code de fonction \(p. 184\)](#)

Didacticiel : Utilisation d'AWS Lambda avec Amazon Simple Notification Service

Vous pouvez utiliser une fonction Lambda dans un compte AWS de sorte qu'elle s'abonne à une rubrique Amazon SNS dans un compte AWS distinct. Dans ce didacticiel, vous utiliserez l'AWS Command Line Interface pour effectuer des opérations AWS Lambda comme la création d'une fonction Lambda, la création d'une rubrique Amazon SNS et l'ajout des autorisations permettant à ces deux ressources d'accéder l'une à l'autre.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Dans le didacticiel, vous utilisez deux comptes. Les commandes d'AWS CLI illustrent cela en utilisant deux profils nommés et configurés pour être utilisés avec un compte différent. Si vous utilisez des profils aux noms différents ou le profil par défaut et un profil nommé, modifiez les commandes si nécessaire.

Créer une rubrique Amazon SNS

À partir du compte A, créez la rubrique Amazon SNS source.

```
$ aws sns create-topic --name lambda-x-account --profile accountA
```

Notez l'ARN de la rubrique qui est renvoyé par la commande. Vous en aurez besoin lorsque vous ajouterez des autorisations afin de permettre à la fonction Lambda de s'abonner à cette rubrique.

Créer le rôle d'exécution

À partir du compte B, créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.

2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaBasicExecutionRole.
 - Nom de rôle – **lambda-sns-role**.

La stratégie AWSLambdaBasicExecutionRole possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Créez une fonction Lambda

Dans le compte B, créez la fonction qui traite des événements depuis Amazon SNS. L'exemple de code suivant reçoit une entrée d'événement Amazon SNS et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction \(p. 184\)](#).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
// console.log('Received event:', JSON.stringify(event, null, 4));

    var message = event.Records[0].Sns.Message;
    console.log('Message received from SNS:', message);
    callback(null, "Success");
};
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name SNS-X-Account \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::01234567891B:role/service-role/lambda-sns-execution-role \
--timeout 60 --profile accountB
```

Notez l'ARN de la fonction qui est renvoyé par la commande. Vous en aurez besoin lorsque vous ajouterez les autorisations permettant à Amazon SNS d'appeler votre fonction.

Configuration des autorisations entre comptes

À partir du compte A, accordez l'autorisation permettant au compte B de s'abonner à la rubrique :

```
$ aws sns add-permission --label lambda-access --aws-account-id 12345678901B \
```

```
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--action-name Subscribe ListSubscriptionsByTopic Receive --profile accountA
```

A partir du compte B, ajoutez l'autorisation Lambda permettant d'appeler la fonction &LAM; depuis Amazon SNS.

```
$ aws lambda add-permission --function-name SNS-X-Account \
--source-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--statement-id sns-x-account --action "lambda:InvokeFunction" \
--principal sns.amazonaws.com --profile accountB
{
    "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:lambda:us-east-2:12345678901B:function:SNS-X-Account\"}},\"Action\":[\"lambda:InvokeFunction\"],\"Resource\":\"arn:aws:lambda:us-east-2:01234567891A:function:SNS-X-Account\", \"Effect\":\"Allow\", \"Principal\":{\"Service\":\"sns.amazonaws.com\"}, \"Sid\":\"sns-x-account1\"}"
}
```

N'utilisez pas le paramètre --source-account pour ajouter un compte source à la stratégie Lambda lors de la création de la stratégie. Le compte source n'est pas pris en charge pour les sources d'événements Amazon SNS et entraînera le refus de l'accès.

Créez un abonnement

À partir du compte B, abonnez la fonction Lambda à la rubrique. Lorsqu'un message est envoyé à la rubrique lambda-x-account dans le compte A, Amazon SNS appelle la fonction SNS-X-Account dans le compte B.

```
$ aws sns subscribe --protocol lambda \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--notification-endpoint arn:aws:lambda:us-east-2:12345678901B:function:SNS-X-Account \
--profile accountB
{
    "SubscriptionArn": "arn:aws:sns:us-east-2:12345678901A:lambda-x-account:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

La sortie contient l'ARN d'abonnement à la rubrique.

Tester l'abonnement

À partir du compte A, testez l'abonnement. Tapez Hello World dans un fichier texte et enregistrez-le sous message.txt. Ensuite, exécutez la commande suivante :

```
$ aws sns publish --message file://message.txt --subject Test \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--profile accountA
```

Cela renverra un ID de message avec un identifiant unique, qui indique que le message a été acceptée par l'Amazon SNS. Ensuite, Amazon SNS va essayer de le remettre aux abonnés de la rubrique. Vous avez également la possibilité de fournir une chaîne JSON directement au paramètre message, mais l'utilisation d'un fichier texte prend en charge les sauts de ligne dans le message.

Pour en savoir plus sur Amazon SNS, consultez [Présentation d'Amazon Simple Notification Service](#).

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js 8 \(p. 185\)](#)
- [Java 8 \(p. 185\)](#)
- [Go \(p. 186\)](#)
- [Python 3 \(p. 186\)](#)

Node.js 8

L'exemple suivant traite les messages depuis Amazon SNS et enregistre leur contenu.

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
// console.log('Received event:', JSON.stringify(event, null, 4));

    var message = event.Records[0].Sns.Message;
    console.log('Message received from SNS:', message);
    callback(null, "Success");
};
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Java 8

L'exemple suivant traite les messages depuis Amazon SNS et enregistre leur contenu.

Example LambdaWithSNS.java

```
package example;

import java.text.SimpleDateFormat;
import java.util.Calendar;

import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;

public class LogEvent implements RequestHandler<SNSEvent, Object> {
    public Object handleRequest(SNSEvent request, Context context){
        String timeStamp = new SimpleDateFormat("yyyy-MM-
dd_HH:mm:ss").format(Calendar.getInstance().getTime());
        context.getLogger().log("Invocation started: " + timeStamp);
        context.getLogger().log(request.getRecords().get(0).getSNS().getMessage());

        timeStamp = new SimpleDateFormat("yyyy-MM-
dd_HH:mm:ss").format(Calendar.getInstance().getTime());
        context.getLogger().log("Invocation completed: " + timeStamp);
        return null;
    }
}
```

Dependencies

- [aws-lambda-java-core](#)
- [aws-lambda-java-events](#)

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Go

L'exemple suivant traite les messages depuis Amazon SNS et enregistre leur contenu.

Example `lambda_handler.go`

```
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/events"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        snsRecord := record.SNS
        fmt.Printf("[%s %s] Message = %s \n", record.EventSource, snsRecord.Timestamp,
        snsRecord.Message)
    }
}

func main() {
    lambda.Start(handler)
}
```

Build the executable with `go build` and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Go \(p. 344\)](#).

Python 3

L'exemple suivant traite les messages depuis Amazon SNS et enregistre leur contenu.

Example `lambda_handler.py`

```
from __future__ import print_function
import json
print('Loading function')

def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    message = event['Records'][0]['Sns']['Message']
    print("From SNS: " + message)
    return message
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Utilisation de AWS Lambda avec Amazon API Gateway

Vous pouvez appeler les fonctions AWS Lambda via HTTPS. Pour ce faire, vous devez définir une API REST personnalisée et le point de terminaison via [Amazon API Gateway](#), puis mapper des méthodes

individuelles, comme `GET` et `PUT`, vers des fonctions Lambda. Sinon, vous pouvez ajouter une méthode spéciale nommée `ANY` pour mapper toutes les méthodes prises en charge (`GET`, `POST`, `PATCH`, `DELETE`) vers la fonction Lambda. Lorsque vous envoyez une requête HTTPS au point de terminaison de l'API, le service Amazon API Gateway appelle la fonction Lambda correspondante. Pour plus d'informations sur la méthode `ANY`, consultez [Création d'un microservice simple avec Lambda et API Gateway \(p. 198\)](#).

Amazon API Gateway ajoute également un niveau entre les utilisateurs et la logique de votre application, ce qui offre les avantages suivants :

- Possibilité de bloquer des requêtes ou des utilisateurs individuels
- Protection contre les attaques par déni de service
- Couche de mise en cache de la réponse à partir de la fonction Lambda

Notez les considérations suivantes concernant l'intégration d'Amazon API Gateway et d'AWS Lambda :

- Modèle d'événement Push – Avec ce modèle (consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#)), Amazon API Gateway appelle la fonction Lambda.
- Appel synchrone – Le service Amazon API Gateway peut appeler la fonction Lambda et recevoir une réponse en temps réel en spécifiant le type d'appel `RequestResponse`. Pour plus d'informations sur les types d'appel, consultez la section [Types d'appel \(p. 74\)](#).
- Structure de l'événement – L'événement que la fonction Lambda reçoit est le corps de la requête HTTPS qu'Amazon API Gateway reçoit, tandis que la fonction Lambda est le code personnalisé écrit pour traiter ce type d'événement spécifique.

Notez que deux types de stratégies d'autorisations sont à votre disposition lorsque vous configurez l'environnement complet :

- Autorisations de la fonction Lambda – Quelle que soit l'origine de l'appel de la fonction Lambda, AWS Lambda exécute cette fonction en assumant le rôle IAM (rôle d'exécution) que vous spécifiez lors de sa création. Avec la stratégie d'autorisations associée à ce rôle, vous accordez à la fonction Lambda les autorisations dont elle a besoin. Par exemple, si la fonction Lambda doit lire un objet, vous accordez les autorisations requises pour les actions Amazon S3 correspondantes dans la stratégie d'autorisations. Pour plus d'informations, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).
- Autorisation permettant à Amazon API Gateway d'appeler la fonction Lambda – Amazon API Gateway ne peut pas appeler la fonction Lambda sans votre autorisation. Vous devez donc mettre à jour en conséquence la stratégie d'autorisations associée à cette fonction.

Didacticiel : Utilisation d'AWS Lambda avec Amazon API Gateway

Dans cet exemple, vous allez créer une API simple à l'aide d'Amazon API Gateway. Une entité Amazon API Gateway représente un ensemble de ressources et de méthodes. Pour les besoins de ce didacticiel, vous créerez une seule ressource (`DynamoDBManager`) et y définirez une seule méthode (`POST`). Cette méthode sera basée sur une fonction Lambda (`LambdaFunctionOverHttps`). Autrement dit, lorsque vous appelez l'API via un point de terminaison HTTPS, Amazon API Gateway appelle la fonction Lambda.

La méthode `POST` de la ressource `DynamoDBManager` prend en charge les opérations d'DynamoDB suivantes :

- Créer, mettre à jour et supprimer un élément
- Lire un élément

- Analyser un élément
- D'autres opérations (écho, ping), non liées à DynamoDB, que vous pouvez utiliser pour les tests

La charge utile que vous envoyez dans la requête POST identifie l'opération DynamoDB et fournit les données nécessaires. Par exemple :

- Voici un exemple de charge utile de requête pour une opération DynamoDB de création d'un élément :

```
{  
    "operation": "create",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Item": {  
            "id": "1",  
            "name": "Bob"  
        }  
    }  
}
```

- Voici un exemple de charge utile de requête pour une opération DynamoDB de lecture d'un élément :

```
{  
    "operation": "read",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Key": {  
            "id": "1"  
        }  
    }  
}
```

- Voici un exemple de charge utile de demande pour une opération echo. Vous envoyez une requête HTTP POST au point de terminaison, en utilisant les données suivantes dans le corps de la requête.

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

Note

API Gateway offre des fonctionnalités avancées, telles que les fonctionnalités suivantes :

- Transmission de l'ensemble de la requête – Une fonction Lambda peut recevoir l'ensemble de la requête HTTP (et non juste le corps de la requête) et définir la réponse HTTP (et non juste le corps de la réponse) à l'aide du type d'intégration AWS_PROXY.
- Méthodes fourre-tout – Mappe toutes les méthodes d'une ressource d'API vers une seule fonction Lambda avec un mappage unique à l'aide de la méthode fourre-tout ANY.
- Ressources fourre-tout – Mappe tous les sous-chemins d'une ressource vers une fonction Lambda sans configuration supplémentaire à l'aide du nouveau paramètre de chemin ({proxy +}).

Pour en savoir plus sur ces fonctions de passerelle d'API, consultez [Configuration de l'intégration de proxy pour une ressource de proxy](#).

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Nom de rôle – **lambda-apigateway-role**.
 - Autorisations – stratégie personnalisée avec l'autorisation pour DynamoDB et CloudWatch Logs.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Stmt1428341300017",  
            "Action": [  
                "dynamodb>DeleteItem",  
                "dynamodb>GetItem",  
                "dynamodb>PutItem",  
                "dynamodb>Query",  
                "dynamodb>Scan",  
                "dynamodb>UpdateItem"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        },  
        {  
            "Sid": "",  
            "Resource": "*",  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs>PutLogEvents"  
            ],  
            "Effect": "Allow"  
        }  
    ]  
}
```

```
    ]  
}
```

La stratégie personnalisée possède les autorisations dont la fonction a besoin pour écrire des données dans DynamoDB et charger les journaux. Notez le nom Amazon Resource Name (ARN) du rôle pour une utilisation ultérieure.

Créer la fonction

L'exemple de code suivant reçoit une entrée d'événement Kinesis et traite les messages qu'elle contient. Pour illustrer ce propos, le code écrit certaines des données d'événement entrantes dans CloudWatch Logs.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction \(p. 196\)](#).

Example index.js

```
console.log('Loading function');

var AWS = require('aws-sdk');
var dynamo = new AWS.DynamoDB.DocumentClient();

/**
 * Provide an event that contains the following keys:
 *
 * - operation: one of the operations in the switch statement below
 * - tableName: required for operations that interact with DynamoDB
 * - payload: a parameter to pass to the operation being performed
 */
exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    var operation = event.operation;

    if (event.tableName) {
        event.payload.TableName = event.tableName;
    }

    switch (operation) {
        case 'create':
            dynamo.put(event.payload, callback);
            break;
        case 'read':
            dynamo.get(event.payload, callback);
            break;
        case 'update':
            dynamo.update(event.payload, callback);
            break;
        case 'delete':
            dynamo.delete(event.payload, callback);
            break;
        case 'list':
            dynamo.scan(event.payload, callback);
            break;
        case 'echo':
            callback(null, "Success");
            break;
        case 'ping':
```

```
        callback(null, "pong");
        break;
    default:
        callback('Unknown operation: ${operation}');
    }
};
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

Test de la fonction Lambda

Appelez la fonction manuellement à l'aide de l'échantillon de données d'événement. Nous vous recommandons d'appeler la fonction à l'aide de la console, car son interface facilite l'examen des résultats de l'exécution, y compris le résumé de l'exécution, les journaux écrits par votre code et les résultats renvoyés par la fonction (parce que la console effectue toujours l'exécution synchrone : elle appelle la fonction Lambda avec le type d'appel `RequestResponse`).

Pour tester la fonction Lambda

1. Copiez le code JSON suivant dans un fichier et enregistrez-le sous le nom `input.txt`.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Exécutez la commande `invoke` suivante :

```
$ aws lambda invoke --function-name LambdaFunctionOverHttps \
--payload fileb://input.txt outputfile.txt
```

Créer une API à l'aide d'Amazon API Gateway

Au cours de cette étape, vous associez la fonction Lambda à une méthode dans l'API que vous avez créée via Amazon API Gateway, puis vous testez l'environnement complet. Autrement dit, lorsqu'une requête HTTP est envoyée à une méthode d'API, Amazon API Gateway appelle votre fonction Lambda.

Premièrement, vous créez une API (`DynamoDBOperations`) via Amazon API Gateway avec une seule ressource (`DynamoDBManager`) et une seule méthode (`POST`). Vous associez la méthode `POST` à la fonction Lambda. Vous testez ensuite l'environnement complet.

Créer l'API

Exécutez la commande `create-rest-api` suivante pour créer l'API `DynamoDBOperations` pour ce didacticiel.

```
$ aws apigateway create-rest-api --name DynamoDBOperations
{
    "id": "bs8fqo6bp0",
    "name": "DynamoDBOperations",
    "createdDate": 1539803980,
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "EDGE"
        ]
    }
}
```

Enregistrez l'ID de l'API pour l'utiliser dans des commandes ultérieures. Vous aurez également besoin de l'ID de ressource racine de l'API. Pour l'obtenir, exécutez la commande `get-resources`.

```
$ API=bs8fqo6bp0
$ aws apigateway get-resources --rest-api-id $API
{
    "items": [
        {
            "path": "/",
            "id": "e8kitthgdb"
        }
    ]
}
```

À ce stade, vous n'avez que la ressource racine, mais vous ajoutez davantage de ressources dans l'étape suivante.

Créer une ressource dans l'API

Exécutez la commande `create-resource` suivante pour créer une ressource (`DynamoDBManager`) dans l'API que vous avez conçue dans la section précédente.

```
$ aws apigateway create-resource --rest-api-id $API --path-part DynamoDBManager \
--parent-id e8kitthgdb
{
    "path": "/DynamoDBManager",
    "pathPart": "DynamoDBManager",
    "id": "resource-id",
    "parentId": "e8kitthgdb"
}
```

Notez l'ID de la réponse. Il s'agit de l'ID de la ressource `DynamoDBManager` que vous avez créée.

Créer une méthode POST sur la ressource

Exécutez la commande `put-method` suivante pour créer une méthode `POST` sur la ressource `DynamoDBManager` dans votre API.

```
$ RESOURCE=iuig5w
$ aws apigateway put-method --rest-api-id $API --resource-id $RESOURCE \
--http-method POST --authorization-type NONE
{
```

```
    "apiKeyRequired": false,  
    "httpMethod": "POST",  
    "authorizationType": "NONE"  
}
```

Nous spécifions NONE pour le paramètre --authorization-type, ce qui signifie que les requêtes non authentifiées pour cette méthode sont prises en charge. Cette configuration est appropriée pour les tests, mais en production, vous devez utiliser soit l'authentification basée sur des clés ou sur des rôles.

Définir la fonction Lambda comme destination de la méthode POST

Exécutez la commande suivante pour définir la fonction Lambda, comme point d'intégration de la méthode POST. Il s'agit de la méthode qu'Amazon API Gateway appelle lorsque vous créez une requête HTTP pour le point de terminaison de la méthode POST. Cette commande et d'autres utilisent des ARN qui incluent votre ID de compte et votre région. Enregistrez-les dans des variables (votre ID de compte figure dans l'ARN du rôle que vous avez utilisé pour créer la fonction).

```
$ REGION=us-east-2  
$ ACCOUNT=123456789012  
$ aws apigateway put-integration --rest-api-id $API --resource-id $RESOURCE \  
--http-method POST --type AWS --integration-http-method POST \  
--uri arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/arn:aws:lambda:$REGION:  
$ACCOUNT:function:LambdaFunctionOverHttps/invocations  
{  
    "type": "AWS",  
    "httpMethod": "POST",  
    "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/  
arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps/invocations",  
    "passthroughBehavior": "WHEN_NO_MATCH",  
    "timeoutInMillis": 29000,  
    "cacheNamespace": "iuig5w",  
    "cacheKeyParameters": []  
}
```

--integration-http-method est la méthode qu'API Gateway utilise pour communiquer avec AWS Lambda. --uri est un identifiant unique du point de terminaison auquel Amazon API Gateway peut envoyer la requête.

Définissez la section content-type de la réponse de la méthode POST et la réponse d'intégration au format JSON, comme suit :

- Exécutez la commande suivante pour définir la réponse de la méthode POST au format JSON. Voici le type de réponse que votre méthode d'API renvoie.

```
$ aws apigateway put-method-response --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST \  
--status-code 200 --response-models application/json=Empty  
{  
    "statusCode": "200",  
    "responseModels": {  
        "application/json": "Empty"  
    }  
}
```

- Exécutez la commande suivante pour définir la réponse d'intégration de la méthode POST au format JSON. Voici le type de réponse renvoyé par la fonction Lambda.

```
$ aws apigateway put-integration-response --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST \  
--status-code 200 --response-templates application/json=""
```

```
{  
    "statusCode": "200",  
    "responseTemplates": {  
        "application/json": null  
    }  
}
```

Déploiement de l'API

Dans cette étape, vous déployez l'API que vous avez créée dans un environnement appelé `prod`.

```
$ aws apigateway create-deployment --rest-api-id $API --stage-name prod  
{  
    "id": "20vgsz",  
    "createdDate": 1539820012  
}
```

Attribuer une autorisation d'appel à l'API

Maintenant que vous avez créé une API via Amazon API Gateway et que vous l'avez déployée, vous pouvez la tester. Tout d'abord, vous devez ajouter des autorisations afin de permettre à Amazon API Gateway d'appeler votre fonction Lambda lorsque vous envoyez une requête HTTP à la méthode `POST`.

Pour ce faire, vous devez ajouter les autorisations requises à la stratégie d'autorisations associée à votre fonction Lambda. Exécutez la commande `add-permission` AWS Lambda suivante pour accorder au service Amazon API Gateway (`apigateway.amazonaws.com`) les autorisations principales requises pour appeler la fonction Lambda (`LambdaFunctionOverHttps`).

```
$ aws lambda add-permission --function-name LambdaFunctionOverHttps \  
--statement-id apigateway-test-2 --action lambda:InvokeFunction \  
--principal apigateway.amazonaws.com \  
--source-arn "arn:aws:execute-api:$REGION:$ACCOUNT:$API/*/POST/DynamoDBManager"  
{  
    "Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal\":  
    \"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",  
    \"Resource\":\"arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps\",  
    \"Condition\":\"ArnLike:{AWS:SourceArn}:arn:aws:execute-api:us-east-2:123456789012:mnh1yprki7/  
    */POST/DynamoDBManager\"}}}"  
}
```

Vous devez accorder cette autorisation pour permettre les tests (si vous accédez à l'Amazon API Gateway et si vous choisissez Test pour tester la méthode de l'API, cette autorisation est nécessaire). Notez que la valeur `--source-arn` spécifie un caractère générique (*) comme valeur d'environnement (ce qui indique l'environnement de test uniquement). Cela vous permet de réaliser les tests sans avoir à déployer l'API.

Maintenant, exécutez à nouveau la même commande, mais cette fois-ci, vous accordez à l'API déployée les autorisations requises pour appeler la fonction Lambda.

```
$ aws lambda add-permission --function-name LambdaFunctionOverHttps \  
--statement-id apigateway-prod-2 --action lambda:InvokeFunction \  
--principal apigateway.amazonaws.com \  
--source-arn "arn:aws:execute-api:$REGION:$ACCOUNT:$API/prod/POST/DynamoDBManager"  
{  
    "Statement": "{\"Sid\":\"apigateway-prod-2\",\"Effect\":\"Allow\",\"Principal\":  
    \"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",  
    \"Resource\":\"arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps\",  
    \"Condition\":\"ArnLike:{AWS:SourceArn}:arn:aws:execute-api:us-east-2:123456789012:mnh1yprki7/  
    prod/POST/DynamoDBManager\"}}}"  
}
```

}

Vous accordez cette autorisation pour permettre à l'API déployée d'appeler la fonction Lambda. Notez que la valeur `--source-arn` indique `prod`, qui est le nom d'environnement que nous avons utilisé lors du déploiement de l'API.

Créer une table Amazon DynamoDB

Créez la table DynamoDB que la fonction Lambda utilise.

Pour créer une table DynamoDB

1. Ouvrez la [console DynamoDB](#).
2. Choisissez Create table (Créer une table).
3. Créez une table avec les paramètres suivants.
 - Nom de la table – **lambda-apigateway**
 - Clé primaire – **id** (chaîne)
4. Sélectionnez Create.

Déclencher la fonction avec une requête HTTP

Dans cette étape, vous êtes prêt à envoyer une requête HTTP au point de terminaison de la méthode `POST`. Vous pouvez utiliser Curl ou une méthode (`testInvokeMethod`) fournie par Amazon API Gateway.

Vous pouvez utiliser les commandes d'interface de ligne de commande Amazon API Gateway pour envoyer une requête HTTP `POST` au point de terminaison de la ressource (`DynamoDBManager`). Etant donné que vous avez déployé votre instance Amazon API Gateway, vous pouvez utiliser Curl pour appeler les méthodes pour la même opération.

La fonction Lambda prend en charge l'utilisation de l'opération `create` pour créer un élément dans la table d'DynamoDB. Pour demander cette opération, utilisez le code JSON suivant :

Example `create-item.json`

```
{  
    "operation": "create",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Item": {  
            "id": "1234ABCD",  
            "number": 5  
        }  
    }  
}
```

Enregistrez l'entrée de test dans un fichier nommé `create-item.json`. Exécutez la commande `testInvokeMethod` Amazon API Gateway pour envoyer une requête HTTP de méthode `POST` au point de terminaison de la ressource (`DynamoDBManager`).

```
$ aws apigateway testInvokeMethod --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \  
--body file://create-item.json
```

Vous pouvez également utiliser la commande Curl suivante :

```
$ curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-apigateway\", \"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}" https://$API.execute-api.$REGION.amazonaws.com/prod/DynamoDBManager
```

Pour envoyer la requête pour l'opération echo prise en charge par la fonction Lambda, vous pouvez utiliser la charge utile suivante :

Example echo.json

```
{  
  "operation": "echo",  
  "payload": {  
    "somekey1": "somevalue1",  
    "somekey2": "somevalue2"  
  }  
}
```

Enregistrez l'entrée de test dans un fichier nommé echo.json. Exécutez la commande d'interface de ligne de commande test-invoke-method Amazon API Gateway pour envoyer une requête HTTP de méthode POST au point de terminaison de la ressource (DynamoDBManager) en utilisant le JSON précédent dans le corps de la requête.

```
$ aws apigateway test-invoke-method --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \  
--body file://echo.json
```

Vous pouvez également utiliser la commande Curl suivante :

```
$ curl -X POST -d "{\"operation\":\"echo\",\"payload\":[\"somekey1\":\"somevalue1\", \"somekey2\":\"somevalue2\"]}" https://$API.execute-api.$REGION.amazonaws.com/prod/DynamoDBManager
```

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 196\)](#)
- [Python 3 \(p. 197\)](#)
- [Go \(p. 198\)](#)

Node.js

L'exemple suivant traite les messages à partir d'API Gateway et gère les documents DynamoDB en fonction de la méthode de requête.

Example index.js

```
console.log('Loading function');  
  
var AWS = require('aws-sdk');  
var dynamo = new AWS.DynamoDB.DocumentClient();  
  
/**  
 * Provide an event that contains the following keys:  
 *
```

```
*   - operation: one of the operations in the switch statement below
*   - tableName: required for operations that interact with DynamoDB
*   - payload: a parameter to pass to the operation being performed
*/
exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    var operation = event.operation;

    if (event.tableName) {
        event.payload.TableName = event.tableName;
    }

    switch (operation) {
        case 'create':
            dynamo.put(event.payload, callback);
            break;
        case 'read':
            dynamo.get(event.payload, callback);
            break;
        case 'update':
            dynamo.update(event.payload, callback);
            break;
        case 'delete':
            dynamo.delete(event.payload, callback);
            break;
        case 'list':
            dynamo.scan(event.payload, callback);
            break;
        case 'echo':
            callback(null, "Success");
            break;
        case 'ping':
            callback(null, "pong");
            break;
        default:
            callback('Unknown operation: ${operation}');
    }
};
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Python 3

L'exemple suivant traite les messages à partir d'API Gateway et gère les documents DynamoDB en fonction de la méthode de requête.

Example LambdaFunctionOverHttps.py

```
from __future__ import print_function

import boto3
import json

print('Loading function')


def handler(event, context):
    '''Provide an event that contains the following keys:
       - operation: one of the operations in the operations dict below
       - tableName: required for operations that interact with DynamoDB
```

```
    - payload: a parameter to pass to the operation being performed
    ''
#printf("Received event: " + json.dumps(event, indent=2))

operation = event['operation']

if 'tableName' in event:
    dynamo = boto3.resource('dynamodb').Table(event['tableName'])

operations = {
    'create': lambda x: dynamo.put_item(**x),
    'read': lambda x: dynamo.get_item(**x),
    'update': lambda x: dynamo.update_item(**x),
    'delete': lambda x: dynamo.delete_item(**x),
    'list': lambda x: dynamo.scan(**x),
    'echo': lambda x: x,
    'ping': lambda x: 'pong'
}

if operation in operations:
    return operations[operation](event.get('payload'))
else:
    raise ValueError('Unrecognized operation "{}".format(operation))
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Python \(p. 301\)](#).

Go

L'exemple suivant traite les messages depuis API Gateway et consigne des informations sur la requête.

Example LambdaFunctionOverHttps.go

```
import (
    "strings"
    "github.com/aws/aws-lambda-go/events"
)

func handleRequest(ctx context.Context, request events.APIGatewayProxyRequest)
(events.APIGatewayProxyResponse, error) {
    fmt.Printf("Processing request data for request %s.\n",
    request.RequestContext.RequestId)
    fmt.Printf("Body size = %d.\n", len(request.Body))

    fmt.Println("Headers:")
    for key, value := range request.Headers {
        fmt.Printf("    %s: %s\n", key, value)
    }

    return events.APIGatewayProxyResponse { Body: request.Body, StatusCode: 200 }, nil
}
```

Build the executable with `go build` and create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Go \(p. 344\)](#).

Création d'un microservice simple avec Lambda et API Gateway

Dans ce didacticiel, vous allez utiliser la console Lambda pour créer une fonction Lambda, et un point de terminaison Amazon API Gateway pour déclencher cette fonction. Vous pourrez appeler le point de

terminaison avec n'importe quelle méthode (GET, POST, PATCH, etc.) pour déclencher votre fonction Lambda. Lorsque le point de terminaison est appelé, la totalité de la demande est transmise à votre fonction Lambda. L'action de votre fonction dépend de la méthode avec laquelle vous appelez votre point de terminaison :

- DELETE : suppression d'un élément d'une table DynamoDB
- GET : analyse de la table et renvoie de tous les éléments
- POST : création d'un élément
- PUT : mise à jour d'un élément

Créer une API à l'aide d'Amazon API Gateway

Suivez les étapes de cette section pour créer une fonction Lambda et un point de terminaison API Gateway pour la déclencher :

Pour créer une API

1. Connectez-vous à AWS Management Console et ouvrez la console AWS Lambda.
2. Choisissez Create a Lambda function.
3. Choisissez Blueprint.
4. Entrez **microservice** dans la barre de recherche. Choisissez le modèle **microservice-http-endpoint**, puis choisissez Configurer.
5. 以下を設定します。
 - Nom – **lambda-microservice**.
 - Rôle – Créez un nouveau rôle à partir d'un ou de plusieurs modèles.
 - Nom de rôle – **lambda-apigateway-role**.
 - Modèles de stratégie – Autorisations Microservice.
 - API – Créer une nouvelle API.
 - Sécurité – Ouvrir.

Sélectionnez Create function.

Lorsque vous terminez l'assistant et créez votre fonction, Lambda crée une ressource de proxy nommée **lambda-microservice** sous le nom d'API que vous avez sélectionné. Pour plus d'informations sur les ressources de proxy, consultez [Configuration de l'intégration de proxy pour une ressource de proxy](#).

Une ressource de proxy a un type d'intégration **AWS_PROXY** et une méthode fourre-tout **ANY**. Le type d'intégration **AWS_PROXY** applique un modèle de mappage par défaut pour transmettre l'ensemble de la demande à la fonction Lambda et transforme la sortie de cette dernière en réponses HTTP. La méthode **ANY** définit la même configuration d'intégration pour toutes les méthodes prises en charge, y compris **GET**, **POST**, **PATCH**, **DELETE** , etc.

Tester l'envoi d'une requête HTTPS

Dans cette étape, vous allez utiliser la console pour tester la fonction Lambda. En outre, vous pouvez exécuter une commande `curl` pour tester l'expérience de bout en bout. Autrement dit, vous envoyez une requête HTTPS à votre méthode d'API et demandez à Amazon API Gateway d'appeler la fonction Lambda. Pour terminer la procédure, assurez-vous que vous avez créé une table DynamoDB et nommez-la « MyTable ». Pour plus d'informations, consultez [Créer une table DynamoDB avec un flux activé \(p. 167\)](#)

Pour tester l'API

1. Avec votre fonction MyLambdaMicroService toujours ouverte dans la console, sélectionnez l'onglet Actions, puis Configure test event.
2. Remplacez le texte existant par le suivant :

```
{  
    "httpMethod": "GET",  
    "queryStringParameters": {  
        "TableName": "MyTable"  
    }  
}
```

3. Après avoir entré le texte ci-dessus, choisissez Save and test.

Modèle AWS SAM pour une application API Gateway

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda à partir du [didacticiel \(p. 187\)](#). Copiez le texte ci-dessous dans un fichier .yaml et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre Handler et Runtime doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  LambdaFunctionOverHttps:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: index.handler  
      Runtime: nodejs8.10  
      Policies: AmazonDynamoDBFullAccess  
      Events:  
        HttpPost:  
          Type: Api  
          Properties:  
            Path: '/DynamoDBOperations/DynamoDBManager'  
            Method: post
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Utilisation d'AWS Lambda en tant que backend pour applications mobiles Android

Vous pouvez utiliser AWS Lambda pour héberger la logique back-end des applications mobiles. Autrement dit, vous pouvez exécuter une partie de code de votre application mobile en tant que fonctions Lambda. Cette approche vous permet d'insérer une logique minimale dans l'application mobile elle-même, ce qui

facilite l'ajustement des capacités et la mise à jour (par exemple, vous appliquez uniquement les mises à jour de code à la fonction Lambda au lieu de les déployer vos clients d'application).

Après avoir créé la fonction Lambda, vous pouvez l'appeler à partir de votre application mobile à l'aide des kits SDK AWS Mobile, tels que le kit SDK AWS pour Android. Pour plus d'informations, consultez [Outils pour Amazon web Services](#).

Note

Vous pouvez également appeler votre fonction Lambda via HTTP à l'aide d'Amazon API Gateway, au lieu d'utiliser le kit SDK AWS. Amazon API Gateway ajoute une couche supplémentaire entre vos utilisateurs mobiles et votre logique d'application, ce qui offre les avantages suivants :

- Possibilité de bloquer des requêtes ou des utilisateurs individuels
- Protection contre les attaques par déni de service
- Couche de mise en cache de la réponse à partir de la fonction Lambda

Pour plus d'informations, consultez [Utilisation de AWS Lambda avec Amazon API Gateway \(p. 186\)](#).

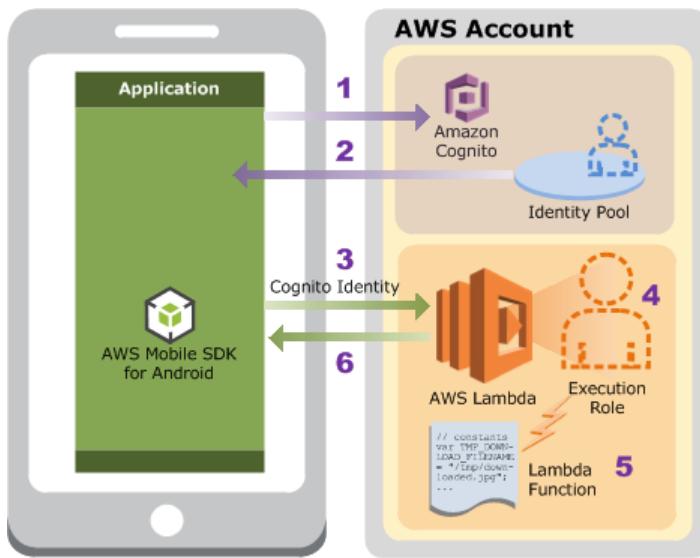
Notez les considérations suivantes concernant l'intégration de l'application mobile et d'AWS Lambda :

- Modèle d'événement Push – Dans ce modèle (consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#)), l'application appelle la fonction Lambda en transmettant les données d'événement en tant que paramètre.
- Appel synchrone ou asynchrone – L'application peut appeler la fonction Lambda et recevoir une réponse en temps réel en spécifiant le type d'appel `RequestResponse` (ou `Event` pour l'appel asynchrone). Pour plus d'informations sur les types d'appel, consultez la section [Gestion des autorisations : utilisation d'une stratégie de fonction Lambda \(p. 262\)](#).
- Structure d'événement – L'événement que la fonction Lambda reçoit est défini par votre application, tandis que la fonction Lambda est le code personnalisé écrit pour traiter ce type d'événement spécifique.

Deux types de stratégies d'autorisations sont à votre disposition lorsque vous configurez l'expérience de bout en bout :

- Autorisations de la fonction Lambda – Quelle que soit l'origine de l'appel de la fonction Lambda, AWS Lambda exécute cette fonction en assumant le rôle IAM (rôle d'exécution) que vous spécifiez lors de sa création. Avec la stratégie d'autorisations associée à ce rôle, vous accordez à la fonction Lambda les autorisations dont elle a besoin. Par exemple, si la fonction Lambda doit lire un objet, vous accordez les autorisations requises pour les actions Amazon S3 correspondantes dans la stratégie d'autorisations. Pour plus d'informations, consultez [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).
- Autorisations permettant à l'application mobile d'appeler la fonction Lambda – L'application doit disposer des informations d'identification de sécurité valides et des autorisations pour appeler une fonction Lambda. Pour les applications mobiles, vous pouvez utiliser le service Amazon Cognito pour gérer les autorisations, l'authentification et les identités des utilisateurs.

Le schéma suivant illustre le flux de l'application (l'illustration suppose que l'application mobile utilise le kit SDK AWS Mobile pour Android pour effectuer des appels d'API) :



1. L'application mobile récupère les informations d'identification AWS à partir d'un groupe d'identités Amazon Cognito. Le rôle associé au groupe d'identités donne à l'application l'autorisation d'utiliser AWS Lambda.
2. L'application mobile appelle la fonction Lambda.
3. AWS Lambda renvoie les résultats à l'application mobile.

Rubriques

- [Didacticiel : Utilisation d'AWS Lambda en tant que backend pour applications mobiles \(p. 202\)](#)
- [Exemple de code de fonction \(p. 208\)](#)

Didacticiel : Utilisation d'AWS Lambda en tant que backend pour applications mobiles

Dans ce didacticiel, vous créez une application mobile Android simple. L'objectif principal de ce didacticiel est de vous montrer comment connecter différents composants pour permettre à une application mobile Android d'appeler une fonction Lambda et de traiter la réponse.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 261\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaBasicExecutionRole.
 - Nom de rôle – **lambda-android-role**.

La stratégie AWSLambdaBasicExecutionRole possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple suivant utilise des données pour générer une réponse de type chaîne.

Note

Pour obtenir des exemples en d'autres langages de programmation, consultez [Exemple de code de fonction \(p. 208\)](#).

Example index.js

```
exports.handler = function(event, context, callback) {
    console.log("Received event: ", event);
    var data = {
        "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
    };
    callback(null, data);
}
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name AndroidBackendLambdaFunction \
--zip-file file://function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn
```

Test de la fonction Lambda

Appelez la fonction manuellement à l'aide de l'échantillon de données d'événement.

Pour tester la fonction Lambda (AWS CLI)

1. Enregistrez l'exemple de code d'événement JSON suivant dans un fichier, `input.txt`.

```
{ "firstName": "first-name", "lastName": "last-name" }
```

2. Exécutez la commande `invoke` suivante :

```
$ aws lambda invoke --function-name AndroidBackendLambdaFunction \
--payload file://file-path/input.txt outfile.txt
```

Créer un groupe d'identités Amazon Cognito

Dans cette section, vous allez créer un groupe d'identités Amazon Cognito. Le groupe d'identités inclut deux rôles IAM. Vous mettez à jour le rôle IAM pour les utilisateurs non authentifiés et accordez les autorisations d'exécution de la fonction `AndroidBackendLambdaFunction` Lambda.

Pour plus d'informations sur les rôles IAM, consultez [Rôles IAM](#) dans le IAM Guide de l'utilisateur. Pour plus d'informations sur les services Amazon Cognito, consultez la page d'informations du produit [Amazon Cognito](#).

Pour créer un groupe d'identités

1. Ouvrez la [console Amazon Cognito](#).
2. Créez un groupe d'identités appelé `JavaFunctionAndroidEventHandlerPool`. Avant de suivre la procédure de création d'un groupe d'identités, notez les éléments suivants :
 - Le groupe d'identités que vous créez doit autoriser l'accès aux identités non authentifiées, car notre exemple d'application mobile ne requiert pas la connexion d'un utilisateur. Par conséquent, veillez à bien sélectionner l'option Activer l'accès aux identités non authentifiées.
 - Ajoutez l'instruction suivante à la stratégie d'autorisation associée aux identités non authentifiées.

```
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda:us-
east-1:123456789012:function:AndroidBackendLambdaFunction"
    ]
}
```

La stratégie générée apparaît comme suit :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "mobileanalytics:PutEvents",
                "cognito-sync:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

```
},
{
  "Effect": "Allow",
  "Action": [
    "lambda:invokefunction"
  ],
  "Resource": [
    "arn:aws:lambda:us-east-1:account-
id:function:AndroidBackendLambdaFunction"
  ]
}
}
```

Pour obtenir les instructions pour créer un groupe d'identités, connectez-vous à la [console Amazon Cognito](#) et suivez les instructions de l'assistant Nouveau groupe d'identités.

3. Notez l'ID du groupe d'identités. Vous spécifiez cet ID dans l'application mobile que vous créerez dans la section suivante. L'application utilise cet ID quand elle envoie des requêtes à Amazon Cognito afin de demander des informations d'identification de sécurité temporaires.

Créer une application Android

Créez une application mobile Android simple qui génère des événements et qui appelle les fonctions Lambda en transmettant les données d'événement en tant que paramètres.

Les instructions suivantes ont été validées via Android Studio.

1. Créez un projet Android appelé `AndroidEventGenerator` via la configuration suivante :

- Sélectionnez la plateforme Phone and Tablet.
- Sélectionnez Blank Activity.

2. Dans le fichier `build.gradle` (`Module:app`), ajoutez les éléments suivants dans la section `dependencies` :

```
compile 'com.amazonaws:aws-android-sdk-core:2.2.+'
compile 'com.amazonaws:aws-android-sdk-lambda:2.2.+'
```

3. Générez le projet afin de télécharger les dépendances requises, en fonction des besoins.

4. Dans le fichier manifeste de l'application Android (`AndroidManifest.xml`), ajoutez les autorisations suivantes pour permettre à votre application de se connecter à Internet. Vous pouvez les ajouter juste avant la balise de fin `</manifest>`.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

5. Dans `MainActivity`, ajoutez les importations suivantes :

```
import com.amazonaws.mobileconnectors.lambdainvoker.*;
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

6. Dans la section `package`, ajoutez les deux classes suivantes (`RequestClass` et `ResponseClass`). Notez que le type POJO est la même que celui que vous avez créé dans la fonction Lambda dans la section précédente.

- `RequestClass`. Les instances de cette classe jouent le rôle d'objet POJO (Plain Old Java Object) pour les données d'événement qui se composent du prénom et du nom de famille. Si vous utilisez

l'exemple Java pour la fonction Lambda que vous avez créée dans la section précédente, cet objet POJO est le même que celui que vous avez créé dans le code de cette fonction.

```
package com.example....lambdaeventgenerator;
public class RequestClass {
    String firstName;
    String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public RequestClass(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public RequestClass() {
    }
}
```

- ResponseClass

```
package com.example....lambdaeventgenerator;
public class ResponseClass {
    String greetings;

    public String getGreetings() {
        return greetings;
    }

    public void setGreetings(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass() {
    }
}
```

7. Dans le même package, créez une interface dénommée MyInterface pour appeler la fonction AndroidBackendLambdaFunction Lambda.

```
package com.example....lambdaeventgenerator;
import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;
public interface MyInterface {

    /**
     * This method is triggered when a request is made to the Lambda function.
     * It takes a String parameter and returns a String response.
     */
    String invoke(String input);
}
```

```
* Invoke the Lambda function "AndroidBackendLambdaFunction".  
* The function name is the method name.  
*/  
@LambdaFunction  
ResponseClass AndroidBackendLambdaFunction(RequestClass request);  
}
```

L'annotation `@LambdaFunction` dans le code mappe la méthode spécifique au client avec la fonction Lambda du même nom. Pour plus d'informations sur cette annotation, consultez [AWS Lambda](#) dans le AWS Mobile SDK pour Android Developer Guide.

8. Pour que l'application reste simple, nous allons ajouter du code afin d'appeler la fonction Lambda dans le gestionnaire d'événements `onCreate()`. Dans `MainActivity`, ajoutez le code suivant à la fin du code `onCreate()`.

```
// Create an instance of CognitoCachingCredentialsProvider  
CognitoCachingCredentialsProvider cognitoProvider = new  
    CognitoCachingCredentialsProvider(  
        this.getApplicationContext(), "identity-pool-id", Regions.US_WEST_2);  
  
// Create LambdaInvokerFactory, to be used to instantiate the Lambda proxy.  
LambdaInvokerFactory factory = new LambdaInvokerFactory(this.getApplicationContext(),  
    Regions.US_WEST_2, cognitoProvider);  
  
// Create the Lambda proxy object with a default Json data binder.  
// You can provide your own data binder by implementing  
// LambdaDataBinder.  
final MyInterface myInterface = factory.build(MyInterface.class);  
  
RequestClass request = new RequestClass("John", "Doe");  
// The Lambda function invocation results in a network call.  
// Make sure it is not called from the main thread.  
new AsyncTask<RequestClass, Void, ResponseClass>() {  
    @Override  
    protected ResponseClass doInBackground(RequestClass... params) {  
        // invoke "echo" method. In case it fails, it will throw a  
        // LambdaFunctionException.  
        try {  
            return myInterface.AndroidBackendLambdaFunction(params[0]);  
        } catch (LambdaFunctionException lfe) {  
            Log.e("Tag", "Failed to invoke echo", lfe);  
            return null;  
        }  
    }  
  
    @Override  
    protected void onPostExecute(ResponseClass result) {  
        if (result == null) {  
            return;  
        }  
  
        // Do a toast  
        Toast.makeText(MainActivity.this, result.getGreetings(),  
            Toast.LENGTH_LONG).show();  
    }  
}.execute(request);
```

9. Exécutez le code et vérifiez-le, comme suit :

- La section `Toast.makeText()` affiche la réponse renvoyée.

- Vérifiez qu'CloudWatch Logs affiche le journal créé par la fonction Lambda. Il doit indiquer les données d'événement (prénom et nom de famille). Vous pouvez également vérifier ces informations dans la console AWS Lambda.

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 208\)](#)
- [Java \(p. 208\)](#)

Node.js

L'exemple suivant utilise des données pour générer une réponse de type chaîne.

Example index.js

```
exports.handler = function(event, context, callback) {
    console.log("Received event: ", event);
    var data = {
        "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
    };
    callback(null, data);
}
```

Zip up the sample code to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

Java

L'exemple suivant utilise des données pour générer une réponse de type chaîne.

Dans le code, le handler (`myHandler`) utilise les types d'entrée et de sortie `RequestClass` et `ResponseClass`. Le code fournit la mise en œuvre pour ces types.

Example HelloPojo.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class HelloPojo {

    // Define two classes/POJOs for use with Lambda function.
    public static class RequestClass {
        String firstName;
        String lastName;

        public String getFirstName() {
            return firstName;
        }

        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }
    }

    public String handleRequest(RequestClass params, Context context) {
        String response = "Hello " + params.firstName + " " + params.lastName;
        return response;
    }
}
```

```
        }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public RequestClass(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public RequestClass() {
    }
}

public static class ResponseClass {
    String greetings;

    public String getGreetings() {
        return greetings;
    }

    public void setGreetings(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass() {
    }
}

public static ResponseClass myHandler(RequestClass request, Context context){
    String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
    context.getLogger().log(greetingString);
    return new ResponseClass(greetingString);
}
}
```

Dependencies

- `aws-lambda-java-core`

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Utilisation de AWS Lambda avec Amazon CloudWatch Events

Vous pouvez créer une fonction Lambda et demander à AWS Lambda de l'exécuter sur une base régulière. Vous pouvez spécifier un taux fixe (par exemple, exécution d'une fonction Lambda toutes les

heures ou toutes les 15 minutes) ou une expression Cron. Pour plus d'informations sur la planification des expressions, consultez la section [Planification des expressions à l'aide de la fréquence ou de cron \(p. 214\)](#).

Cette fonctionnalité est disponible lorsque vous créez une fonction Lambda à l'aide de la console AWS Lambda ou de l'AWS CLI. Pour la configurer avec l'AWS CLI, consultez [Exécution d'une fonction AWS Lambda de manière planifiée avec l'interface de ligne de commande AWS](#). La console fournit des CloudWatch Événements comme source d'événement. Au moment de la création d'une fonction Lambda, sélectionnez cette source d'événement et spécifiez un intervalle de temps.

Si vous avez apporté des modifications manuelles aux autorisations de votre fonction, vous devrez peut-être réappliquer l'accès de l'événement planifié à cette fonction. Pour ce faire, exécutez la commande d'interface suivante :

```
$ aws lambda add-permission --function-name function_name\n    --action 'lambda:InvokeFunction' --principal events.amazonaws.com \\n\n    --statement-id 'statement_id' \\n\n    --source-arn arn:aws:events:region:account-id:rule/rule_name
```

Chaque compte AWS peut avoir jusqu'à 100 sources d'événements uniques de type Événements CloudWatch Events - Calendrier. Chacune d'elles peut être la source d'événement de jusqu'à cinq fonctions Lambda. Autrement dit, vous pouvez exécuter jusqu'à 500 fonctions Lambda de manière planifiée dans votre compte AWS.

La console fournit également un plan (lambda-canary) qui utilise le type de source Événements CloudWatch Events - Calendrier. Grâce à ce plan, vous pouvez créer un exemple de fonction Lambda et tester cette fonctionnalité. L'exemple de code que le plan fournit recherche la présence d'une page web spécifique et d'une chaîne de texte donnée sur cette page web. Si la page web ou la chaîne de texte est introuvable, la fonction Lambda génère une erreur.

Didacticiel : Utilisation d'AWS Lambda avec les événements planifiés

Dans ce didacticiel, vous effectuez les opérations suivantes :

- Vous créez une fonction Lambda via le plan lambda-canary. Vous configurez cette fonction pour qu'elle soit exécutée toutes les minutes. Notez que si la fonction renvoie une erreur, AWS Lambda consigne les métriques correspondantes dans CloudWatch.
- Vous configurez une alarme CloudWatch au niveau de la métrique `Errors` de la fonction Lambda afin de publier un message dans votre rubrique Amazon SNS quand AWS Lambda émet des métriques d'erreur dans CloudWatch. Vous vous abonnez à des rubriques Amazon SNS pour recevoir des notifications par e-mail. Dans ce didacticiel, vous effectuez les opérations suivantes pour obtenir cette configuration :
 - Créez une rubrique Amazon SNS.
 - Abonnez-vous à la rubrique afin de recevoir des notifications par e-mail quand un nouveau message y est publié.
 - Dans Amazon CloudWatch, définissez une alarme au niveau de la métrique `Errors` de la fonction Lambda, afin de publier un message dans votre rubrique SNS lorsque des erreurs se produisent.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

Créez une fonction Lambda

1. Connectez-vous à la AWS Management Console et ouvrez la console AWS Lambda à l'adresse <https://console.aws.amazon.com/lambda/>.
2. Sélectionnez Create function.
3. Choisissez Plans.
4. Entrez **canary** dans la barre de recherche. Choisissez le plan lambda-canary, puis Configurer.
5. 以下を設定します。
 - Nom – **lambda-canary**.
 - Rôle – Créez un nouveau rôle à partir d'un ou de plusieurs modèles.
 - Nom de rôle – **lambda-apigateway-role**.
 - Modèles de stratégie – Autorisations Microservice.
 - Règle – Créer une nouvelle règle.
 - Nom de la règle – **CheckWebsiteScheduledEvent**.
 - Description de la règle – **CheckWebsiteScheduledEvent trigger**.
 - Expression de planification – **rate(1 minute)**.
 - Activé – Vrai (coché).
 - Variables d'environnement
 - site – <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
 - attendue – **What Is AWS Lambda?**
6. Sélectionnez Create function.

CloudWatch Events émet un événement toutes les minutes, en fonction de l'expression de planification. L'événement déclenche la fonction Lambda, qui vérifie que la chaîne attendue apparaît dans la page spécifiée. Pour plus d'informations sur la planification des expressions, consultez la section [Planification des expressions à l'aide de la fréquence ou de cron \(p. 214\)](#).

Test de la fonction Lambda

Testez la fonction avec un exemple d'événement fourni par la console Lambda.

1. Open the Lambda console [Functions page](#).
2. Choisissez lambda-canary.
3. En regard du bouton Test en haut de la page, choisissez Configurer des événements de test dans le menu déroulant.
4. Créez un nouvel événement à l'aide du modèle d'événement CloudWatch Events.
5. Sélectionnez Create.
6. Sélectionnez Test.

Le résultat de l'exécution de la fonction s'affiche en haut de la page.

Créer une rubrique Amazon SNS et s'y abonner

Créez une rubrique Amazon Simple Notification Service pour recevoir des notifications lorsque la fonction canary renvoie une erreur.

Pour créer une rubrique

1. Ouvrez la [console Amazon SNS](#).

2. Choisissez Create topic.
3. Créez une rubrique avec les paramètres suivants.
 - Nom – **lambda-canary-notifications**.
 - Nom complet – **Canary**.
4. Choisissez Create subscription.
5. Créez un abonnement avec les paramètres suivants.
 - Protocole – **Email**.
 - Point de terminaison – votre adresse e-mail.

Amazon SNS envoie un e-mail à partir de Canary <no-reply@sns.amazonaws.com>, reflétant le nom convivial de la rubrique. Utilisez le lien figurant dans l'e-mail pour confirmer votre adresse.

Configurer une alarme

Configurez une alarme dans Amazon CloudWatch pour surveiller la fonction Lambda et envoyer une notification si elle échoue.

Pour créer une alarme

1. Ouvrez la [console CloudWatch](#).
2. Choisissez Alarms.
3. Choisissez Créer une alarme.
4. Choisissez Alarms.
5. Créez une alarme avec les paramètres suivants.

- Métriques – Erreurs lambda-canary.

Recherchez **lambda canary errors** pour trouver la métrique.

- Statistique – **Sum**.

Choisissez la statistique dans le menu déroulant au-dessus de l'aperçu graphique.

- Nom – **lambda-canary-alarm**.
- Description – **Lambda canary alarm**.
- Seuil – Lorsque Errors est **>=1**.
- Envoyer la notification à – **lambda-canary-notifications**

Tester l'alarme

Mettez à jour la configuration de la fonction pour que la fonction renvoie une erreur et déclenche l'alarme.

Pour déclencher une alarme

1. Open the Lambda console [Functions page](#).
2. Choisissez lambda-canary.
3. Sous Variables d'environnement, définissez attendue sur **404**.
4. Choisissez Enregistrer

Attendez une minute, puis recherchez un message de Amazon SNS dans votre messagerie.

Modèle AWS SAM pour une application CloudWatch Events

Vous pouvez générer cette application à l'aide d'[AWS SAM](#). Pour en savoir plus sur la création de modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Vous trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda à partir du [didacticiel \(p. 210\)](#). Copiez le texte ci-dessous dans un fichier .yaml et enregistrez-le en regard du package ZIP que vous avez créé au préalable. Notez que les valeurs de paramètre Handler et Runtime doivent correspondre à celles utilisées lors de la création de la fonction dans la section précédente.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Parameters:
  NotificationEmail:
    Type: String
Resources:
  CheckWebsitePeriodically:
    Type: AWS::Serverless::Function
    Properties:
      Handler: LambdaFunctionOverHttps.handler
      Runtime: runtime
      Policies: AmazonDynamoDBFullAccess
    Events:
      CheckWebsiteScheduledEvent:
        Type: Schedule
        Properties:
          Schedule: rate(1 minute)

  AlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Protocol: email
          Endpoint: !Ref NotificationEmail

  Alarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmActions:
        - !Ref AlarmTopic
      ComparisonOperator: GreaterThanOrEqualToThreshold
      Dimensions:
        - Name: FunctionName
          Value: !Ref CheckWebsitePeriodically
      EvaluationPeriods: 1
      MetricName: Errors
      Namespace: AWS/Lambda
      Period: 60
      Statistic: Sum
      Threshold: '1'
```

Pour plus d'informations sur la manière d'empaqueter et de déployer votre application sans serveur à l'aide des commandes de mise en package et de déploiement, consultez [Déploiement d'applications sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Planification des expressions à l'aide de la fréquence ou de cron

Expression de fréquence

```
rate(Value Unit)
```

Où :

Le champ *Value* peut être un nombre entier positif.

Le champ *Unit* peut correspondre à des minutes, des heures ou des jours.

Par exemple :

Exemple	Expression cron
Appeler la fonction Lambda toutes les 5 minutes	rate(5 minutes)
Appeler la fonction Lambda toutes les heures	rate(1 hour)
Appeler la fonction Lambda tous les sept jours	rate(7 days)

Remarques :

- Les fréquences de moins d'une minute ne sont pas possibles.
- Pour une valeur au singulier, l'unité doit être au singulier (par exemple, `rate(1 day)`). Dans le cas contraire, elle doit être au pluriel (par exemple, `rate(5 days)`).

Expression cron

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Tous les champs sont obligatoires et le fuseau horaire est UTC uniquement. Le tableau suivant décrit ces champs.

Champ	Valeurs	Caractères génériques
minutes	0-59	, - * /
heures	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
mois	1-12 ou JAN-DEC	, - * /
Day-of-week	1-7 ou SUN-SAT	, - * ? / L #
Année	1970-2199	, - * /

Le tableau suivant décrit les caractères génériques.

Caractère	Définition	exemple
/	Spécifie les incrément	0/15 dans le champ des minutes oblige l'exécution à avoir lieu toutes les 15 minutes.
L	Indique qu'il s'agit du dernier élément	Utilisé dans le champ Day-of-month, il s'agit du dernier jour du mois. Dans le champ Day-of-week, il s'agit du dernier jour de la semaine (samedi).
W	Spécifie un jour de la semaine	Lorsqu'il est utilisé avec une date, tel que 5/w, il spécifie le jour de la semaine le plus proche du 5 du mois. Si le 5 tombe un samedi, l'exécution a lieu le vendredi. Si le 5 tombe un dimanche, l'exécution a lieu le lundi.
#	Spécifie le jour du mois	La valeur 3#2 indique le deuxième mardi du mois (mardi est considéré comme étant le troisième jour de la semaine de 7 jours).
*	Spécifie toutes les valeurs	S'il est utilisé dans le champ Day-of-month, il implique tous les jours du mois.
?	Aucune valeur spécifiée	Utilisé conjointement avec une autre valeur. Par exemple, si une date spécifique est définie, mais que le jour de la semaine correspondant n'a pas d'importance.
-	Spécifie des plages de valeurs	10-12comprend 10, 11 et 12
,	Spécifie des valeurs supplémentaires	SUN,MON,TUE signifie dimanche, lundi et mardi
/	Spécifie les incrément	5/10 signifie 5, 15, 25, 35, etc.

Le tableau suivant répertorie des exemples courants d'expressions cron.

Exemple	Expression cron
Appel d'une fonction Lambda à 10 h (UTC) tous les jours	<code>cron(0 10 * * ? *)</code>
Appel de la fonction Lambda à 12 h15 (UTC) tous les jours	<code>cron(15 12 * * ? *)</code>
Appel d'une fonction Lambda à 18 h (UTC) toutes les semaines du lundi au vendredi	<code>cron(0 18 ? * MON-FRI *)</code>
Appel d'une fonction Lambda à 8 h (UTC) chaque premier jour du mois	<code>cron(0 8 1 * ? *)</code>
Appel d'une fonction Lambda toutes les 10 minutes du lundi au vendredi	<code>cron(0/10 * ? * MON-FRI *)</code>

Exemple	Expression cron
Appel d'une fonction Lambda toutes les 5 minutes du lundi au vendredi entre 8 h et 17 h 55 (UTC)	cron(0/5 8-17 ? * MON-FRI *)
Appeler une fonction Lambda à 9 h (UTC) le premier du lundi de chaque mois	cron(0 9 ? * 2#1 *)

Notez bien ce qui suit :

- Les exemples précédents partent du principe que vous utilisez la AWS CLI. Si vous utilisez la console Lambda, n'incluez pas le préfixe `cron` à votre expression.
- Les expressions cron qui entraînent des fréquences d'une rapidité supérieure à une minute ne sont pas prises en charge.
- L'une des valeurs pour le jour du mois ou le jour de la semaine doit être un point d'interrogation (?).

Utilisation de AWS Lambda avec une Equilibreur de charge d'application

Vous pouvez utiliser une fonction Lambda pour traiter les demandes à partir d'un Equilibreur de charge d'application. Elastic Load Balancing prend en charge les fonctions Lambda comme cibles pour une Equilibreur de charge d'application. Utilisez les règles de l'équilibrage de charge pour acheminer les demandes HTTP vers une fonction, selon le chemin d'accès ou les valeurs des en-têtes. Traitez la demande et renvoyez une réponse HTTP à partir de votre fonction Lambda.

Elastic Load Balancing appelle votre fonction Lambda de façon synchrone avec un événement qui contient le corps de la demande et les métadonnées.

Example Événement de demande Equilibreur de charge d'application

```
{
  'requestContext': {
    'elb': {
      'targetGroupArn': 'arn:aws:elasticloadbalancing:us-east-1:011685312445:targetgroup/lambdac-target/d6190d154bc908a5'
    }
  },
  'httpMethod': 'GET',
  'path': '/health',
  'queryStringParameters': {},
  'headers': {
    'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'accept-encoding': 'gzip',
    'accept-language': 'en-US,en;q=0.5',
    'connection': 'keep-alive',
    'cookie': 'cookie',
    'host': 'lambda-846800462.elb.amazonaws.com',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:60.0) Gecko/20100101 Firefox/60.0',
    'x-amzn-trace-id': 'Root=1-5bdb40ca-556d8b0c50dc66f0511bf520',
    'x-forwarded-for': '72.21.198.66',
    'x-forwarded-port': '80',
  }
}
```

```
        'x-forwarded Proto': 'http',
},
'body': '',
'isBase64Encoded': False
}
```

Votre fonction traite l'événement et renvoie une réponse à l'équilibrEUR de charge en JSON. Elastic Load Balancing convertit la réponse en HTTP et la renvoie à l'utilisateur.

Example Format de la réponse

```
{
    "statusCode": 200,
    "statusDescription": "HTTP OK",
    "isBase64Encoded": False,
    "headers": {
        "server": "my-server",
        "set-cookie": "name=value",
        "Content-Type": "text/html; charset=utf-8"
    },
    "body": "Welcome"
}
```

Pour configurer une Equilibreur de charge d'application comme déclencheur de fonction, accordez à Elastic Load Balancing l'autorisation d'exécuter la fonction, créez un groupe cible qui achemine les demandes vers la fonction, et ajoutez une règle à l'équilibrEUR de charge qui envoie les demandes au groupe cible.

Utilisez la commande `add-permission` pour ajouter une instruction d'autorisation à la stratégie basée sur les ressources de votre fonction.

```
$ aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
{
    "Statement": "{\"Sid\":\"load-balancer\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"elasticloadbalancing.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:alb-function\"}"
}
```

Pour obtenir des instructions sur la configuration de l'écouteur et du groupe cible de Equilibreur de charge d'application, consultez [Fonctions Lambda comme cibles](#) dans le Guide de l'utilisateur pour les Application Load Balancers.

Utilisation d'AWS Lambda avec l'AWS Command Line Interface

Vous pouvez utiliser l'AWS Command Line Interface pour gérer les fonctions et les autres ressources d'AWS Lambda. L'AWS CLI utilise AWS SDK for Python (Boto) pour interagir avec l'API Lambda. Vous pouvez l'utiliser pour en savoir plus sur l'API, et appliquer ces connaissances pour générer des applications qui utilisent Lambda avec le kit SDK AWS.

Dans ce didacticiel, vous gérez et appelez des fonctions Lambda avec l'AWS CLI.

Prérequis

This tutorial assumes that you have some knowledge of basic Lambda operations and the Lambda console. If you haven't already, follow the instructions in [Mise en route \(p. 3\)](#) to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/lambda-project$ this is a command  
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution](#) (p. 261) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS Lambda.
 - Autorisations – AWSLambdaBasicExecutionRole.
 - Nom de rôle – **lambda-cli-role**.

La stratégie AWSLambdaBasicExecutionRole possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple de code suivant reçoit un événement en entrée et consigne certaines données de l'événement entrant dans CloudWatch Logs.

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, "Success");
};
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement.

```
$ zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
$ aws lambda create-function --function-name helloworld \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role role-arn
{
    "FunctionName": "helloworld",
    "CodeSize": 351,
    "MemorySize": 128,
    "FunctionArn": "function-arn",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
    "Timeout": 3,
    "LastModified": "2015-04-07T22:02:58.854+0000",
    "Runtime": "nodejs8.10",
    "Description": ""
}
```

Appelez votre fonction Lambda à l'aide de la commande `invoke`.

```
$ aws lambda invoke --function-name helloworld --log-type Tail \
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}' \
outputfile.txt
{
    "LogResult": "base64-encoded-log",
    "StatusCode": 200
}
```

En spécifiant le paramètre `--log-type`, la commande demande également la fin du journal généré par la fonction. Les données de journal figurant dans la réponse sont codées en base64. Utilisez le programme `base64` pour décoder le journal.

```
$ echo base64-encoded-log | base64 --decode
START RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value1 = value1
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value2 = value2
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value3 = value3
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 result: "value1"
END RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
REPORT RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
Duration: 13.35 ms      Billed Duration: 100 ms   Memory Size: 128 MB
Max Memory Used: 9 MB
```

Comme vous avez appelé la fonction à l'aide du type d'appel par défaut (`RequestResponse`), la connexion reste ouverte jusqu'à ce que l'exécution se termine. Lambda écrit la réponse dans le fichier de sortie.

Répertorier les fonctions Lambda dans votre compte

Exécutez la commande d'AWS CLI `list-functions` suivante pour récupérer la liste des fonctions que vous avez créées.

```
$ aws lambda list-functions --max-items 10
{
    "Functions": [
        {
            "FunctionName": "helloworld",
            "MemorySize": 128,
```

```

        "CodeSize": 412,
        "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
        "Handler": "ProcessKinesisRecords.handler",
        "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
        "Timeout": 3,
        "LastModified": "2015-02-22T21:03:01.172+0000",
        "Runtime": "nodejs6.10",
        "Description": ""
    },
    {
        "FunctionName": "ProcessKinesisRecords",
        "MemorySize": 128,
        "CodeSize": 412,
        "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
        "Handler": "ProcessKinesisRecords.handler",
        "Role": "arn:aws:iam::account-id:role/lambda-execute-test-kinesis",
        "Timeout": 3,
        "LastModified": "2015-02-22T21:03:01.172+0000",
        "Runtime": "nodejs6.10",
        "Description": ""
    },
    ...
],
    "NextMarker": null
}

```

En réponse, Lambda renvoie une liste de 10 fonctions maximum. S'il existe plus de fonctions à récupérer, `NextMarker` fournit un marqueur que vous pourrez utiliser dans la requête `list-functions` suivante. Dans le cas contraire, la valeur est « `null` ». La commande d'AWS CLI `list-functions` suivante est un exemple qui illustre le paramètre `--marker`.

```
$ aws lambda list-functions --max-items 10 \
--marker value-of-NextMarker-from-previous-response
```

Récupérer une fonction Lambda

La commande d'interface de ligne de commande Lambda `get-function` renvoie les métadonnées de la fonction Lambda et une URL présignée que vous pouvez utiliser pour télécharger le package de déploiement de cette fonction.

```

$ aws lambda get-function --function-name helloworld
{
    "Code": {
        "RepositoryType": "S3",
        "Location": "pre-signed-url"
    },
    "Configuration": {
        "FunctionName": "helloworld",
        "MemorySize": 128,
        "CodeSize": 287,
        "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
        "Timeout": 3,
        "LastModified": "2015-04-07T22:02:58.854+0000",
        "Runtime": "nodejs8.10",
        "Description": ""
    }
}

```

Pour plus d'informations, consultez [GetFunction \(p. 437\)](#).

Nettoyage

Exécutez la commande `delete-function` suivante pour supprimer la fonction `helloworld`.

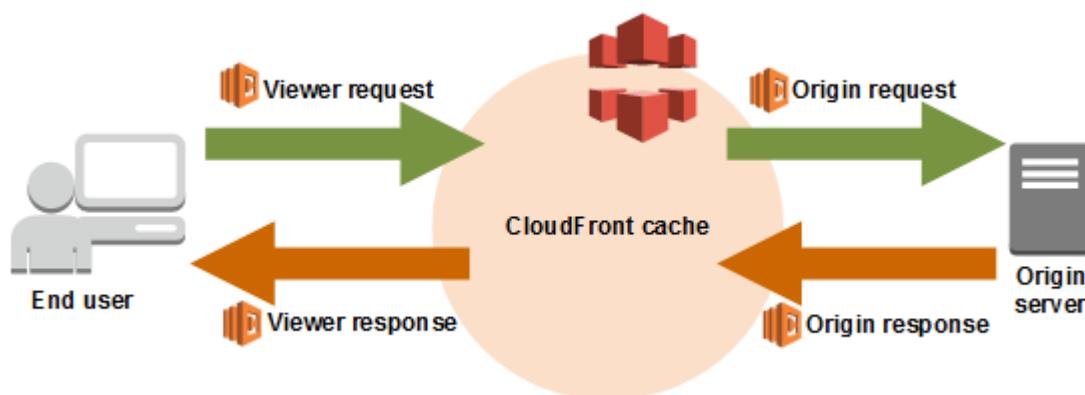
```
$ aws lambda delete-function --function-name helloworld
```

Supprimez le rôle IAM que vous avez créé dans la console IAM. Pour plus d'informations sur la suppression d'un rôle, consultez [Suppression de rôles ou de profils d'instance](#) dans le IAM Guide de l'utilisateur.

Lambda@Edge

Lambda@Edge vous permet d'exécuter des fonctions Lambda pour personnaliser le contenu transmis par CloudFront, en exécutant les fonctions dans des emplacements AWS plus proches de l'utilisateur. Les fonctions s'exécutent en réponse à des événements CloudFront sans allouer ou gérer des serveurs. Vous pouvez utiliser les fonctions Lambda pour modifier les requêtes et réponses CloudFront aux stades suivants :

- Après la réception par CloudFront d'une demande provenant de l'utilisateur (demande de l'utilisateur)
- Avant la transmission par CloudFront d'une demande à l'origine (demande à l'origine)
- Après la réception par CloudFront de la réponse provenant de l'origine (réponse de l'origine)
- Avant la transmission par CloudFront de la réponse pour l'utilisateur (réponse à l'utilisateur)



Vous pouvez également générer des réponses pour les utilisateurs sans jamais envoyer de demande à l'origine.

Lambda@Edge vous permet de créer diverses solutions, par exemple :

- Inspecter des cookies afin de réécrire des URL vers différentes versions d'un site en vue de réaliser des tests A/B.
- Envoyer des objets différents à vos utilisateurs en fonction de l'en-tête `User-Agent`, qui contient des informations sur l'appareil qui a envoyé la demande. Par exemple, vous pouvez envoyer des images aux utilisateurs dans différentes résolutions selon l'appareil qu'ils utilisent.
- Inspecter les en-têtes ou les jetons autorisés, en insérant un en-tête correspondant et en autorisant le contrôle de l'accès avant de transférer une demande vers l'origine.

- Ajouter, supprimer et modifier des en-têtes, et réécrire le chemin d'accès de l'URL pour diriger les utilisateurs vers différents objets dans le cache.
- Générer de nouvelles réponses HTTP pour rediriger les utilisateurs non authentifiés vers les pages de connexion, ou créer et déployer des pages web statiques depuis le périphérique. Pour plus d'informations, consultez [Utilisation de fonctions Lambda pour générer des réponses HTTP aux demandes de l'utilisateur et de l'origine](#) dans le Amazon CloudFront Manuel du développeur.

Pour plus d'informations sur l'utilisation de Lambda@Edge, consultez [Utilisation de CloudFront avec Lambda@Edge](#).

Surveillance et dépannage des applications Lambda

AWS Lambda suit automatiquement le comportement des appels de votre fonction Lambda et fournit des informations en retour que vous pouvez surveiller. En outre, les métriques qu'il fournit permettent d'analyser les appels de fonction dans leur globalité, y compris l'intégration des sources d'événements et les performances des ressources en aval. Vous trouverez dans les sections suivantes des conseils sur les outils pouvant servir à analyser le comportement de vos appels de la fonction Lambda :

Rubriques

- [Utiliser Amazon CloudWatch \(p. 223\)](#)
- [Utilisation d'AWS X-Ray \(p. 231\)](#)

Utiliser Amazon CloudWatch

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et présente les métriques via Amazon CloudWatch. Pour vous aider à surveiller le code à mesure qu'il s'exécute, Lambda effectue un suivi automatique du nombre de demandes, de la durée d'exécution par demande et du nombre de demandes générant une erreur, puis publie les métriques CloudWatch associées. Vous pouvez tirer parti de ces métriques pour configurer des alarmes CloudWatch personnalisées. Pour plus d'informations sur CloudWatch, consultez le [Guide de l'utilisateur Amazon CloudWatch](#).

Vous pouvez afficher les débits de requêtes et des taux d'erreur pour chaque fonction Lambda via la console AWS Lambda, la console CloudWatch et d'autres ressources Amazon Web Services (AWS). Les rubriques suivantes décrivent les métriques Lambda CloudWatch et expliquent comment y accéder.

- [Accès aux métriques Amazon CloudWatch pour AWS Lambda \(p. 225\)](#)
- [Indicateurs AWS Lambda \(p. 228\)](#)

Vous pouvez insérer des instructions de journalisation dans le code pour vous aider vérifier qu'il fonctionne comme prévu. Lambda s'intègre automatiquement à Amazon CloudWatch Logs et transmet tous les journaux provenant de votre code à un groupe CloudWatch Logs associé à une fonction Lambda (`/aws/lambda/<nom de la fonction>`). Pour en savoir plus sur les groupes de journaux et découvrir comment y accéder via la console CloudWatch, consultez [Surveillance des fichiers journaux système, d'application et personnalisés](#) dans le Guide de l'utilisateur Amazon CloudWatch. Pour découvrir comment accéder aux entrées de journal CloudWatch, consultez [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 227\)](#).

Note

Si le code de la fonction Lambda s'exécute, mais que vous ne voyez pas de données de journal générées après quelques minutes, il se peut que votre rôle d'exécution pour cette fonction n'ait pas accordé les autorisations requises pour écrire les données de journal dans CloudWatch Logs. Pour découvrir comment vérifier que vous avez configuré le rôle d'exécution correctement pour accorder ces autorisations, consultez la page [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).

Scénarios de résolution des problèmes AWS Lambda

Cette section utilise des exemples pour décrire comment surveiller les fonctions Lambda et résoudre les problèmes afférents via les fonctionnalités de journalisation et de surveillance de CloudWatch.

Scénario 1 de résolution des problèmes : la fonction Lambda ne se comporte pas comme prévu

Dans ce scénario, vous venez de terminer le [Didacticiel : Utilisation d'AWS Lambda avec Amazon S3 \(p. 128\)](#). Toutefois, la fonction Lambda que vous avez créée pour importer une image miniature dans Amazon S3 lors de la création d'un objet S3 ne se comporte pas comme prévu. Lorsque vous importez des objets dans Amazon S3, aucune miniature n'est importée. Les options suivantes s'offrent à vous pour résoudre ce problème.

Pour déterminer pourquoi la fonction Lambda ne se comporte pas comme prévu

1. Vérifiez le code afin de vous assurer qu'il fonctionne correctement. Un taux d'erreur élevé indique la présence d'un problème.

Vous pouvez tester le code localement (comme vous le feriez avec n'importe quelle autre fonction Node.js), via la fonctionnalité de test de la console Lambda ou via la commande `Invoke` de l'interface AWS CLI. Chaque fois que le code est exécuté en réponse à un événement, il écrit une entrée dans le groupe de journaux associé à une fonction Lambda (`/aws/lambda/<nom de la fonction>`).

Voici quelques exemples d'erreurs qui peuvent apparaître dans les journaux :

- Si vous voyez une trace de pile dans le journal, le code comporte probablement une erreur. Vérifiez-le et déboguez l'erreur indiquée par la trace de la pile.
- Si vous voyez une erreur `permissions denied` dans le journal, le rôle IAM que vous avez fourni comme rôle d'exécution n'a peut-être pas les autorisations nécessaires. Vérifiez que le rôle IAM possède toutes les autorisations nécessaires pour accéder aux ressources AWS référencées par votre code. Pour vous assurer que vous avez correctement configuré le rôle d'exécution, consultez la section [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#).
- Si vous voyez une erreur `timeout exceeded` dans le journal, votre fonction a été interrompue, car elle n'a pas renvoyé de résultat avant l'expiration du délai d'attente configuré. Il est possible que le délai d'expiration soit trop court ou que l'exécution du code soit trop longue.
- Si vous voyez une erreur `memory exceeded` dans le journal, les paramètres de la mémoire sont trop bas. Définissez une valeur plus élevée. Pour plus d'informations sur les limites liées à la taille de mémoire, consultez la section [CreateFunction \(p. 410\)](#).

2. Vérifiez que la fonction Lambda reçoit les requêtes.

Même si le code de la fonction a l'effet prévu et s'il répond correctement aux appels tests, la fonction ne reçoit peut-être pas les requêtes à partir d'Amazon S3. Si Amazon S3 parvient à appeler la fonction, vous devriez constater une augmentation des métriques liées aux requêtes CloudWatch. Dans le cas contraire, vérifiez la stratégie d'autorisations d'accès associée à la fonction.

Scénario 2 de résolution des problèmes : augmentation de la durée d'exécution de la fonction Lambda

Dans ce scénario, vous venez de terminer le [Didacticiel : Utilisation d'AWS Lambda avec Amazon S3 \(p. 128\)](#). Toutefois, la fonction Lambda que vous avez créée pour importer une image miniature dans Amazon S3 lors de la création d'un objet S3 ne se comporte pas comme prévu. Lorsque vous importez des objets dans Amazon S3, vous voyez que l'importation des miniatures est lancée, mais l'exécution du code prend plus de temps que prévu. Vous pouvez résoudre ce problème de deux façons différentes. Par exemple, vous pouvez surveiller la métrique de durée de la fonction Lambda pour voir si le temps d'exécution augmente. Vous pouvez également constater une augmentation de la métrique CloudWatch d'erreurs pour la fonction Lambda, qui peut être liée à un problème au niveau du délai d'expiration.

Pour déterminer pourquoi la durée de l'exécution d'une fonction Lambda a augmenté

1. Testez le code avec différents paramètres de mémoire.

Si l'exécution du code est trop longue, les ressources de calcul nécessaires ne sont peut-être pas suffisantes pour exécuter sa logique. Essayez d'augmenter la mémoire allouée à la fonction, puis testez le code à nouveau via la fonctionnalité de test de la console Lambda. Vous pouvez déterminer la mémoire utilisée, le délai d'exécution du code et la mémoire allouée dans les entrées de journal de la fonction. Si vous modifiez le paramètre de mémoire, cela peut avoir une incidence sur la façon dont vous êtes facturé pour la durée d'exécution. Pour en savoir plus sur la tarification, consultez [AWS Lambda](#).

2. Utilisez des journaux afin de déterminer l'origine du goulot d'étranglement.

Vous pouvez tester le code localement (comme vous le feriez avec n'importe quelle autre fonction Node.js), via la fonctionnalité de test de la console Lambda ou via la commande `aws lambda invoke` de l'interface AWS CLI. Chaque fois que le code est exécuté en réponse à un événement, il écrit une entrée dans le groupe de journaux associé à une fonction Lambda (`/aws/lambda/<nom de la fonction>`). Ajoutez des instructions de journalisation au niveau de différentes parties de votre code, telles que des appels à d'autres services, pour voir combien de temps il lui faut pour les exécuter.

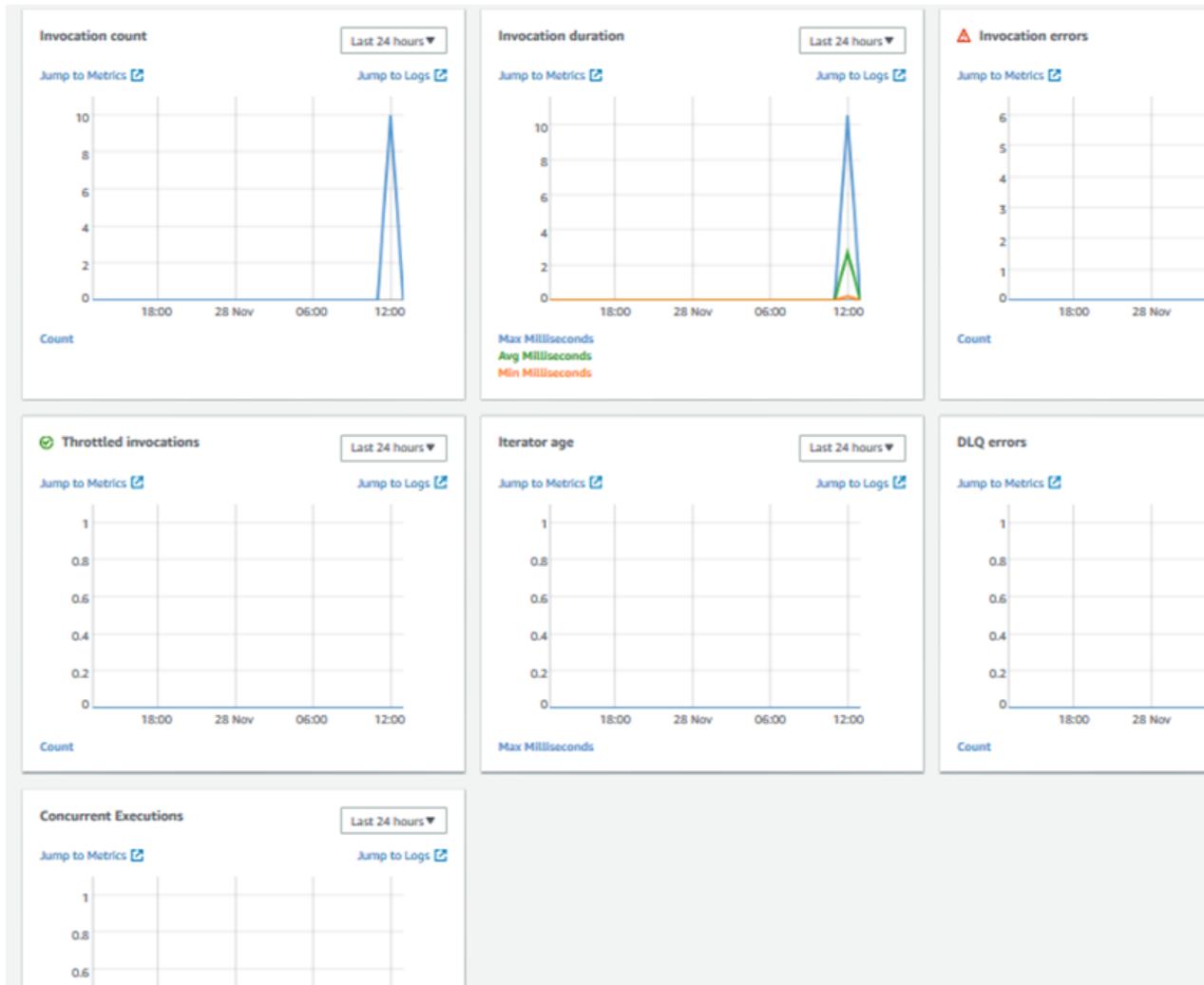
Accès aux métriques Amazon CloudWatch pour AWS Lambda

AWS Lambda surveille automatiquement les fonctions en votre nom et présente les métriques via Amazon CloudWatch. Ces métriques incluent le nombre total de demandes, la durée et les taux d'erreur. Pour en savoir plus sur les métriques Lambda, consultez [Indicateurs AWS Lambda \(p. 228\)](#). Pour plus d'informations sur CloudWatch, consultez [Guide de l'utilisateur Amazon CloudWatch](#).

Pour surveiller les métriques Lambda et pour afficher les journaux, utilisez la console Lambda, la console CloudWatch, l'AWS CLI ou l'API CloudWatch. Les procédures suivantes vous montrent comment accéder aux métriques à l'aide de ces différentes méthodes.

Pour accéder aux métriques grâce à la console Lambda

1. Connectez-vous à la AWS Management Console et ouvrez la console AWS Lambda à l'adresse <https://console.aws.amazon.com/lambda/>.
2. Sur la page Functions , sélectionnez le nom de la fonction, puis cliquez sur l'onglet Monitoring.



Une représentation graphique des métriques de la fonction Lambda s'affiche.

- Choisissez Jump to logs (Passer aux journaux) pour afficher les journaux.

Pour accéder aux métriques grâce à la console CloudWatch

- Ouvrez la console CloudWatch à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
- Dans la barre de navigation, sélectionnez une région.
- Dans le volet de navigation, choisissez Métriques.
- Dans le volet Métriques CloudWatch par catégorie, sélectionnez Métriques Lambda.
- (Facultatif) Dans le volet de graphique, sélectionnez une statistique et une période de temps, puis créez une alarme CloudWatch avec ces paramètres.

Pour accéder aux métriques grâce à AWS CLI

Utilisez les commandes `list-metrics` et `get-metric-statistics`.

Pour accéder aux métriques via l'interface de ligne de commande CloudWatch

Utilisez les commandes `mon-list-metrics` et `mon-get-stats`.

Pour accéder aux mesures grâce à l'API d'CloudWatch

Utilisez les opérations [ListMetrics](#) et [GetMetricStatistics](#).

Accès aux journaux Amazon CloudWatch pour AWS Lambda

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et présente les métriques via Amazon CloudWatch. Pour vous aider à résoudre les problèmes d'une fonction, Lambda consigne toutes les requêtes gérées par cette dernière et stocke automatiquement les journaux que votre code génère via Amazon CloudWatch Logs.

Vous pouvez insérer des instructions de journalisation dans le code pour vous aider vérifier qu'il fonctionne comme prévu. Lambda s'intègre automatiquement à CloudWatch Logs et transmet tous les journaux provenant de votre code à un groupe CloudWatch Logs associé à une fonction Lambda, nommée `/aws/lambda/<nom de la fonction>`. Pour en savoir plus sur les groupes de journaux et découvrir comment y accéder via la console CloudWatch, consultez [Surveillance des fichiers journaux système, d'application et personnalisés](#) dans le Guide de l'utilisateur Amazon CloudWatch.

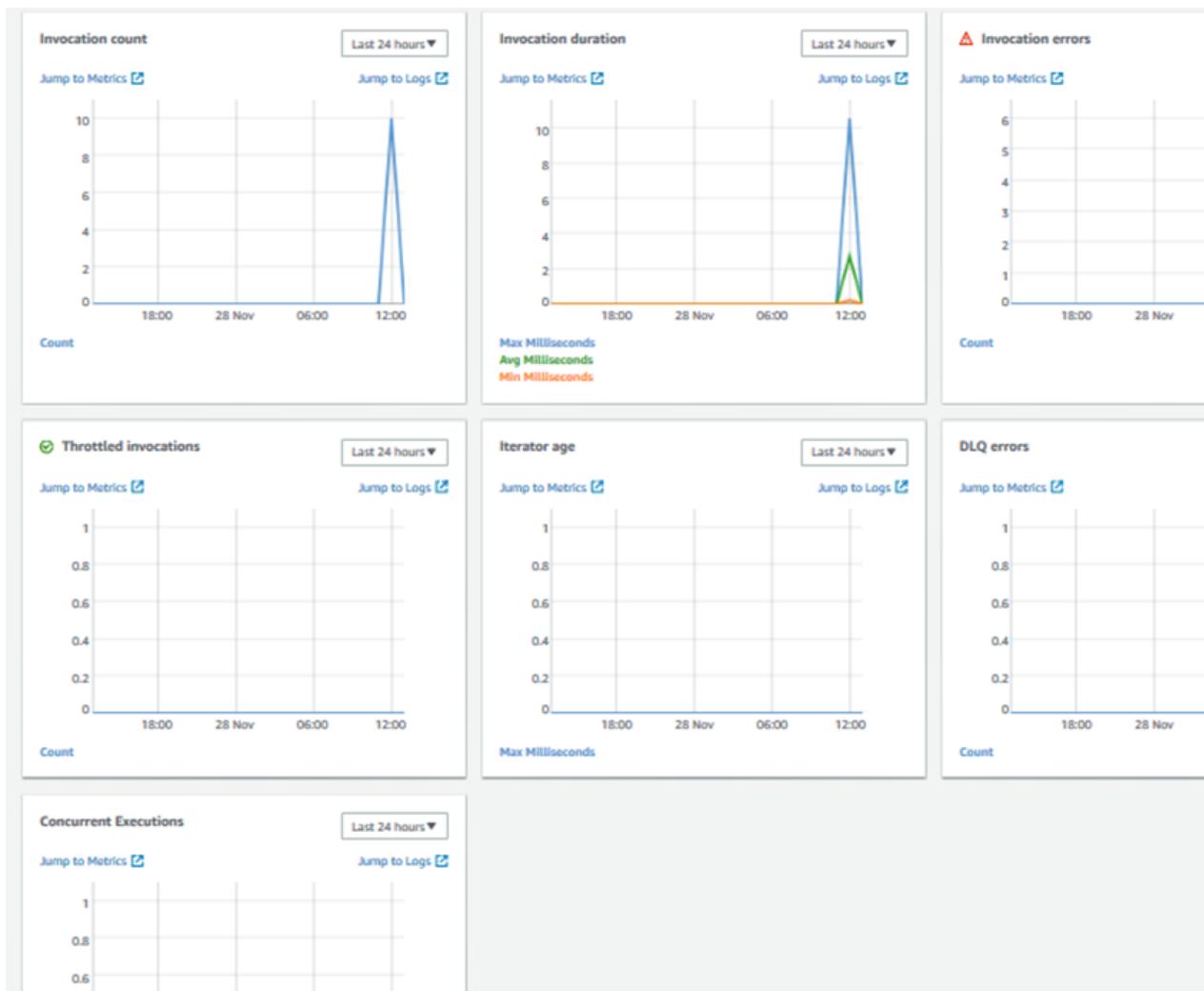
Pour afficher les journaux correspondant à Lambda, utilisez la console Lambda, la console CloudWatch, l'AWS CLI ou l'API CloudWatch. La procédure suivante vous montre comment afficher les journaux via la console Lambda.

Note

L'utilisation des journaux Lambda n'implique aucun coût supplémentaire. Toutefois, les frais CloudWatch Logs standard s'appliquent. Pour plus d'informations, consultez [Tarification CloudWatch](#).

Pour afficher les journaux via la console Lambda

1. Connectez-vous à la AWS Management Console et ouvrez la console AWS Lambda à l'adresse <https://console.aws.amazon.com/lambda/>.
2. Sur la page Functions , sélectionnez le nom de la fonction, puis cliquez sur l'onglet Monitoring.



Une représentation graphique des métriques de la fonction Lambda s'affiche.

3. Choisissez Jump to logs (Passer aux journaux) pour afficher les journaux.

Pour plus d'informations sur l'accès à CloudWatch Logs, consultez les guides suivants :

- [Guide de l'utilisateur Amazon CloudWatch](#)
- [Amazon CloudWatch Logs API Reference](#)
- [Surveillance des fichiers journaux](#) dans le Guide de l'utilisateur Amazon CloudWatch

Indicateurs AWS Lambda

Cette rubrique décrit l'espace de noms, les métriques et les dimensions AWS Lambda. AWS Lambda surveille automatiquement les fonctions en votre nom et présente les métriques via Amazon CloudWatch. Ces métriques comprennent tous les appels, les erreurs, la durée, les limitations, les erreurs DLQ et l'âge de l'itérateur relatifs aux appels basés sur les flux.

CloudWatch est principalement un référentiel de métriques. Une métrique est le concept fondamental de CloudWatch et représente un ensemble de points de données ordonnés dans le temps. Vous (ou

les services AWS) publiez les points de données des métriques dans CloudWatch et vous récupérez les statistiques concernant ces points de données sous la forme d'un ensemble ordonné de données chronologiques.

Les métriques sont uniquement définies par un nom, un espace de noms et une ou plusieurs dimensions. Chaque point de données comporte un horodatage et, le cas échéant, une unité de mesure. Lorsque vous demandez des statistiques, le flux de données renvoyé est identifié par l'espace de noms, le nom de la métrique et la dimension. Pour plus d'informations sur CloudWatch, consultez le [Guide de l'utilisateur Amazon CloudWatch](#).

Métriques AWS Lambda CloudWatch

L'espace de noms AWS/Lambda inclut les métriques suivantes.

Métrique	Description
Invocations	<p>Mesure le nombre de fois qu'une fonction est appelée en réponse à un événement ou un appel d'API. Cela remplace la métrique RequestCount obsolète. Cela inclut les appels réussis ou en échec, mais n'inclut pas les tentatives de limitation. Cela équivaut aux demandes facturées pour la fonction. Notez que AWS Lambda envoie uniquement ces mesures à CloudWatch si elles ont une valeur différente de zéro.</p> <p>Unités : nombre</p>
Errors	<p>Mesure le nombre d'appels ayant échoué en raison d'erreurs dans la fonction (code de réponse 4XX). Cela remplace la métrique ErrorCount obsolète. Les appels ayant échoué peuvent déclencher une tentative qui réussit. Les points suivants sont abordés :</p> <ul style="list-style-type: none"> Exceptions gérées (par exemple, context.fail(error)) Exceptions non gérées, ce qui entraîne l'arrêt du code Exceptions de mémoire insuffisante Délais Erreurs d'autorisations <p>Cela n'inclut pas les appels qui échouent en raison des taux d'appel entraînant un dépassement de limites simultanées par défaut (code d'erreur 429) ou les défaillances dues à des erreurs de service interne (code d'erreur 500).</p> <p>Unités : nombre</p>
DeadLetterErrors	<p>Métrique incrémentée quand Lambda ne parvient pas à écrire la charge utile de l'événement en échec sur vos files d'attente Lettre Morte. Cela peut être dû à l'un des éléments suivants :</p> <ul style="list-style-type: none"> Erreurs d'autorisations Limitations des services en aval Configuration incorrecte des ressources Délais <p>Unités : nombre</p>
Duration	<p>Mesure le temps écoulé entre le moment où le code de la fonction commence à s'exécuter en raison d'un appel jusqu'à l'arrêt de son exécution. La valeur de</p>

Métrique	Description
	<p>point de données maximale possible est la configuration du délai d'expiration de la fonction. La durée facturée sera arrondie aux 100 millisecondes près. Notez que AWS Lambda envoie uniquement ces mesures à CloudWatch si elles ont une valeur différente de zéro.</p> <p>Unités : millisecondes</p>
Throttles	<p>Mesure le nombre de tentatives d'appel de fonction Lambda qui ont été limitées en raison des taux d'appel entraînant un dépassement de limites simultanées du client (code d'erreur 429). Les appels ayant échoué peuvent déclencher une tentative qui réussit.</p> <p>Unités : nombre</p>
IteratorAge	<p>Émis pour les appels basés sur les flux uniquement (fonctions déclenchées par un flux Amazon DynamoDB ou Kinesis). Mesure l'âge du dernier enregistrement pour chaque lot d'enregistrements traité. L'âge est la différence entre l'heure à laquelle Lambda a reçu le lot et celle à laquelle le dernier enregistrement du lot a été inscrit dans le flux.</p> <p>Unités : millisecondes</p>
ConcurrentExecutions	<p>Émise comme une métrique agrégée pour toutes les fonctions du compte et pour les fonctions qui ont une limite de devise personnalisée spécifiée. Non applicable pour les versions ou les alias. Mesure la somme des exécutions simultanées pour une fonction donnée à un point donné dans le temps. Doit s'afficher comme moyenne de métrique si elle a été agrégée sur une période de temps.</p> <p>Unités : nombre</p>
UnreservedConcurrentExecutions	<p>Émise comme une métrique agrégée pour toutes les fonctions du compte uniquement. Non applicable pour les fonctions, les versions ou les alias. Représente la somme des fonctions simultanées qui n'ont pas de limite de devise personnalisée spécifiée. Doit s'afficher comme moyenne de métrique si elle a été agrégée sur une période de temps.</p> <p>Unités : nombre</p>

Rapport entre les erreurs et les appels

Lors du calcul du taux d'erreur sur les appels de fonctions Lambda, il est important de faire la distinction entre une demande d'appel et un appel réel. Il est possible pour le taux d'erreur de dépasser le nombre d'invocations de fonction Lambda. Lambda signale une métrique d'invocation uniquement si le code de fonction Lambda est exécuté. Si la demande d'invocation donne une limitation ou une autre erreur d'initialisation qui empêche le code de fonction Lambda d'être invoqué, Lambda signalera une erreur, mais ne consignera pas une métrique d'invocation.

- Lambda émet `Invocations=1` lorsque la fonction est exécutée. Si la fonction Lambda n'est pas exécutée, rien n'est émis.
- Lambda émet un point de données pour `Errors` pour chaque demande d'invocation. `Errors=0` signifie qu'il n'y a aucune erreur d'exécution de fonction. `Errors=1` signifie qu'il y a une erreur d'exécution de fonction.
- Lambda émet un point de données pour `Throttles` pour chaque demande d'invocation. `Throttles=0` signifie qu'il n'y a aucune restriction d'invocation. `Throttles=1` signifie qu'il y a une restriction d'invocation.

Dimensions AWS Lambda CloudWatch

Vous pouvez utiliser les dimensions du tableau suivant pour affiner les métriques retournées pour vos fonctions Lambda.

Dimension	Description
FunctionName	Filtre les données des métriques par fonction Lambda.
Resource	Filtre les données de métrique par une ressource de fonction Lambda, comme une version ou un alias de fonction.
ExecutedVersion	Filtre les données des métriques par versions de fonction Lambda. S'applique uniquement aux appels d'alias.

Utilisation d'AWS X-Ray

Une application basée sur Lambda classique se compose d'une ou de plusieurs fonctions déclenchées par des événements tels que des importations d'objets dans Amazon S3, des notifications Amazon SNS et des actions d'API. Une fois lancées, ces fonctions appellent généralement des ressources en aval telles que des tables DynamoDB ou des compartiments Amazon S3, ou effectuent d'autres appels d'API. AWS Lambda exploite Amazon CloudWatch afin qu'il émette automatiquement les métriques et journaux pour tous les appels de votre fonction. Toutefois, ce mécanisme n'est pas pratique pour le suivi de la source d'événement qui a appelé votre fonction Lambda, ou pour le suivi des appels en aval que votre fonction a effectués. Pour obtenir une présentation complète du fonctionnement du suivi, consultez [AWS X-Ray](#).

Suivi des applications basées sur Lambda avec AWS X-Ray

AWS X-Ray est un service AWS qui vous permet de détecter, d'analyser et d'optimiser les problèmes de performances avec vos applications AWS Lambda. X-Ray collecte les métadonnées du service Lambda et de tout service en amont ou en aval qui compose votre application. X-Ray utilise ces métadonnées pour générer un graphique de service détaillé qui illustre la dégradation des performances, les pics de latence et d'autres problèmes ayant un impact sur les performances de votre application Lambda.

Une fois que vous avez utilisé [Lambda sur la cartographie des services AWS X-Ray \(p. 231\)](#) pour identifier une ressource ou un composant qui pose un problème, vous pouvez faire un zoom avant et visualiser une représentation de la requête. Cette représentation visuelle couvre le temps allant du déclenchement d'une fonction Lambda par une source d'événement jusqu'à l'exécution complète de la fonction. X-Ray fournit une répartition des opérations de votre fonction, par exemple les informations concernant les appels que la fonction Lambda a effectué auprès d'autres services. En outre, l'intégration de X-Ray à Lambda vous fournit la visibilité relativement aux traitements supplémentaires liés au service AWS Lambda. Vous avez ainsi accès à des informations spécifiques telles que le temps d'arrêt et le nombre d'appels de la demande.

Note

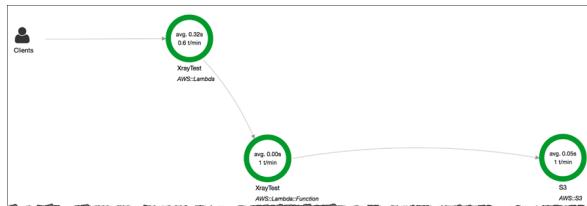
Seuls les services actuellement intégrés à X-Ray sont affichés en tant que traces autonomes, en dehors de votre trace Lambda. Pour obtenir la liste des services qui prennent actuellement en charge X-Ray, consultez [Intégration d'AWS X-Ray avec d'autres services AWS](#).

Lambda sur la cartographie des services AWS X-Ray

X-Ray affiche trois types de nœuds sur la Service Map pour les requêtes traitées par Lambda :

- Service Lambda (AWS::Lambda) : ce type de nœud représente le temps passé par la requête dans le service Lambda. Le décompte commence quand Lambda reçoit la requête et se termine lorsque celle-ci quitte le service Lambda.
- Fonction Lambda (AWS::Lambda::Function) : ce type de nœud représente la durée d'exécution de la fonction Lambda.
- Appels de service en aval : dans ce type, chaque appel de service en aval effectué depuis la fonction Lambda est représenté par un nœud distinct.

Dans le graphique suivant, les nœuds représentent (de gauche à droite) : le service Lambda, la fonction utilisateur et un appel en aval vers Amazon S3 :



Pour plus d'informations, consultez [Affichage de la cartographie des services](#).

Lambda comme suivi AWS X-Ray

Depuis la cartographie des services, vous pouvez faire un zoom avant afin de consulter un suivi de la fonction Lambda. Le suivi affiche des informations détaillées concernant les appels de votre fonction, représentés sous la forme de segments et de sous-segments :

- Segment de service Lambda : ce segment représente différentes informations selon la source de l'événement utilisé pour appeler la fonction :
 - Sources d'événement synchrones et de flux : le segment de service mesure la durée entre le moment où le service Lambda reçoit la requête/l'événement et celui où la requête quitte ce service (après l'exécution de l'appel final de la requête).
 - Asynchrone : le segment de service représente le temps de réponse, c'est-à-dire le temps mis par le service Lambda pour renvoyer une réponse 202 au client.

Le segment de service Lambda peut inclure deux types de sous-segments :

- Temps d'arrêt (appels asynchrones uniquement) : représente le temps passé par la fonction dans le service Lambda avant d'être appelée. Ce sous-segment commence lorsque le service Lambda reçoit la requête/l'événement et se termine lorsque la fonction Lambda est appelée pour la première fois.
- Tentative : représente une seule tentative d'appel, comportant tous les traitements supplémentaires liés au service Lambda. Les exemples d'efforts supplémentaires sont le temps passé à initialiser le code de la fonction et la durée d'exécution de la fonction.
- Segment de fonction Lambda : représente la durée d'exécution de la fonction pour une tentative d'appel spécifique. Il commence lorsque le gestionnaire de fonctions commence l'exécution et se termine lorsque la fonction se termine. Ce segment peut inclure trois types de sous-segments :
 - Initialisation : le temps passé à exécuter le code `initialization` de la fonction, défini comme le code en dehors du gestionnaire de fonctions ou des initialiseurs statiques Lambda.
 - Appels en aval : appels effectués auprès d'autres services AWS par le code de la fonction Lambda.
 - Sous-segments personnalisés : sous-segments personnalisés ou annotations utilisateur que vous pouvez ajouter au segment de fonction Lambda en utilisant le SDK X-Ray.

Note

Pour chaque appel suivi, Lambda émet le segment de service Lambda et tous ses sous-segments. Ces segments sont émis indépendamment de l'exécution et nécessitent que vous utilisez le kit SDK XRay pour les appels d'API AWS.

Configuration de AWS X-Ray avec Lambda

Vous trouverez ci-après des informations détaillées sur la configuration de X-Ray avec Lambda.

Avant de commencer

Pour activer le suivi sur votre fonction Lambda à l'aide de l'interface de ligne de commande Lambda, vous devez commencer par ajouter les autorisations du suivi au rôle d'exécution de la fonction. Pour ce faire, effectuez les étapes suivantes :

- Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
- Trouvez le rôle d'exécution Lambda pour votre fonction.
- Attachez la stratégie gérée suivante : `AWSXrayWriteOnlyAccess`

Pour en savoir plus sur ces stratégies, consultez [AWS X-Ray](#).

Si vous activez le mode de suivi d'actifs à l'aide de la console Lambda, les autorisations de suivi sont ajoutées automatiquement, comme expliqué dans la section suivante.

Suivi

Le chemin d'une demande via votre application est suivie avec un ID de suivi. Une trace collecte tous les segments générés par une seule requête, généralement une demande HTTP GET ou POST.

Il existe deux modes de suivi pour une fonction Lambda :

- Transfert : il s'agit du paramètre par défaut pour toutes les fonctions Lambda si vous avez ajouté les autorisations de suivi à votre rôle d'exécution de la fonction. Cette approche signifie que la fonction Lambda est uniquement suivie si X-Ray a été activé sur un service en amont, par exemple AWS Elastic Beanstalk.
- Active : lorsqu'une fonction Lambda a ce paramètre, Lambda crée automatiquement des exemples de requêtes d'appels, en fonction de l'algorithme d'échantillonnage spécifié par X-Ray.

Note

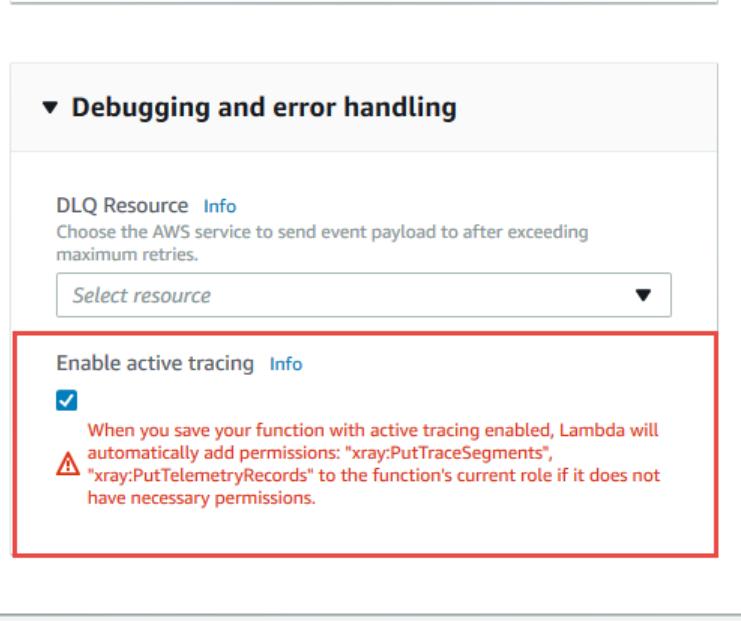
X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif des demandes servies par votre application. L'algorithme d'échantillonnage par défaut est d'1 demande par minute, avec 5 % des demandes échantillonées au-delà de cette limite. Cependant, si le volume de trafic vers votre fonction est faible, vous pouvez voir une augmentation du taux d'échantillonnage.

Vous pouvez modifier le mode de suivi de la fonction Lambda en utilisant la console de gestion Lambda ou les actions d'API &LAM ; [CreateFunction \(p. 410\)](#) ou [UpdateFunctionConfiguration \(p. 514\)](#).

Si vous utilisez la console Lambda, les éléments suivants s'appliquent :

- Lorsque vous activez le mode de suivi d'une fonction, les autorisations de suivi sont automatiquement associées au rôle d'exécution de la fonction. Si vous recevez une erreur indiquant qu'Lambda n'a pas pu ajouter la stratégie `AWSXrayWriteOnlyAccess` à votre rôle d'exécution de la fonction, connectez-vous à la console IAM sur <https://console.aws.amazon.com/iam/> et ajoutez manuellement la stratégie.

- Pour activer le suivi actif, accédez à l'onglet Configuration de votre fonction et cochez la case Enable active tracing.



Si vous utilisez les actions d'API Lambda [CreateFunction \(p. 410\)](#) ou [UpdateFunctionConfiguration \(p. 514\)](#) :

- Si vous souhaitez activer le mode de suivi, définissez le paramètre `TracingConfig` avec la propriété `Mode` sur `Active`. Là encore, toute nouvelle fonction a un mode de suivi défini sur `PassThrough` par défaut.
- Toute fonction Lambda nouvelle ou mise à jour possède sa version `$LATEST` définie sur la valeur que vous spécifiez.

Note

Vous recevez une erreur si vous n'avez pas ajouté le suivi des autorisations à votre rôle d'exécution de la fonction. Pour en savoir plus, consultez [Avant de commencer \(p. 233\)](#).

Emission des segments de suivi depuis une fonction Lambda

Pour chaque appel suivi, Lambda émet le segment de service Lambda et tous ses sous-segments. En outre, Lambda émet le segment de fonction Lambda et le sous-segment init. Ces segments sont émis indépendamment de la durée d'exécution de la fonction et sans modifications du code ou des bibliothèques supplémentaires requises. Si vous souhaitez que les suivis X-Ray de la fonction Lambda incluent des segments personnalisés, des annotations ou des sous-segments pour les appels en aval, il peut s'avérer nécessaire d'inclure des bibliothèques supplémentaires et d'annoter votre code.

Notez qu'un code d'instrumentation doit être implémenté dans le gestionnaire de la fonction Lambda, et non dans le code d'initialisation.

Les exemples suivants expliquent comment procéder dans les programmes d'exécution pris en charge :

- [Instrumentation du code Python dans AWS Lambda \(p. 311\)](#)

- Instrumentation du code Node.js dans AWS Lambda (p. 299)
- Instrumentation du code Java dans AWS Lambda (p. 340)
- ??? (p. 354)

Démon AWS X-Ray dans l'environnement Lambda

Le [démon AWS X-Ray](#) est une application logicielle qui recueille des données de segment brutes et les relaie vers le service AWS X-Ray. Le démon fonctionne conjointement avec les kits SDK AWS X-Ray afin que les données envoyées par les kits SDK puissent atteindre le service X-Ray.

Lorsque vous suivez votre fonction Lambda, le démon X-Ray est exécuté automatiquement dans l'environnement Lambda afin de collecter des données de suivi et de les envoyer à X-Ray. Lors du suivi, le démon X-Ray consomme un maximum de 16 Mo ou 3 % de l'allocation de mémoire de votre fonction. Par exemple, si vous allouez 128 Mo de mémoire à votre fonction Lambda, le démon X-Ray a 16 Mo de l'allocation de mémoire de votre fonction. Si vous allouez 1024 Mo à votre fonction Lambda, le démon X-Ray a 31 Mo de mémoire allouée (3 %). Pour plus d'informations, consultez [Le démon AWS X-Ray](#).

Note

Lambda essaiera de mettre hors service le démon X-Ray pour éviter de dépasser la limite de mémoire de votre fonction. Par exemple, supposons que vous ayez attribué 128 Mo de mémoire à votre fonction Lambda, le démon X-Ray aura 16 Mo de mémoire allouée. Votre fonction Lambda peut alors utiliser une allocation de mémoire de 112 Mo. Toutefois, si votre fonction dépasse 112 Mo, le démon X-Ray sera interrompu afin d'éviter de lancer une erreur de mémoire insuffisante.

Utilisation des variables d'environnement pour communiquer avec AWS X-Ray

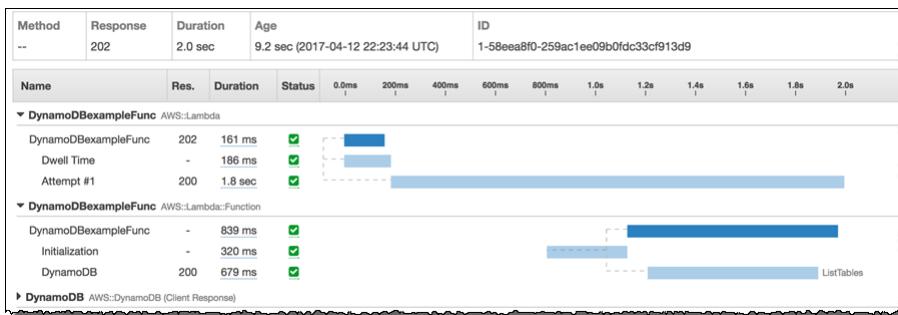
AWS Lambda utilise des variables d'environnement pour faciliter la communication avec le démon X-Ray et configurer le kit SDK X-Ray.

- `_X_AMZN_TRACE_ID` : contient l'en-tête de suivi, qui comprend la décision d'échantillonnage, l'ID de suivi et l'ID de segment parent. (Pour en savoir plus sur ces propriétés, consultez [En-tête de suivi](#).) Si Lambda reçoit un en-tête de suivi lorsque votre fonction est appelée, cet en-tête sera utilisé pour alimenter la variable d'environnement `_X_AMZN_TRACE_ID`. Si un en-tête de suivi n'a pas été reçu, Lambda en génère un pour vous.
- `AWS_XRAY_CONTEXT_MISSING` : le kit SDK X-Ray utilise cette variable pour déterminer son comportement si votre fonction tente d'enregistrer des données X-Ray, mais qu'aucun en-tête de suivi n'est disponible. Lambda règle cette valeur sur `LOG_ERROR` par défaut.
- `AWS_XRAY_DAEMON_ADDRESS` : cette variable d'environnement expose l'adresse du démon X-Ray au format suivant : `ADRESSE_IP:PORT`. Vous pouvez utiliser l'adresse du démon X-Ray pour envoyer des données de suivi directement au démon X-Ray, sans utiliser le kit SDK X-Ray.

Suivis Lambda dans la console AWS X-Ray : exemples

L'exemple suivant montre les suivis Lambda pour deux fonctions Lambda différentes. Chaque suivi présente une structure de suivi pour un autre type d'appel : synchrone et asynchrone.

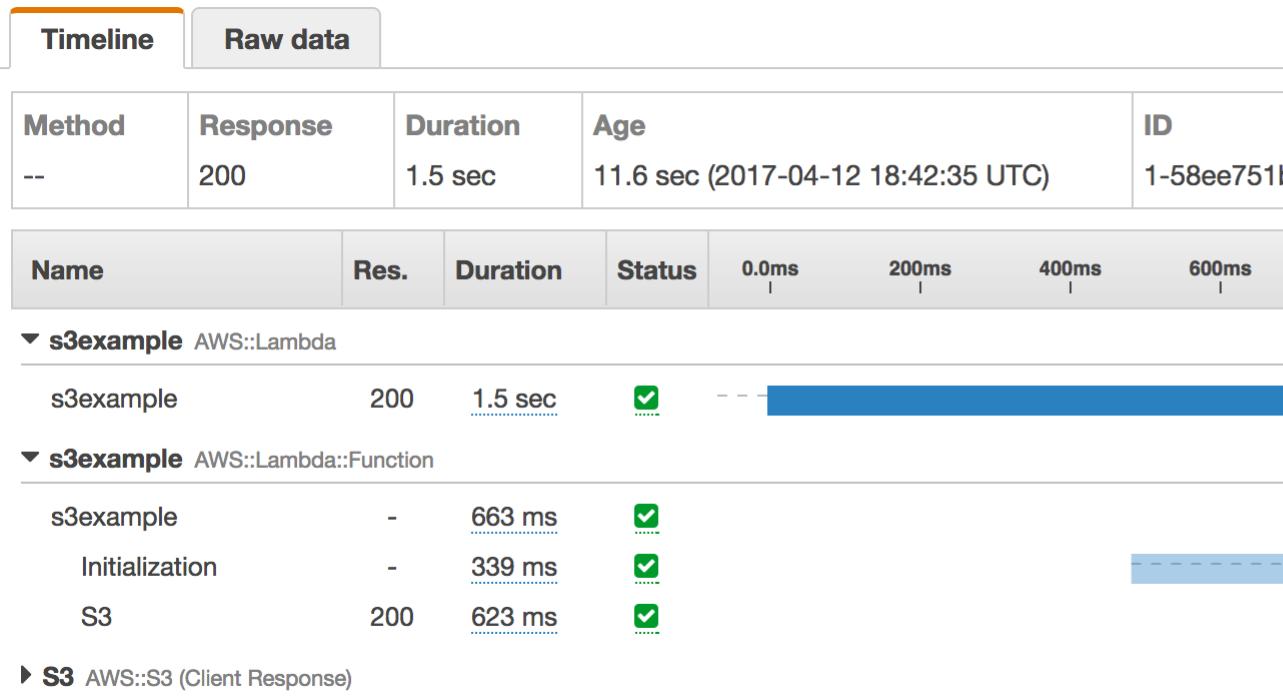
- Asynchrone : l'exemple suivant montre une requête Lambda asynchrone avec un appel réussi et un appel en aval vers DynamoDB.



Le segment de service Lambda encapsule le temps de réponse, qui correspond au temps de renvoi d'une réponse (par exemple, 202) au client. Il inclut les sous-segments pour le temps passé dans la file d'attente du service Lambda (temps d'arrêt) et pour chaque tentative d'appel. (Une seule tentative d'appel s'affiche dans l'exemple précédent.) Chaque sous-segment de tentative dans le segment de service disposera d'un segment de fonction utilisateur correspondant. Dans cet exemple, la fonction utilisateur contient deux sous-segments : le sous-segment d'initialisation représentant la fonction d'initialisation du code qui est exécuté avant le gestionnaire et un sous-segment d'appel en aval représentant un appel `ListTables` à DynamoDB.

Les codes d'état et les messages d'erreur sont affichés pour chaque sous-segment d'appel et pour chaque appel en aval.

- Synchrone : l'exemple suivant illustre une requête synchrone avec un appel en aval à Amazon S3.



Le segment de service Lambda capture la totalité du temps passé par la requête dans le service Lambda. Le segment de service aura un segment de fonction d'utilisateur correspondant. Dans cet exemple, le segment de la fonction utilisateur contient un sous-segment représentant le code d'initialisation de la fonction (code exécuté avant le gestionnaire), et un sous-segment représentant l'appel `PutObject` à Amazon S3.

Note

Si vous souhaitez suivre les appels HTTP, vous devez utiliser un client HTTP. Pour plus d'informations, consultez [Suivi des appels vers des services web HTTP en aval à l'aide du kit SDK X-Ray pour Java](#) ou [Suivi des appels vers des services web HTTP en aval à l'aide du kit SDK X-Ray pour Node.js](#).

Administration des applications basées sur Lambda

AWS Lambda s'intègre à de nombreux outils d'administration proposés par AWS, y compris le balisage AWS, AWS CloudTrail et AWS IAM. Les sections ci-dessous contiennent des recommandations sur la gestion des applications basées sur Lambda, y compris l'organisation des applications basées sur Lambda à l'aide de balises et l'audit de l'activité de votre compte AWS à l'aide de CloudTrail. Elles présentent en outre le modèle de sécurité AWS en rapport avec la sécurisation des applications basées sur Lambda. Nous évoquerons également une tâche d'administration unique à AWS Lambda, qui consiste à gérer le comportement des exécutions simultanées d'une fonction Lambda.

Les sections ci-dessous contiennent des recommandations sur l'organisation et le suivi de vos appels de la fonction Lambda, et présentent le modèle de sécurité AWS en rapport avec la sécurisation des applications basées sur Lambda :

Rubriques

- [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#)
- [Journalisation des appels d'API AWS Lambda avec AWS CloudTrail \(p. 272\)](#)
- [Balisage des fonctions Lambda \(p. 275\)](#)
- [Gestion de la simultanéité \(p. 279\)](#)
- [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 283\)](#)
- [Limites AWS Lambda \(p. 288\)](#)

Authentification et contrôle d'accès pour AWS Lambda

L'accès à AWS Lambda requiert des informations d'identification qu'AWS peut utiliser pour authentifier vos demandes. Ces informations d'identification doivent avoir des autorisations pour accéder aux ressources AWS, par exemple une fonction AWS Lambda ou un compartiment Amazon S3. Les sections suivantes fournissent des détails sur la façon dont vous pouvez utiliser [AWS Identity and Access Management \(IAM\)](#) et Lambda pour contribuer à sécuriser vos ressources en contrôlant qui peut y accéder :

- [Authentification \(p. 238\)](#)
- [Contrôle d'accès \(p. 240\)](#)

Authentification

Vous pouvez utiliser les types d'identité suivants pour accéder à AWS :

- Utilisateur racine d'un compte AWS – Lorsque vous créez un compte AWS, vous commencez avec une seule identité de connexion disposant d'un accès complet à tous les services et ressources AWS du compte. Cette identité est appelée utilisateur racine du compte AWS et elle est accessible après

connexion à l'aide de l'adresse e-mail et du mot de passe utilisés pour la création du compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes, y compris pour les tâches administratives. Au lieu de cela, respectez la [bonne pratique qui consiste à avoir recours à l'utilisateur racine uniquement pour créer le premier utilisateur IAM](#). Ensuite, mettez en sécurité les informations d'identification utilisateur racine et utilisez-les pour effectuer uniquement certaines tâches de gestion des comptes et des services.

- Utilisateur IAM – Un [utilisateur IAM](#) est une identité au sein de votre compte AWS qui dispose d'autorisations personnalisées spécifiques (par exemple, des autorisations pour créer une fonction dans Lambda). Vous pouvez utiliser un nom d'utilisateur et un mot de passe IAM pour vous connecter aux pages Web AWS sécurisées telles que [AWS Management Console](#), les [forums de discussion AWS](#) et le [AWS Support Center](#).

En plus de générer un nom utilisateur et un mot de passe, vous pouvez générer des [clés d'accès](#) pour chaque utilisateur. Vous pouvez utiliser ces clés lorsque vous accédez aux services AWS par programmation, soit par le biais d'un [kit SDK](#) soit à l'aide d'[AWS Command Line Interface \(CLI\)](#). Les outils de l'interface de ligne de commande et les kits SDK utilisent les clés d'accès pour signer de façon cryptographique votre demande. Si vous n'utilisez pas les outils AWS, vous devez signer la demande vous-même. Lambda supports prend en charge Signature Version 4, un protocole permettant l'authentification des demandes d'API entrantes. Pour plus d'informations sur l'authentification des demandes, consultez la rubrique [Processus de signature Signature Version 4](#) dans la documentation AWS General Reference.

- Rôle IAM – Un [rôle IAM](#) est une identité IAM que vous pouvez créer dans votre compte et qui dispose d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Un rôle IAM vous permet d'obtenir des clés d'accès temporaires qui peuvent être utilisées pour accéder aux ressources et services AWS. Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :
 - Accès d'utilisateurs fédérés – Au lieu de créer un utilisateur IAM, vous pouvez utiliser des identités d'utilisateur existantes d'AWS Directory Service, l'annuaire utilisateurs de votre entreprise, ou d'un fournisseur d'identité web. Ces derniers sont appelés utilisateurs fédérés. AWS attribue un rôle à un utilisateur fédéré lorsque l'accès est demandé via un [fournisseur d'identité](#). Pour plus d'informations sur les utilisateurs fédérés, consultez [Utilisateurs fédérés et rôles](#) dans le manuel IAM Guide de l'utilisateur.
 - Accès d'un service AWS – Vous pouvez utiliser un rôle IAM de votre compte pour autoriser un autre service AWS à accéder aux ressources de votre compte. Par exemple, vous pouvez créer un rôle qui permet à Amazon Redshift d'accéder à un compartiment Amazon S3 en votre nom, puis de charger les données stockées dans le compartiment dans un cluster Amazon Redshift. Pour plus d'informations, consultez la rubrique [Création d'un rôle pour déléguer des autorisations à un service AWS](#) dans le IAM Guide de l'utilisateur.
 - Applications qui s'exécutent sur Amazon EC2 – Vous pouvez utiliser un rôle IAM pour gérer des informations d'identification temporaires pour les applications qui s'exécutent sur une instance EC2 et effectuent des demandes d'API AWS. Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un rôle AWS à une instance EC2 et le rendre disponible à toutes les applications associées, vous pouvez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez [Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2](#) dans le IAM Guide de l'utilisateur.

Contrôle d'accès

Vous pouvez avoir des informations d'identification valides pour authentifier vos demandes, mais à moins d'avoir les autorisations requises, vous ne pouvez pas créer de ressources AWS Lambda ni accéder à de telles ressources. Par exemple, vous devez disposer d'autorisations pour créer une fonction Lambda, ajouter une source d'événement et publier une version de votre fonction Lambda.

Les sections suivantes décrivent comment gérer les autorisations pour AWS Lambda. Nous vous recommandons de commencer par lire la présentation.

- [Présentation de la gestion des autorisations d'accès à vos ressources AWS Lambda \(p. 240\)](#)
- [Utilisation des stratégies basées sur une identité \(stratégies IAM\) pour AWS Lambda \(p. 244\)](#)
- [Utilisation de stratégies basées sur une ressource pour AWS Lambda \(stratégies de fonction Lambda\) \(p. 257\)](#)

Présentation de la gestion des autorisations d'accès à vos ressources AWS Lambda

Chaque ressource AWS appartient à un compte AWS et les autorisations permettant de créer des ressources et d'y accéder sont régies par les stratégies d'autorisation. Un administrateur de compte peut attacher des stratégies d'autorisation aux identités IAM (c'est-à-dire aux utilisateurs, groupes et rôles), et certains services (tels que AWS Lambda) prennent également en charge l'attachement de stratégies d'autorisation aux ressources.

Note

Un administrateur de compte (ou utilisateur administrateur) est un utilisateur doté des priviléges d'administrateur. Pour plus d'informations, consultez [Meilleures pratiques pour IAM](#) dans le IAM Guide de l'utilisateur.

Lorsque vous accordez des autorisations, vous décidez qui doit les obtenir, à quelles ressources ces autorisations s'appliquent et les actions spécifiques que vous souhaitez autoriser sur ces ressources.

Rubriques

- [Ressources et opérations AWS Lambda \(p. 240\)](#)
- [Présentation de la propriété des ressources \(p. 241\)](#)
- [Gestion de l'accès aux ressources \(p. 241\)](#)
- [Spécification des éléments d'une stratégie : actions, effets, ressources et mandataires \(p. 243\)](#)
- [Spécification des conditions dans une stratégie \(p. 244\)](#)

Ressources et opérations AWS Lambda

Dans AWS Lambda, les principales ressources sont une fonction Lambda et un mappage de source d'événement. Vous créez un mappage de source d'événement dans le modèle Pull AWS Lambda pour associer une fonction Lambda à une source d'événement. Pour plus d'informations, consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

AWS Lambda prend également en charge des types de ressources supplémentaires, l'alias et la version. Cependant, vous ne pouvez créer des alias et des versions que dans le contexte d'une fonction Lambda existante. Celles-ci sont appelées sous-ressources.

Ces ressources et sous-ressources ont des noms ARN (Amazon Resource Name) uniques associés, comme cela est illustré dans le tableau suivant.

Type de ressource	Format ARN
Fonction	arn:aws:lambda: <i>region:account-id:function:function-name</i>
Alias de la fonction	arn:aws:lambda: <i>region:account-id:function:function-name:alias-name</i>
Version de la fonction	arn:aws:lambda: <i>region:account-id:function:function-name:version</i>
Mappage de source d'événement	arn:aws:lambda: <i>region:account-id:event-source-mapping:event-source-mapping-id</i>

AWS Lambda fournit un ensemble d'opérations à utiliser avec les ressources Lambda. Pour obtenir la liste des opérations disponibles, consultez [Actions \(p. 392\)](#).

Présentation de la propriété des ressources

Le propriétaire de la ressource est le compte AWS qui l'a créée. En d'autres termes, le propriétaire de la ressource est le compte AWS de l'entité mandataire (le compte racine, un utilisateur IAM ou un rôle IAM) qui authentifie la demande créatrice de la ressource. Les exemples suivants illustrent comment cela fonctionne :

- Si vous utilisez les informations d'identification du compte racine de votre compte AWS pour créer une fonction Lambda, votre compte AWS est le propriétaire de la ressource (dans Lambda, la ressource est la fonction Lambda).
- Si vous créez un utilisateur IAM dans votre compte AWS que vous autorisez à créer une fonction Lambda, il pourra créer cette fonction. Toutefois, votre compte AWS, auquel l'utilisateur appartient, est propriétaire de la ressource de fonction Lambda.
- Si vous créez un rôle IAM dans votre compte AWS avec des autorisations pour créer une fonction Lambda, quiconque capable d'assumer ce rôle peut créer ce type de fonction. Votre compte AWS, auquel appartient le rôle, est le propriétaire de la ressource de fonction Lambda.

Gestion de l'accès aux ressources

Une stratégie d'autorisation décrit qui a accès à quoi. La section suivante explique les options disponibles pour créer des stratégies d'autorisation.

Note

Cette section décrit l'utilisation d'IAM dans le contexte d'AWS Lambda. Elle ne fournit pas d'informations détaillées sur le service IAM. Pour accéder à la documentation complète d'IAM, consultez [Qu'est-ce qu'IAM ?](#) dans le IAM Guide de l'utilisateur. Pour plus d'informations sur la syntaxe des stratégies IAM et les descriptions, consultez [Référence de stratégie AWS IAM](#) dans le IAM Guide de l'utilisateur.

Les stratégies attachées à une identité sont qualifiées de stratégies basées sur l'identité (stratégies IAM) et les stratégies attachées à une ressource sont qualifiées de stratégies basées sur une ressource. AWS Lambda prend uniquement en charge les stratégies basées sur une identité (stratégies IAM) et les stratégies basées sur une ressource.

Rubriques

- [Stratégies basées sur une identité \(stratégies IAM\) \(p. 242\)](#)

- Stratégies basées sur une ressource (stratégies de fonction Lambda) (p. 242)

Stratégies basées sur une identité (stratégies IAM)

Vous pouvez attacher des stratégies à des identités IAM. Par exemple, vous pouvez effectuer les opérations suivantes :

- Attacher une stratégie d'autorisation à un utilisateur ou à un groupe dans votre compte – Un administrateur de compte peut utiliser une stratégie d'autorisation associée à un utilisateur particulier pour autoriser cet utilisateur à créer une fonction Lambda.
- Attacher une stratégie d'autorisation à un rôle (accorder des autorisations entre comptes) – Vous pouvez attacher une stratégie d'autorisation basée sur une identité à un rôle IAM pour accorder des autorisations entre comptes. Par exemple, l'administrateur dans le Compte A peut créer un rôle pour accorder des autorisations entre comptes à un autre compte AWS (par exemple, le Compte B) ou à un service AWS comme suit :
 1. L'administrateur du Compte A crée un rôle IAM et attache une stratégie d'autorisation à ce rôle qui accorde des autorisations sur les ressources dans le Compte A.
 2. L'administrateur du Compte A attache une stratégie d'approbation au rôle identifiant le Compte B comme mandataire pouvant assumer ce rôle.
 3. L'administrateur du Compte B peut alors déléguer des autorisations pour assumer le rôle à tous les utilisateurs figurant dans le Compte B. Cela autorise les utilisateurs du Compte B à créer des ressources ou à y accéder dans le Compte A. Le mandataire dans la stratégie d'approbation peut également être un mandataire de service AWS si vous souhaitez accorder à un service AWS des autorisations pour assumer ce rôle.

Pour plus d'informations sur l'utilisation d'IAM pour déléguer des autorisations, consultez [Gestion d'accès](#) dans le IAM Guide de l'utilisateur.

Voici un exemple de stratégie qui accorde les autorisations requises pour l'action `lambda>ListFunctions` au niveau de toutes les ressources. Dans l'implémentation actuelle, Lambda ne prend pas en charge l'identification de ressources spécifiques à l'aide des ARN de ressource (également appelés autorisations au niveau des ressources) pour certaines actions d'API. Vous devez donc spécifier un caractère générique (*).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Pour plus d'informations sur l'utilisation des stratégies basées sur l'identité avec Lambda, consultez [Utilisation des stratégies basées sur une identité \(stratégies IAM\) pour AWS Lambda](#) (p. 244). Pour plus d'informations sur les utilisateurs, les groupes, les rôles et les autorisations, consultez [Identités \(utilisateurs, groupes et rôles\)](#) dans le IAM Guide de l'utilisateur.

Stratégies basées sur une ressource (stratégies de fonction Lambda)

Chaque fonction Lambda peut être associée à des stratégies d'autorisations basées sur une ressource. Pour Lambda, une fonction Lambda est la ressource principale et ces stratégies sont appelées

stratégies de fonction Lambda. Vous pouvez utiliser une stratégie de fonction Lambda pour accorder des autorisations entre comptes comme alternative à l'utilisation des stratégies basées sur l'identité avec les rôles IAM. Par exemple, vous pouvez accorder à Amazon S3 les autorisations requises pour appeler la fonction Lambda en ajoutant simplement les autorisations à la stratégie de cette fonction au lieu de créer un rôle IAM.

Important

Les stratégies de fonction Lambda sont principalement utilisées lorsque vous configurez une source d'événement dans AWS Lambda pour accorder à un service ou à une source d'événement les autorisations requises pour appeler la fonction Lambda (consultez [Invoke \(p. 452\)](#)). Il y a exception à cette règle lorsqu'une source d'événement (par exemple, Amazon DynamoDB ou Kinesis) utilise le modèle Pull, où les autorisations sont gérées dans le rôle d'exécution de la fonction Lambda à la place. Pour plus d'informations, consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

Voici un exemple de stratégie de fonction Lambda comportant une seule instruction. L'instruction accorde au service Amazon S3 l'autorisation principale pour l'action `lambda:InvokeFunction` au niveau de la fonction Lambda `HelloWorld`. Cette condition garantit que le compartiment dans lequel l'événement s'est produit appartient au même compte que celui qui possède la fonction Lambda.

```
{  
  "Policy":{  
    "Version":"2012-10-17",  
    "Statement": [  
      {  
        "Effect":"Allow",  
        "Principal":{  
          "Service":"s3.amazonaws.com"  
        },  
        "Action":"lambda:InvokeFunction",  
        "Resource":"arn:aws:lambda:region:account-id:function:HelloWorld",  
        "Sid":"65bafc90-6a1f-42a8-a7ab-8aa9bc877985",  
        "Condition":{  
          "StringEquals":{  
            "AWS:SourceAccount": "account-id"  
          },  
          "ArnLike":{  
            "AWS:SourceArn": "arn:aws:s3:::ExampleBucket"  
          }  
        }  
      }  
    ]  
  }  
}
```

Pour plus d'informations sur l'utilisation des stratégies basées sur une ressource avec Lambda, consultez [Utilisation de stratégies basées sur une ressource pour AWS Lambda \(stratégies de fonction Lambda\) \(p. 257\)](#). Pour plus d'informations sur les rôles IAM (stratégies basées sur l'identité) par rapport aux stratégies basées sur une ressource, consultez [En quoi les rôles IAM diffèrent-ils des stratégies basées sur une ressource](#) dans le IAM Guide de l'utilisateur.

Spécification des éléments d'une stratégie : actions, effets, ressources et mandataires

Pour chaque ressource AWS Lambda (voir [Ressources et opérations AWS Lambda \(p. 240\)](#)), le service définit un ensemble d'opérations d'API (voir [Actions \(p. 392\)](#)). Pour accorder des autorisations pour ces opérations d'API, Lambda définit un ensemble d'actions que vous pouvez spécifier dans une stratégie. Notez que l'exécution d'une opération d'API peut exiger des autorisations pour plusieurs actions. Lors de

L'octroi des autorisations pour des actions spécifiques, vous identifiez également la ressource au niveau de laquelle les actions sont autorisées ou refusées.

Voici les éléments les plus élémentaires d'une stratégie :

- **Ressource** – Dans une stratégie, vous utilisez un Amazon Resource Name (ARN) pour identifier la ressource à laquelle la stratégie s'applique. Pour plus d'informations, consultez [Ressources et opérations AWS Lambda \(p. 240\)](#).
- **Action** – Vous utilisez des mots clés d'action pour identifier les opérations de ressource que vous voulez accorder ou refuser. Par exemple, l'autorisation `lambda:InvokeFunction` permet à l'utilisateur d'effectuer l'opération `Invoke` de AWS Lambda.
- **Effet** – Vous spécifiez l'effet produit lorsque l'utilisateur demande l'action spécifique—qui peut être un accord ou un refus. Si vous n'accordez pas explicitement l'accès pour (autoriser) une ressource, l'accès est implicitement refusé. Vous pouvez aussi explicitement refuser l'accès à une ressource, ce que vous pouvez faire afin de vous assurer qu'un utilisateur n'y a pas accès, même si une stratégie différente accorde l'accès.
- **Mandataire** – Dans les stratégies basées sur une identité (stratégies IAM), l'utilisateur auquel la stratégie est attachée est le mandataire implicite. Pour les stratégies basées sur une ressource, vous spécifiez l'utilisateur, le compte, le service ou une autre entité qui doit recevoir les autorisations (s'applique uniquement aux stratégies basées sur une ressource).

Pour plus d'informations sur la syntaxe des stratégies IAM et obtenir des descriptions, consultez [Présentation des stratégies IAM AWS](#) dans le manuel IAM Guide de l'utilisateur.

Pour visualiser un tableau répertoriant toutes les actions d'API d'AWS Lambda, ainsi que les ressources auxquelles elles s'appliquent, consultez [Autorisations d'API Lambda : référence des actions, ressources et conditions \(p. 263\)](#).

Spécification des conditions dans une stratégie

Lorsque vous accordez des autorisations, vous pouvez utiliser le langage des stratégies IAM afin de spécifier les conditions définissant à quel moment une stratégie doit prendre effet. Par exemple, il est possible d'appliquer une stratégie après seulement une date spécifique. Pour plus d'informations sur la spécification de conditions dans un langage de stratégie, consultez [Condition](#) dans le IAM Guide de l'utilisateur.

Pour exprimer des conditions, vous utilisez des clés de condition prédéfinies. Il n'existe pas de clés de condition spécifiques à Lambda. Il existe toutefois des clés de condition à l'échelle d'AWS que vous pouvez utiliser selon vos besoins. Pour une liste complète des clés à l'échelle d'AWS, consultez [Clés disponibles pour les conditions](#) dans le IAM Guide de l'utilisateur.

Utilisation des stratégies basées sur une identité (stratégies IAM) pour AWS Lambda

Cette rubrique fournit des exemples de stratégies basées sur une identité dans lesquelles un administrateur de compte peut attacher des stratégies d'autorisation aux identités IAM (c'est-à-dire aux utilisateurs, groupes et rôles).

Important

Nous vous recommandons tout d'abord d'examiner les rubriques de présentation qui détaillent les concepts de base et les options disponibles pour gérer l'accès à vos ressources AWS Lambda. Pour plus d'informations, consultez [Présentation de la gestion des autorisations d'accès à vos ressources AWS Lambda \(p. 240\)](#).

Les sections de cette rubrique couvrent les sujets suivants :

- [Autorisations requises pour utiliser la console AWS Lambda \(p. 245\)](#)
- [Stratégies gérées par AWS \(prédéfinies\) pour AWS Lambda \(p. 246\)](#)
- [Exemples de stratégies gérées par le client \(p. 246\)](#)

Un exemple de stratégie d'autorisation est exposé ci-dessous.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CreateFunctionPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>CreateFunction"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "PermissionToPassAnyRole",  
            "Effect": "Allow",  
            "Action": [  
                "iam:PassRole"  
            ],  
            "Resource": "arn:aws:iam:account-id:role/*"  
        }  
    ]  
}
```

La stratégie possède deux énoncés:

- La première instruction accorde les autorisations pour l'action AWS Lambda action (`lambda>CreateFunction`) au niveau d'une ressource en utilisant l'ARN (Amazon Resource Name) de la fonction Lambda. Actuellement, AWS Lambda ne prend pas en charge les autorisations requises pour cette action particulière au niveau des ressources. Par conséquent, la stratégie spécifie un caractère générique (*) comme étant la valeur `Resource`.
- La deuxième instruction accorde des autorisations pour l'action IAM (`iam:PassRole`) sur les rôles IAM. Le caractère générique (*) à la fin de la valeur `Resource` signifie que l'instruction autorise l'action `iam:PassRole` sur n'importe quel rôle IAM. Pour limiter cette autorisation à un rôle spécifique, remplacez le caractère générique (*) dans la ressource ARN par un nom de rôle spécifique.

La stratégie ne spécifie pas l'élément `Principal` car, dans une stratégie basée sur une identité, vous ne spécifiez pas le mandataire qui obtient l'autorisation. Quand vous attachez une stratégie à un utilisateur, l'utilisateur est le mandataire implicite. Lorsque vous attachez une stratégie d'autorisation à un rôle IAM, le mandataire identifié dans la stratégie d'approbation de ce rôle obtient les autorisations.

Pour accéder à un tableau répertoriant toutes les actions d'API d'AWS Lambda et les ressources et conditions auxquelles elles s'appliquent, consultez [Autorisations d'API Lambda : référence des actions, ressources et conditions \(p. 263\)](#).

Autorisations requises pour utiliser la console AWS Lambda

La console AWS Lambda fournit un environnement intégré vous permettant de créer et de gérer les fonctions Lambda. Cette console offre de nombreuses fonctionnalités et flux de travail qui nécessitent souvent des autorisations pour créer une fonction Lambda en plus des autorisations propres à l'API documentées dans la section [Autorisations d'API Lambda : référence des actions, ressources et conditions](#)

conditions (p. 263). Pour en savoir plus sur ces autorisations de console supplémentaires, consultez la section [Autorisations requises pour utiliser la console AWS Lambda \(p. 250\)](#).

Stratégies gérées par AWS (prédéfinies) pour AWS Lambda

AWS peut remédier à de nombreux cas d'utilisation courants en fournissant des stratégies IAM autonomes qui sont créées et administrées par AWS. Les stratégies gérées octroient les autorisations requises dans les cas d'utilisation courants et vous évitez d'avoir à réfléchir aux autorisations qui sont requises. Pour plus d'informations, consultez [Stratégies gérées par AWS](#) dans le IAM Guide de l'utilisateur.

Les stratégies gérées par AWS suivantes, que vous pouvez attacher aux utilisateurs de votre compte, sont propres à AWS Lambda et sont regroupées par scénario d'utilisation :

- `AWSLambdaReadOnlyAccess` : accorde l'accès en lecture seule aux ressources AWS Lambda. Notez que cette stratégie n'accorde pas l'autorisation pour l'action `lambda:InvokeFunction`. Si vous souhaitez qu'un utilisateur appelle une fonction Lambda, vous pouvez également associer la stratégie gérée AWS `AWSLambdaRole`.
- `AWSLambdaFullAccess` : accorde un accès total aux ressources AWS Lambda.
- `AWSLambdaRole` : accorde les autorisations requises pour appeler les fonctions Lambda.

Vous pouvez consulter ces stratégies d'autorisations en vous connectant à la console IAM et en y recherchant des stratégies spécifiques.

En outre, il existe des autres stratégies gérées par AWS qui peuvent être utilisées avec le rôle IAM (rôle d'exécution) que vous spécifiez au moment de la création d'une fonction Lambda. Pour plus d'informations, consultez [Modèle d'autorisations AWS Lambda \(p. 260\)](#).

Vous pouvez également créer vos propres stratégies IAM personnalisées afin d'accorder des autorisations pour des actions d'API et des ressources AWS Lambda. Vous pouvez associer ces stratégies personnalisées aux utilisateurs ou groupes IAM qui requièrent ces autorisations ou aux rôles d'exécution personnalisés (rôles IAM) que vous créez pour vos fonctions Lambda.

Exemples de stratégies gérées par le client

Cette section fournit un ensemble d'exemples de stratégies que vous pouvez associer à un utilisateur. Si vous créez des stratégies pour la première fois, nous vous recommandons de commencer par concevoir un utilisateur IAM dans votre compte et de lui associer les stratégies dans l'ordre, comme décrit dans les étapes de cette section.

Vous pouvez utiliser la console pour vérifier les effets de chaque stratégie lorsque vous les associez à l'utilisateur. Au départ, l'utilisateur n'a pas les autorisations requises et ne peut donc effectuer aucune action dans la console. À mesure que vous lui associez des stratégies, vous pouvez vérifier que l'utilisateur peut exécuter diverses actions dans la console.

Nous vous recommandons d'utiliser deux fenêtres de navigateur : une pour créer l'utilisateur et pour accorder les autorisations, l'autre pour vous connecter à AWS Management Console avec les informations d'identification de l'utilisateur et pour vérifier les autorisations que vous lui accordez.

Pour voir des exemples illustrant comment créer un rôle IAM utilisable comme rôle d'exécution pour la fonction Lambda, consultez [Création de rôles IAM](#) dans le IAM Guide de l'utilisateur.

Étapes d'exemple

- [Étape 1 : Création d'un utilisateur IAM \(p. 247\)](#)
- [Étape 2 : Permettre à un utilisateur d'afficher une liste des fonctions Lambda \(p. 247\)](#)
- [Étape 3 : Permettre à un utilisateur d'afficher les détails d'une fonction Lambda \(p. 247\)](#)
- [Étape 4 : Permettre à un utilisateur d'appeler une fonction Lambda \(p. 248\)](#)

- Étape 5 : Permettre à un utilisateur de surveiller une fonction Lambda et d'accéder à CloudWatch Logs (p. 248)
- Étape 6 : Permettre à un utilisateur de créer une fonction Lambda (p. 249)

Étape 1 : Création d'un utilisateur IAM

Vous devez d'abord créer un utilisateur IAM, l'ajouter à un groupe IAM possédant des autorisations d'administration, puis lui attribuer ces autorisations. Vous pouvez alors accéder à AWS à l'aide d'une URL spéciale et des informations d'identification de cet utilisateur IAM.

Pour obtenir des instructions, consultez [Création de votre premier groupe d'utilisateurs et d'administrateurs IAM](#) dans le IAM Guide de l'utilisateur.

Étape 2 : Permettre à un utilisateur d'afficher une liste des fonctions Lambda

Un utilisateur IAM de votre compte doit disposer des autorisations pour l'action `lambda>ListFunctions` avant d'avoir accès au contenu de la console. Lorsque vous accordez ces autorisations, la console peut afficher la liste des fonctions Lambda du compte AWS créé dans la région AWS spécifique dont l'utilisateur fait partie.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Étape 3 : Permettre à un utilisateur d'afficher les détails d'une fonction Lambda

Un utilisateur peut sélectionner une fonction Lambda et en afficher les détails (par exemple, les alias, les versions et autres informations de configuration), dans la mesure où il dispose des autorisations pour les actions AWS Lambda suivantes :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DisplayFunctionDetailsPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListVersionsByFunction",  
                "lambda>ListAliases",  
                "lambda:GetFunction",  
                "lambda:GetFunctionConfiguration",  
                "lambda>ListEventSourceMappings",  
                "lambda:GetPolicy"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Étape 4 : Permettre à un utilisateur d'appeler une fonction Lambda

Si vous voulez permettre à un utilisateur d'appeler manuellement une fonction, vous devez lui accorder les autorisations pour l'action `lambda:InvokeFunction`, comme illustré ci-après :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "InvokePermission",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Étape 5 : Permettre à un utilisateur de surveiller une fonction Lambda et d'accéder à CloudWatch Logs

Quand un utilisateur appelle une fonction Lambda, AWS Lambda l'exécute et renvoie les résultats. L'utilisateur a besoin d'autorisations supplémentaires pour surveiller la fonction Lambda.

Pour permettre à l'utilisateur d'accéder aux métriques CloudWatch de la fonction Lambda dans l'onglet Surveillance de la console ou dans la grille de la page d'accueil de cette console, vous devez accorder les autorisations suivantes :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchPermission",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:GetMetricStatistics"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Pour permettre à un utilisateur de cliquer sur les liens d'accès à CloudWatch Logs dans la console AWS Lambda et d'afficher la sortie d'un journal dans CloudWatch Logs, vous devez accorder les autorisations suivantes :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchLogsPerms",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatchlog:DescribeLogGroups",  
                "cloudwatchlog:DescribeLogStreams",  
                "cloudwatchlog:GetLogEvents"  
            ],  
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/lambda/*"  
        }  
    ]  
}
```

```
    ]  
}
```

Étape 6 : Permettre à un utilisateur de créer une fonction Lambda

Si vous voulez permettre à un utilisateur de créer une fonction Lambda, vous devez lui accorder les autorisations suivantes. Les autorisations pour les actions liées à IAM sont obligatoires parce que, lorsque l'utilisateur crée une fonction Lambda, il doit sélectionner un rôle d'exécution IAM qu'assume AWS Lambda pour exécuter cette fonction Lambda.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingRolesAndPolicies",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListRolePolicies",  
                "iam>ListRoles"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "CreateFunctionPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>CreateFunction"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "PermissionToPassAnyRole",  
            "Effect": "Allow",  
            "Action": [  
                "iam>PassRole"  
            ],  
            "Resource": "arn:aws:iam:account-id:role/*"  
        }  
    ]  
}
```

Si vous voulez permettre à un utilisateur de créer un rôle IAM lors de la création d'une fonction Lambda, celui-ci nécessite les autorisations requises pour effectuer l'action `iam:PutRolePolicy`, comme illustré ci-après :

```
{  
    "Sid": "CreateARole",  
    "Effect": "Allow",  
    "Action": [  
        "iam>CreateRole",  
        "iam>CreatePolicy",  
        "iam>PutRolePolicy",  
        "iam>AttachRolePolicy"  
    ],  
    "Resource": "arn:aws:iam:account-id:role/*"  
}
```

Important

Chaque rôle IAM est associé à une stratégie qui accorde des autorisations spécifiques à ce rôle. Que l'utilisateur utilise un rôle existant ou qu'il en crée un, il doit disposer des autorisations

requises pour toutes les actions octroyées dans la stratégie d'autorisations associée au rôle. Vous devez accorder à l'utilisateur les autorisations supplémentaires en conséquence.

Autorisations requises pour utiliser la console AWS Lambda

Pour tirer parti de l'environnement intégré fourni par la console AWS Lambda, les utilisateurs doivent souvent avoir plus d'autorisations que celles décrites dans le tableau de références pour l'API, en fonction de ce que vous voulez leur permettre de faire. Pour plus d'informations sur les opérations d'API Lambda, consultez [Autorisations d'API Lambda : référence des actions, ressources et conditions \(p. 263\)](#).

Supposons, par exemple, que vous autorisiez un utilisateur IAM à créer une fonction Lambda pour traiter les événements générés par des objets Amazon S3. Pour lui permettre de configurer Amazon S3 comme source des événements, la liste déroulante de la console affiche une liste de vos compartiments. Toutefois, la console ne peut afficher la liste de compartiments que si l'utilisateur connecté dispose des autorisations requises pour les actions Amazon S3 appropriées.

Les sections suivantes décrivent les autorisations supplémentaires requises pour les différents points d'intégration.

Si vous n'êtes pas habitué à gérer les autorisations, nous vous conseillons de commencer par la procédure où nous vous expliquons comment créer un utilisateur IAM, comment lui octroyer des autorisations supplémentaires et comment vérifier que ces dernières fonctionnent avec la console AWS Lambda (consultez [Exemples de stratégies gérées par le client \(p. 246\)](#)).

Rubriques

- [Amazon API Gateway \(p. 250\)](#)
- [Amazon CloudWatch Events \(p. 251\)](#)
- [Amazon CloudWatch Logs \(p. 252\)](#)
- [Amazon Cognito \(p. 252\)](#)
- [Amazon DynamoDB \(p. 253\)](#)
- [Amazon Kinesis Data Streams \(p. 254\)](#)
- [Amazon S3 \(p. 255\)](#)
- [Amazon SNS \(p. 255\)](#)
- [AWS IoT \(p. 256\)](#)

Note

Toutes ces stratégies accordent les autorisations spécifiques aux services AWS permettant d'appeler une fonction Lambda. L'utilisateur qui configure cette intégration doit disposer des autorisations requises pour appeler la fonction Lambda. Dans le cas contraire, il ne pourra pas définir la configuration. Vous pouvez attacher la stratégie gérée `AWSLambdaRole` à l'utilisateur pour lui fournir ces autorisations.

Amazon API Gateway

Lorsque vous configurez un point de terminaison API dans la console, cette dernière permet plusieurs appels d'API API Gateway. Ces appels nécessitent les autorisations liées à l'action `apigateway:*`, comme illustré ci-après :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ApiGatewayPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "apigateway:*"
```

```
        ],
        "Resource": "*"
    },
    {
        "Sid": "AddPermissionToFunctionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission",
            "lambda:RemovePermission",
            "lambda:GetPolicy"
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:*"
    },
    {
        "Sid": "ListEventSourcePerm",
        "Effect": "Allow",
        "Action": [
            "lambda>ListEventSourceMappings"
        ],
        "Resource": "*"
    }
]
```

Amazon CloudWatch Events

Vous pouvez planifier quand appeler une fonction Lambda. Une fois que vous sélectionnez une règle CloudWatch Events (ou que vous en définissez une), AWS Lambda crée une cible dans CloudWatch qui appelle la fonction Lambda. Pour que la création de la cible fonctionne, vous devez accorder les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EventPerms",
            "Effect": "Allow",
            "Action": [
                "events:PutRule",
                "events>ListRules",
                "events>ListRuleNamesByTarget",
                "events:PutTargets",
                "events:RemoveTargets",
                "events:DescribeRule",
                "events:TestEventPattern",
                "events>ListTargetsByRule",
                "events>DeleteRule"
            ],
            "Resource": "arn:aws:events:region:account-id:*"
        },
        {
            "Sid": "AddPermissionToFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission",
                "lambda:GetPolicy"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:*"
        }
    ]
}
```

Amazon CloudWatch Logs

Vous pouvez configurer le service Amazon CloudWatch Logs pour qu'il publie les événements et appelle la fonction Lambda. Lorsque vous configurez ce service comme source d'événements, la console répertorie les groupes de journaux de votre compte. Pour que cela se produise, vous devez toutefois accorder les autorisations `logs:DescribeLogGroups`, comme illustré ci-après :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchLogsPerms",  
            "Effect": "Allow",  
            "Action": [  
                "logs:FilterLogEvents",  
                "logs:DescribeLogGroups",  
                "logs:PutSubscriptionFilter",  
                "logs:DescribeSubscriptionFilters",  
                "logs>DeleteSubscriptionFilter",  
                "logs:TestMetricFilter"  
            ],  
            "Resource": "arn:aws:logs:region:account-id:*"  
        },  
        {  
            "Sid": "AddPermissionToFunctionPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:AddPermission",  
                "lambda:RemovePermission",  
                "lambda:GetPolicy"  
            ],  
            "Resource": "arn:aws:lambda:region:account-id:function:/*"  
        },  
        {  
            "Sid": "ListEventSourceMappingsPerms",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListEventSourceMappings"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Note

Les autorisations supplémentaires indiquées sont nécessaires pour la gestion des filtres d'abonnement.

Amazon Cognito

La console répertorie les groupes d'identités de votre compte. Une fois que vous sélectionnez un groupe, vous pouvez le configurer pour utiliser le déclencheur de synchronisation Cognito en tant que type d'événement source. Pour ce faire, vous devez accorder les autorisations supplémentaires suivantes :

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CognitoPerms1",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction",  
                "lambda:ListEventSourceMappings"  
            ]  
        }  
    ]  
}
```

```
        "cognito-identity>ListIdentityPools"
    ],
    "Resource": [
        "arn:aws:cognito-identity:region:account-id:/*"
    ]
},
{
    "Sid": "CognitoPerms2",
    "Effect": "Allow",
    "Action": [
        "cognito-sync:GetCognitoEvents",
        "cognito-sync:SetCognitoEvents"
    ],
    "Resource": [
        "arn:aws:cognito-sync:region:account-id:/*"
    ]
},
{
    "Sid": "AddPermissionToFunctionPolicy",
    "Effect": "Allow",
    "Action": [
        "lambda:AddPermission",
        "lambda:RemovePermission",
        "lambda:GetPolicy"
    ],
    "Resource": "arn:aws:lambda:region:account-id:function:/*"
},
{
    "Sid": "ListEventSourcePerms",
    "Effect": "Allow",
    "Action": [
        "lambda>ListEventSourceMappings"
    ],
    "Resource": "*"
}
]
```

Amazon DynamoDB

La console répertorie toutes les tables de votre compte. Une fois que vous sélectionnez une table, elle vérifie s'il existe un flux DynamoDB pour celle-ci. Si ce n'est pas le cas, elle crée le flux. Si vous souhaitez que l'utilisateur puisse configurer un flux DynamoDB comme source d'événement pour une fonction Lambda, vous devez accorder les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DDBpermissions1",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable"
            ],
            "Resource": "arn:aws:dynamodb:region:account-id:table/*"
        },
        {
            "Sid": "DDBpermissions2",
            "Effect": "Allow",
            "Action": [
                "dynamodb>ListStreams",
                "dynamodb>ListTables"
            ]
        }
    ]
}
```

```
        ],
        "Resource": "*"
    },
    {
        "Sid": "LambdaGetPolicyPerm",
        "Effect": "Allow",
        "Action": [
            "lambda:GetPolicy"
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:*"
    },
    {
        "Sid": "LambdaEventSourcePerms",
        "Effect": "Allow",
        "Action": [
            "lambda>CreateEventSourceMapping",
            "lambda>DeleteEventSourceMapping",
            "lambda:GetEventSourceMapping",
            "lambda>ListEventSourceMappings",
            "lambda:UpdateEventSourceMapping"
        ],
        "Resource": "*"
    }
]
```

Important

Pour permettre à une fonction Lambda de lire un flux DynamoDB, le rôle d'exécution qui lui est associé doit avoir les autorisations appropriées. Avant de pouvoir les accorder, assurez-vous que l'utilisateur dispose également des mêmes autorisations. Pour ce faire, associez la stratégie prédéfinie AWSLambdaDynamoDBExecutionRole à l'utilisateur, puis au rôle d'exécution.

Amazon Kinesis Data Streams

La console répertorie tous les flux Kinesis de votre compte. Une fois que vous sélectionnez un flux, la console crée des mappages de source d'événement dans AWS Lambda. Pour que cela fonctionne, vous devez accorder les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionForDescribeStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/*"
        },
        {
            "Sid": "PermissionForListStreams",
            "Effect": "Allow",
            "Action": [
                "kinesis>ListStreams"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PermissionForGetFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:GetPolicy"
            ]
        }
]
```

```
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:*
```

```
    },
    {
        "Sid": "LambdaEventSourcePerms",
        "Effect": "Allow",
        "Action": [
            "lambda>CreateEventSourceMapping",
            "lambda>DeleteEventSourceMapping",
            "lambda:GetEventSourceMapping",
            "lambda>ListEventSourceMappings",
            "lambda:UpdateEventSourceMapping"
        ],
        "Resource": "*"
    }
}
```

Amazon S3

La console remplit préalablement la liste des compartiments du compte AWS et recherche l'emplacement de chaque compartiment. Lorsque vous configurez Amazon S3 comme source d'événements, la console met à jour la configuration des notifications de compartiment. Pour que cela fonctionne, vous devez accorder les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetBucketNotification",
                "s3:PutBucketNotification",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AddPermissionToFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda>AddPermission",
                "lambda>RemovePermission"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:*
```

Amazon SNS

La console répertorie les rubriques Amazon Simple Notification Service (Amazon SNS) de votre compte. Une fois que vous sélectionnez une rubrique, AWS Lambda abonne la fonction Lambda à cette rubrique Amazon SNS. Pour que cela fonctionne, vous devez accorder les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SNSPerms",
```

```
"Effect": "Allow",
"Action": [
    "sns>ListSubscriptions",
    "sns>ListSubscriptionsByTopic",
    "sns>ListTopics",
    "sns>Subscribe",
    "sns>Unsubscribe"
],
"Resource": "arn:aws:sns:region:account-id:*"
},
{
    "Sid": "AddPermissionToFunctionPolicy",
    "Effect": "Allow",
    "Action": [
        "lambda>AddPermission",
        "lambda>RemovePermission",
        "lambda>GetPolicy"
    ],
    "Resource": "arn:aws:lambda:region:account-id:function:*"
},
{
    "Sid": "LambdaListESMappingsPerms",
    "Effect": "Allow",
    "Action": [
        "lambda>ListEventSourceMappings"
    ],
    "Resource": "*"
}
]
```

AWS IoT

La console répertorie toutes les règles AWS IoT. Une fois que vous sélectionnez une règle, la console remplit le reste des informations associées à cette règle dans l'interface utilisateur. Si vous sélectionnez une règle existante, la console la met à jour avec les informations requises, de telle sorte que les événements sont envoyés à AWS Lambda. Vous pouvez également créer une règle. Pour ce faire, l'utilisateur doit avoir les autorisations supplémentaires suivantes :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "IoTperms",
            "Effect": "Allow",
            "Action": [
                "iot>GetTopicRule",
                "iot>CreateTopicRule",
                "iot>ReplaceTopicRule"
            ],
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
            "Sid": "IoTlistTopicRulePerms",
            "Effect": "Allow",
            "Action": [
                "iot>ListTopicRules"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaPerms",
            "Effect": "Allow",
            "Action": [
                "lambda>InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:rule-name"
        }
    ]
}
```

```
    "lambda:AddPermission",
    "lambda:RemovePermission",
    "lambda:GetPolicy"
],
"Resource": "arn:aws:lambda:region:account-id:function:*
}
}
```

Utilisation de stratégies basées sur une ressource pour AWS Lambda (stratégies de fonction Lambda)

Une fonction Lambda est l'une des ressources d'AWS Lambda. Vous pouvez ajouter des autorisations à la stratégie associée à une fonction Lambda. Les stratégies d'autorisations associées aux fonctions Lambda sont des stratégies basées sur les ressources (ou des stratégies de fonction Lambda dans Lambda). Vous utilisez les stratégies de fonction Lambda pour gérer les autorisations liées à l'appel d'une fonction Lambda (consultez [Invoke \(p. 452\)](#)).

Important

Avant que vous procédez à la création de stratégies basées sur une ressource, nous vous recommandons d'examiner les rubriques de présentation qui détaillent les concepts de base et les options disponibles pour gérer l'accès à vos ressources AWS Lambda. Pour plus d'informations, consultez [Présentation de la gestion des autorisations d'accès à vos ressources AWS Lambda \(p. 240\)](#).

Les stratégies de fonction Lambda sont principalement utilisées lorsque vous configurez une source d'événement dans AWS Lambda pour accorder à un service ou à une source d'événement les autorisations requises pour appeler la fonction Lambda (consultez [Invoke \(p. 452\)](#)). Il y a exception à cette règle lorsqu'une source d'événement (par exemple, Amazon DynamoDB ou Kinesis) utilise le modèle Pull, où les autorisations sont gérées dans le rôle d'exécution de la fonction Lambda à la place. Pour plus d'informations, consultez [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

Les stratégies de fonction Lambda facilitent également l'octroi d'autorisations entre comptes pour appeler la fonction Lambda. Supposons que vous souhaitez accorder des autorisations entre comptes (au service Amazon S3, par exemple) pour appeler la fonction Lambda. Au lieu de créer un rôle IAM pour accorder des autorisations entre comptes, vous pouvez ajouter les autorisations appropriées dans une stratégie de fonction Lambda.

Note

Si l'application personnalisée et la fonction Lambda qu'elle appelle appartiennent au même compte AWS, vous n'avez pas besoin d'accorder des autorisations explicites via la stratégie associée à la fonction Lambda.

AWS Lambda fournit les opérations d'API suivantes pour gérer une stratégie d'autorisations associée à une fonction Lambda :

- [AddPermission \(p. 398\)](#)
- [GetPolicy \(p. 450\)](#)
- [RemovePermission \(p. 493\)](#)

Note

La console AWS Lambda représente le moyen le plus simple de gérer les sources d'événements et leurs autorisations dans une stratégie de fonction Lambda. Si la console du service AWS prend en charge la configuration du mappage de source d'événement, vous pouvez également utiliser cette console. Lorsque vous configurez de nouvelles sources d'événements ou lorsque vous

modifiez des sources d'événements existantes, la console modifie automatiquement la stratégie d'autorisations associée à la fonction Lambda.

Vous pouvez utiliser la console pour consulter la stratégie de fonction en choisissant l'onglet Configuration, puis l'icône clé. La console ne prend pas en charge la modification directe des autorisations d'une stratégie de fonction. Vous devez utiliser soit l'AWS CLI, soit les kits SDK AWS. Voici des exemples AWS CLI des opérations d'API répertoriées précédemment dans cette rubrique :

Exemples

- [Exemple 1 : Autoriser Amazon S3 à appeler une fonction Lambda \(p. 258\)](#)
- [Exemple 2 : Autoriser Amazon API Gateway à appeler une fonction Lambda \(p. 259\)](#)
- [Exemple 3 : Permettre à une application utilisateur créée par un autre compte AWS d'appeler une fonction Lambda \(scénario entre comptes\) \(p. 259\)](#)
- [Exemple 4 : Récupérer une stratégie de fonction Lambda \(p. 260\)](#)
- [Exemple 5 : Supprimer les autorisations d'une stratégie de fonction Lambda \(p. 260\)](#)
- [Exemple 6 : Utiliser des stratégies d'autorisation pour les versions et les alias de fonction Lambda \(p. 260\)](#)

Exemple 1 : Autoriser Amazon S3 à appeler une fonction Lambda

Pour accorder à Amazon S3 l'autorisation d'appeler une fonction Lambda, vous configurez les autorisations comme suit :

- Spécifiez `s3.amazonaws.com` en tant que valeur `principal`.
- Spécifiez `lambda:InvokeFunction` comme l'`action` pour laquelle vous accordez des autorisations.

Pour vous assurer que l'événement est généré à partir d'un compartiment spécifique appartenant à un compte AWS donné, définissez également les éléments suivants :

- Spécifiez l'ARN du compartiment comme valeur `source-arn` afin de limiter les événements d'un compartiment spécifique.
- Spécifiez l'ID de compte AWS qui possède le compartiment, afin de s'assurer que le compartiment désigné appartient au compte.

L'exemple de commande suivant (AWS CLI) ajoute une autorisation à la stratégie de fonction `helloworld` Lambda, qui permet à Amazon S3 d'appeler la fonction.

```
aws lambda add-permission \
--region region \
--function-name helloworld \
--statement-id 1 \
--principal s3.amazonaws.com \
--action lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket \
--source-account 111111111111 \
--profile adminuser
```

Cet exemple présume que le profil `adminuser` (qui dispose des autorisations complètes) ajoute cette autorisation. Par conséquent, le paramètre `--profile` définit le profil `adminuser`.

En réponse, AWS Lambda renvoie le code JSON suivant. La valeur `Statement` est une version de chaîne JSON correspondant à l'instruction ajoutée à la stratégie de la fonction Lambda.

```
{
```

```
"Statement": "{\"Condition\":{\"StringEquals\":{\"AWS:SourceAccount\": \"111111111111\"},\n    \"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:s3:::examplebucket\"}},\n    \"Action\":[\"lambda:InvokeFunction\"],\n    \"Resource\":\"arn:aws:lambda:us-west-2:111111111111:function:helloworld\",\n    \"Effect\":\"Allow\", \"Principal\":{\"Service\":\"s3.amazonaws.com\"},\n    \"Sid\":\"1\"}\n}
```

Pour plus d'informations sur le modèle push, consultez la section [Mappage de source d'événement AWS Lambda \(p. 74\)](#).

Exemple 2 : Autoriser Amazon API Gateway à appeler une fonction Lambda

Pour permettre à Amazon API Gateway d'appeler une fonction Lambda, effectuez les opérations suivantes :

- Spécifiez `apigateway.amazonaws.com` en tant que valeur `principal`.
- Spécifiez `lambda:InvokeFunction` comme l'action pour laquelle vous accordez des autorisations.
- Spécifiez l'ARN du point de terminaison API Gateway comme valeur `source-arn`.

L'exemple de commande suivant (AWS CLI) ajoute une autorisation à la stratégie de fonction `helloworld` Lambda, qui permet à API Gateway d'appeler la fonction.

```
aws lambda add-permission \
--region region \
--function-name helloworld \
--statement-id 5 \
--principal apigateway.amazonaws.com \
--action lambda:InvokeFunction \
--source-arn arn:aws:execute-api:region:account-id:api-id/stage/method/resource-path \
--profile adminuser
```

En réponse, AWS Lambda renvoie le code JSON suivant. La valeur `Statement` est une version de chaîne JSON correspondant à l'instruction ajoutée à la stratégie de la fonction Lambda.

```
{\n    \"Statement\": \"{\\"Condition\\\":{\\\"ArnLike\\\":{\\\"AWS:SourceArn\\\":\\\"arn:aws:apigateway:us-east-1::my-api-id:/test/petsstorewalkthrough/pets\\\"}},\n        \\\"Action\\\":[\\\"lambda:InvokeFunction\\\"],\n        \\\"Resource\\\":\\\"arn:aws:lambda:us-west-2:account-id:function:helloworld\\\",\\n        \\\"Effect\\\\":\"Allow\\\",\\n        \\\"Principal\\\":{\\\"Service\\\\":\\\"apigateway.amazonaws.com\\\"},\\n        \\\"Sid\\\\":\\\"5\\\"}\n    \"\n}
```

Exemple 3 : Permettre à une application utilisateur créée par un autre compte AWS d'appeler une fonction Lambda (scénario entre comptes)

Pour accorder des autorisations à un autre compte AWS (c'est-à-dire, pour créer un scénario entre comptes), vous spécifiez l'ID de compte AWS en tant que valeur `principal`, comme illustré dans la commande d'AWS CLI :

```
aws lambda add-permission \  
--region region \  
--function-name helloworld \  
--statement-id 3 \  
--principal 111111111111 \  
--action lambda:InvokeFunction \  
--profile adminuser
```

En réponse, AWS Lambda renvoie le code JSON suivant. La valeur **Statement** est une version de chaîne JSON correspondant à l'instruction ajoutée à la stratégie de la fonction Lambda.

```
{  
    "Statement": "{\"Action\": [\"lambda:InvokeFunction\"],  
                  \"Resource\": \"arn:aws:lambda:us-west-2:account-id:function:helloworld\",  
                  \"Effect\": \"Allow\",  
                  \"Principal\": {\"AWS\": \"account-id\"},  
                  \"Sid\": \"3\"}"  
}
```

Exemple 4 : Récupérer une stratégie de fonction Lambda

Pour récupérer une stratégie de fonction Lambda, vous utilisez la commande `get-policy` :

```
aws lambda get-policy \  
--function-name example \  
--profile adminuser
```

Exemple 5 : Supprimer les autorisations d'une stratégie de fonction Lambda

Pour supprimer les autorisations d'une stratégie de fonction Lambda, utilisez la `remove-permission`, en spécifiant le nom de la fonction et l'ID de l'instruction :

```
aws lambda remove-permission \  
--function-name example \  
--statement-id 1 \  
--profile adminuser
```

Exemple 6 : Utiliser des stratégies d'autorisation pour les versions et les alias de fonction Lambda

Pour plus d'informations sur les stratégies d'autorisations des versions et des alias de fonction Lambda, consultez [Versions, alias et stratégies de ressources \(p. 51\)](#).

Modèle d'autorisations AWS Lambda

Pour que les applications complètes basées sur AWS Lambda fonctionnent, vous devez gérer diverses autorisations. Par exemple :

- A l'exception des services basés sur les flux (Amazon Kinesis Data Streams et DynamoDB) ou les files d'attente Amazon SQS, vous devez accorder aux sources d'événements les autorisations requises pour appeler la fonction AWS Lambda.

- Pour interroger les sources d'événements basées sur les flux (flux Amazon Kinesis Data Streams et DynamoDB et files d'attente Amazon SQS), AWS Lambda interroge la ressource en votre nom et lit les nouveaux enregistrements. Pour l'activer, vous devez autoriser AWS Lambda à accéder aux nouveaux enregistrements. En retour, AWS Lambda appelle une fonction Lambda abonnée à cette source d'événement pour traiter l'événement.
- Pour toute autre source d'événement qui appelle directement la fonction Lambda, vous devez autoriser la source d'événement à appeler votre fonction AWS Lambda.

Les sections suivantes décrivent la gestion des autorisations.

Rubriques

- [Gestion des autorisations : utilisation d'un rôle IAM \(rôle d'exécution\) \(p. 261\)](#)
- [Gestion des autorisations : utilisation d'une stratégie de fonction Lambda \(p. 262\)](#)

Gestion des autorisations : utilisation d'un rôle IAM (rôle d'exécution)

Un rôle IAM (rôle d'exécution) est associé à chaque fonction Lambda. Vous spécifiez le rôle IAM lors de la création de la fonction Lambda. Les autorisations que vous accordez à ce rôle déterminent ce qu'AWS Lambda peut faire lorsqu'il assume ce rôle. Vous pouvez accorder deux types d'autorisations au rôle IAM :

- Si le code de la fonction Lambda accède à d'autres ressources AWS, notamment pour lire un objet à partir d'un compartiment S3 ou pour créer des journaux dans CloudWatch Logs, vous devez accorder au rôle les autorisations requises pour effectuer les actions Amazon S3 et CloudWatch appropriées.
- Si la source d'événement est basée sur les interrogations (Amazon Kinesis Data Streams, DynamoDB, Amazon SQS), AWS Lambda interroge ces ressources en votre nom. AWS Lambda a besoin d'autorisations pour interroger les flux ou une file d'attente afin de lire de nouveaux enregistrements. Pour l'activer, vous devez autoriser AWS Lambda à accéder aux nouveaux enregistrements. En retour, AWS Lambda appelle une fonction Lambda abonnée à cette source d'événement pour traiter l'événement.

Pour plus d'informations sur les rôles IAM, consultez [Rôles \(délégation et fédération\)](#) dans le IAM Guide de l'utilisateur.

Important

L'utilisateur qui crée le rôle IAM autorise en fait AWS Lambda à assumer ce rôle. L'utilisateur doit être autorisé à effectuer l'action `iam:PassRole`. Si vous êtes un utilisateur avec des priviléges d'administrateur, vous n'avez rien à faire pour configurer les autorisations requises pour l'action `iam:PassRole`, car les administrateurs disposent de toutes les autorisations, y compris pour l'action `iam:PassRole`.

Pour simplifier le processus de création d'un rôle d'exécution, AWS Lambda met à votre disposition les stratégies d'autorisations (prédéfinies) suivantes, gérées par AWS. Ces stratégies incluent des autorisations courantes pour des scénarios spécifiques :

- `AWSLambdaBasicExecutionRole` : accorde uniquement les autorisations requises pour les actions Amazon CloudWatch Logs d'écriture de journaux. Vous pouvez utiliser cette stratégie si la fonction Lambda n'accède à aucune autre ressource AWS que celle qui permet d'écrire des journaux.
- `AWSLambdaKinesisExecutionRole` : accorde les autorisations requises pour les actions Amazon Kinesis Data Streams et CloudWatch Logs. Si vous écrivez une fonction Lambda de sorte qu'elle traite les événements de flux Kinesis, vous pouvez y associer cette stratégie d'autorisations.

- `AWSLambdaDynamoDBExecutionRole` : accorde les autorisations requises pour les actions de flux DynamoDB et les actions CloudWatch Logs. Si vous écrivez une fonction Lambda de sorte qu'elle traite les événements de flux DynamoDB, vous pouvez y associer cette stratégie d'autorisations.
- `AWSLambdaVPCAccessExecutionRole` : accorder les autorisations requises pour les actions Amazon Elastic Compute Cloud (Amazon EC2) afin de gérer les interfaces réseau Elastic (ENI). Si vous écrivez une fonction Lambda pour accéder aux ressources d'un VPC dans le service Amazon Virtual Private Cloud (Amazon VPC), vous pouvez y associer cette stratégie d'autorisations. Cette stratégie accorde également les autorisations requises pour les actions CloudWatch Logs d'écriture de journaux.

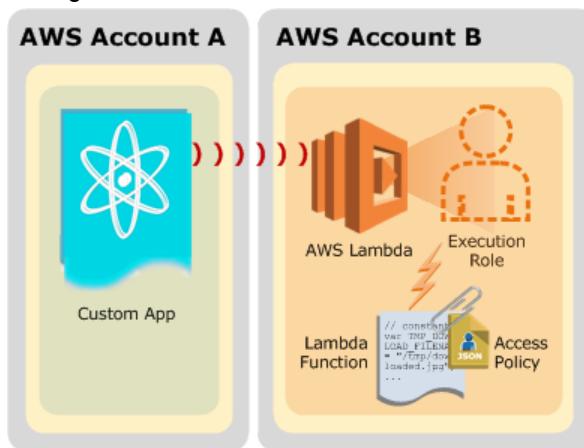
Vous pouvez trouver ces stratégies d'autorisations gérées par AWS dans la console IAM. Recherchez ces stratégies pour voir les autorisations qu'elles accordent.

Gestion des autorisations : utilisation d'une stratégie de fonction Lambda

Toutes les sources d'événements prises en charge, hormis les sources basées sur les interrogations qui sont basées sur les flux (Kinesis et DynamoDB) ou sur les messages (Amazon SQS), appellent la fonction Lambda à l'aide du modèle Push, à condition que vous ayez accordé les autorisations nécessaires. Par exemple, si vous voulez qu'Amazon S3 appelle la fonction Lambda lorsque des objets sont créés dans un compartiment, Amazon S3 nécessite les autorisations requises pour appeler cette fonction.

Vous pouvez accorder ces autorisations via des stratégies de fonction. (Pour obtenir une liste d'exemples, consultez [Modèles de stratégie \(p. 265\)](#).) AWS Lambda offre des API vous permettant de gérer les autorisations dans une stratégie de fonction. Pour obtenir un exemple, veuillez consulter [AddPermission \(p. 398\)](#).

Vous pouvez aussi accorder des autorisations entre comptes à l'aide de la stratégie de fonction. Par exemple, si une application définie par l'utilisateur et la fonction Lambda qu'elle appelle appartiennent au même compte AWS, vous n'avez pas besoin d'accorder des autorisations explicites. Dans le cas contraire, le compte AWS qui possède la fonction Lambda doit inclure des autorisations entre comptes dans la stratégie d'autorisations associée Lambda.



Au lieu d'utiliser une stratégie de fonction Lambda, vous pouvez créer un autre rôle IAM qui accorde aux sources d'événements (par exemple, Amazon S3 ou DynamoDB) les autorisations requises pour appeler la fonction Lambda. Cependant, vous trouverez peut-être les stratégies de ressources plus simples à configurer, d'autant plus qu'elles vous permettent également de déterminer facilement les sources d'événements autorisées à appeler votre fonction Lambda.

Pour plus d'informations sur les stratégies de fonction Lambda, consultez [Utilisation de stratégies basées sur une ressource pour AWS Lambda \(stratégies de fonction Lambda\) \(p. 257\)](#). Pour plus d'informations sur les autorisations Lambda, consultez [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

Autorisations d'API Lambda : référence des actions, ressources et conditions

Lorsque vous configurez des stratégies d'autorisation d'écriture et de [Contrôle d'accès \(p. 240\)](#) que vous pouvez attacher à une entité IAM (stratégies basées sur une identité), vous pouvez utiliser le tableau ci-dessous comme référence. La liste comprend chaque opération d'API AWS Lambda, les actions correspondantes pour lesquelles vous pouvez accorder des autorisations d'exécution de l'action, la ressource AWS pour laquelle vous pouvez accorder des autorisations et les clés de condition des actions d'API spécifiées. Vous spécifiez les actions dans le champ `Action` de la stratégie, la valeur de ressource dans le champ `Resource` de la stratégie et une clé de condition dans le champ `Condition keys` de la stratégie.

Pour spécifier une action, utilisez le préfixe `lambda:` suivi du nom de l'opération d'API (par exemple, `lambda:CreateFunction`).

Note

Vous pouvez également utiliser des stratégies basées sur les ressources pour accorder les autorisations pour l'API AWS Lambda `Invoke` du tableau suivant. Pour plus d'informations, consultez [Utilisation de stratégies basées sur une ressource pour AWS Lambda \(stratégies de fonction Lambda\) \(p. 257\)](#).

Vous pouvez utiliser des clés de condition à l'échelle d'AWS dans les stratégies AWS Lambda pour exprimer des conditions. Pour une liste complète des clés à l'échelle d'AWS, consultez [Clés disponibles pour les conditions](#) dans le IAM Guide de l'utilisateur.

AWS Lambda propose également des clés de condition prédéfinies à un ensemble limité d'opérations d'API. Par exemple, vous pouvez :

- Limiter l'accès selon l'ARN (Amazon Resource Name) de la fonction Lambda pour les opérations suivantes :
 - `CreateEventSourceMapping`
 - `DeleteEventSourceMapping`
 - `UpdateEventSourceMapping`

Voici un exemple de stratégie qui applique cette condition :

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "DeleteEventSourceMappingPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:DeleteEventSourceMapping"
        ],
        "Resource": "arn:aws:lambda:region:account-id:event-source-mapping:UUID",
        "Condition": {"StringEquals": {"lambda:FunctionArn": "arn:aws:lambda:region:account-id:function:function-name"}}
    }
]
```

- Limitez le mappage en fonction du mandataire de service AWS pour les opérations suivantes :

- AddPermission
- RemovePermission

Voici un exemple de stratégie qui applique cette condition :

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "AddPermissionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission"
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:funcion-name",
        "Condition": {"StringEquals": {"lambda:Principal": "s3.amazonaws.com"}}
    }
]
```

API AWS Lambda et autorisations requises pour les actions

[AddPermission \(p. 398\)](#)

Action(s) : lambda:AddPermission

Ressource: arn:aws:lambda:**region:account-id:?:/***

[CreateEventSourceMapping \(p. 406\)](#)

Action(s) : lambda>CreateEventSourceMapping

Ressource: arn:aws:lambda:**region:account-id:?:**

[CreateFunction \(p. 410\)](#)

Action(s) : lambda>CreateFunction

Ressource: arn:aws:lambda:**region:account-id:?:**

[DeleteEventSourceMapping \(p. 420\)](#)

Action(s) : lambda>DeleteEventSourceMapping

Ressource: arn:aws:lambda:**region:account-id:?:**

[DeleteFunction \(p. 423\)](#)

Action(s) : lambda>DeleteFunction,

Ressource: arn:aws:lambda:**region:account-id:?:**

[GetEventSourceMapping \(p. 434\)](#)

Action(s) : lambda>GetEventSourceMapping

Ressource: arn:aws:lambda:**region:account-id:?:**

[GetFunction \(p. 437\)](#)

Action(s) : lambda>GetFunction

Ressource: arn:aws:lambda:**region:account-id:?:**

[GetFunctionConfiguration \(p. 440\)](#)

Action(s) : lambda:DescribeMountTargetSecurityGroups,

Ressource: arn:aws:lambda:**region:account-id:?**

[GetPolicy \(p. 450\)](#)

Action(s) : lambda:DescribeMountTargets

Ressource: arn:aws:lambda:**region:account-id:?**

[Invoke \(p. 452\)](#)

Action(s) : lambda:DescribeTags

Ressource: arn:aws:lambda:**region:account-id:?**

[InvokeAsync \(p. 457\)](#)

Action(s) : lambda:ModifyMountTargetSecurityGroups

Ressource: arn:aws:lambda:**region:account-id:?**

[ListEventSourceMappings \(p. 462\)](#)

Action(s) : lambda>ListEventSourceMappings

Ressource: arn:aws:lambda:**region:account-id:?**

[ListFunctions \(p. 465\)](#)

Action(s) : lambda>ListFunctions

Ressource: arn:aws:lambda:**region:account-id:?**

[RemovePermission \(p. 493\)](#)

Action(s) : lambda:RemovePermission

Ressource: arn:aws:lambda:**region:account-id:?**

[UpdateEventSourceMapping \(p. 504\)](#)

Action(s) : lambda:UpdateEventSourceMapping

Ressource: arn:aws:lambda:**region:account-id:?**

[UpdateFunctionCode \(p. 508\)](#)

Action(s) : lambda:UpdateFunctionCode

Ressource: arn:aws:lambda:**region:account-id:?**

[UpdateFunctionConfiguration \(p. 514\)](#)

Action(s) : lambda:UpdateFunctionConfiguration

Ressource: arn:aws:lambda:**region:account-id:?**

Modèles de stratégie

Lorsque vous créez une fonction AWS Lambda dans la console via l'un des plans, Lambda vous permet de créer un rôle pour votre fonction à partir d'une liste de modèles de stratégie Lambda. En sélectionnant

l'un de ces modèles, votre fonction Lambda crée automatiquement le rôle avec les autorisations requises associées à cette stratégie.

La section suivante répertorie les autorisations qui sont appliquées à chaque modèle de stratégie dans la liste Policy templates. Le nom des modèles de stratégie est basé sur les plans auxquels ils correspondent. Lambda renseigne automatiquement les éléments des espaces réservés (région et ID de compte, par exemple) avec les informations appropriées. Pour plus d'informations sur la création d'une fonction Lambda avec les modèles de stratégie, consultez la section [Créer une fonction Lambda avec la console \(p. 6\)](#).

Modèles standard

Les modèles suivants sont appliqués automatiquement selon le type de fonction Lambda que vous créez :

Basic : « Autorisations Lambda standard »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "logs>CreateLogGroup",  
            "Resource": "arn:aws:logs:region:accountId:*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": [  
                "arn:aws:logs:region:accountId:log-group:[ [ logGroups ] ]:*"  
            ]  
        }  
    ]  
}
```

VPCAccess : « Autorisations d'accès au VPC Lambda »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2>CreateNetworkInterface",  
                "ec2>DeleteNetworkInterface",  
                "ec2:DescribeNetworkInterfaces"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Kinesis : « Autorisation de l'observateur de flux Lambda Kinesis »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "kinesis:PutRecord",  
            "Resource": "arn:aws:kinesis:region:accountId:stream:myStreamName:*"  
        }  
    ]  
}
```

```

        "Action":"lambda:InvokeFunction",
        "Resource":"arn:aws:lambda:region:accountId:function:functionName*"
    },
    {
        "Effect":"Allow",
        "Action":"kinesis>ListStreams",
        "Resource":"arn:aws:kinesis:region:accountId:stream/*"
    },
    {
        "Effect":"Allow",
        "Action":[
            "kinesis:DescribeStream",
            "kinesis:GetRecords",
            "kinesis:GetShardIterator"
        ],
        "Resource":"arn:aws:kinesis:region:accountId:
stream/streamName"
    }
}
}

```

DynamoDB : « Autorisation de l'observateur de flux Lambda DynamoDB »

```

{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect":"Allow",
            "Action":"lambda:InvokeFunction",
            "Resource":"arn:aws:lambda:region:accountId:function:functionName*"
        },
        {
            "Effect":"Allow",
            "Action":[
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb>ListStreams"
            ],
            "Resource":"arn:aws:dynamodb:region:accountId:table/tableName/stream/*"
        }
    ]
}

```

Edge : « Autorisations Lambda de périphérique standard »

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs>CreateLogGroup",
                "logs>CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:***"
            ]
        }
    ]
}

```

RedrivePolicySNS : « Autorisations SNS de file d'attente de lettres mortes »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:region:accountId:topicName"  
        }  
    ]  
}
```

RedrivePolicySQS : « Autorisations SQS de file d'attente de lettres mortes »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:region:accountId:queueName"  
        }  
    ]  
}
```

Modèles de plan

Les modèles suivants sont sélectionnés lorsque vous choisissez un plan. Vous pouvez également les sélectionner dans la liste déroulante afin d'ajouter des autorisations supplémentaires.

CloudFormation : « Autorisation en lecture seule de la pile CloudFormation »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudformation:DescribeStacks"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

AMI : « Autorisations en lecture seule des AMI »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [
```

```
        "ec2:DescribeImages"
    ],
    "Resource": "*"
}
]
```

KMS : « Autorisations de déchiffrement KMS »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

S3 : « Autorisations en lecture seule d'un objet S3 »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::/*"
        }
    ]
}
```

Elasticsearch : « Autorisations Elasticsearch »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "es:ESHttpPost"
            ],
            "Resource": "*"
        }
    ]
}
```

SES : « Autorisations de retour SES »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "
```

```
        "Action": [
            "ses:SendBounce"
        ],
        "Resource": "*"
    }
}
```

TestHarness : « Autorisations Test Harness »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:region:accountId:table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:region:accountId:function:*
        }
    ]
}
```

Microservice : « Autorisations Microservice »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb>DeleteItem",
                "dynamodb>GetItem",
                "dynamodb>PutItem",
                "dynamodb>Scan",
                "dynamodb>UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:region:accountId:table/*"
        }
    ]
}
```

VPN : « Autorisations du moniteur de connexion VPN »

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        },
    ]
}
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "ec2:DescribeRegions",  
        "ec2:DescribeVpnConnections"  
    ],  
    "Resource": "*"  
}  
]  
}
```

SQS : « Autorisations de l'observateur SQS »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:DeleteMessage",  
                "sns:ReceiveMessage"  
            ],  
            "Resource": "arn:aws:sqs:/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "arn:aws:lambda:region:accountID:function: functionName*"  
        }  
    ]  
}
```

IoTButton : « Autorisations du bouton AWS IoT »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns>ListSubscriptionsByTopic",  
                "sns>CreateTopic",  
                "sns:SetTopicAttributes",  
                "sns:Subscribe",  
                "sns:Publish"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

RekognitionNoDataAccess : « Aucune autorisation de données Amazon Rekognition »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "rekognition:CompareFaces",  
        "rekognition:DetectFaces",  
        "rekognition:DetectLabels"  
    ],  
    "Resource": "*"  
}  
]  
}
```

RekognitionReadOnlyAccess : « Autorisations Amazon Rekognition de lecture seule »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rekognition>ListCollections",  
                "rekognition>ListFaces",  
                "rekognition>SearchFaces",  
                "rekognition>SearchFacesByImage"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

RekognitionWriteOnlyAccess : « Autorisations Amazon Rekognition en écriture seule »

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rekognition>CreateCollection",  
                "rekognition>IndexFaces"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Journalisation des appels d'API AWS Lambda avec AWS CloudTrail

AWS Lambda est intégré à AWS CloudTrail, un service qui enregistre les actions effectuées par un utilisateur, un rôle ou un service AWS dans AWS Lambda. CloudTrail capture tous les appels d'API pour

AWS Lambda en tant qu'événements. Les appels capturés incluent des appels de la console AWS Lambda et les appels de code vers les opérations d'API AWS Lambda. Si vous créez un journal de suivi, vous pouvez diffuser en continu les événements CloudTrail sur un compartiment Amazon S3, y compris les événements pour AWS Lambda. Si vous ne configurez pas de journal de suivi, vous pouvez toujours afficher les événements les plus récents dans la console CloudTrail dans Event history (Historique des événements). À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été envoyée à AWS Lambda, l'adresse IP source à partir de laquelle la demande a été effectuée, l'auteur de la demande et la date de la demande, ainsi que d'autres informations.

Pour en savoir plus sur CloudTrail, y compris la façon de le configurer et de l'activer, consultez le manuel [AWS CloudTrail User Guide](#).

Informations AWS Lambda dans CloudTrail

CloudTrail est activé sur votre compte AWS lorsque vous créez le compte. Quand une activité d'événement prise en charge a lieu dans AWS Lambda, elle est enregistrée dans un événement CloudTrail avec d'autres événements de services AWS dans Event history (Historique des événements). Vous pouvez afficher, rechercher et télécharger les événements récents dans votre compte AWS. Pour plus d'informations, consultez [Affichage des événements avec l'historique des événements CloudTrail](#).

Pour un enregistrement continu des événements dans votre compte AWS, y compris les événements pour AWS Lambda, créez un journal de suivi. Une journal de suivi permet à CloudTrail de livrer les fichiers journaux dans un compartiment Amazon S3. Par défaut, lorsque vous créez un journal de suivi dans la console, il s'applique à toutes les régions AWS. Le journal de suivi consigne les événements de toutes les régions dans la partition AWS et livre les fichiers journaux dans le compartiment Amazon S3 de votre choix. En outre, vous pouvez configurer d'autres services AWS pour analyser plus en profondeur les données d'événement collectées dans les journaux CloudTrail et agir sur celles-ci. Pour plus d'informations, consultez les ressources suivantes :

- [Présentation de la création d'un journal de suivi](#)
- [Intégrations et services pris en charge par CloudTrail](#)
- [Configuration des Notifications de Amazon SNS pour CloudTrail](#)
- [Réception de fichiers journaux CloudTrail de plusieurs régions](#) et [Réception de fichiers journaux CloudTrail de plusieurs comptes](#)

AWS Lambda prend en charge la journalisation des actions suivantes en tant qu'événements dans les fichiers journaux CloudTrail :

- [AddPermission \(p. 398\)](#)
- [CreateEventSourceMapping \(p. 406\)](#)
- [CreateFunction \(p. 410\)](#)

(Le paramètre `ZipFile` ne figure pas dans les journaux CloudTrail pour `CreateFunction`.)

- [DeleteEventSourceMapping \(p. 420\)](#)
- [DeleteFunction \(p. 423\)](#)
- [GetEventSourceMapping \(p. 434\)](#)
- [GetFunction \(p. 437\)](#)
- [GetFunctionConfiguration \(p. 440\)](#)
- [GetPolicy \(p. 450\)](#)
- [ListEventSourceMappings \(p. 462\)](#)
- [ListFunctions \(p. 465\)](#)
- [RemovePermission \(p. 493\)](#)

- `UpdateEventSourceMapping` (p. 504)
 - `UpdateFunctionCode` (p. 508)

(Le paramètre `ZipFile` ne figure pas dans les journaux CloudTrail pour `UpdateFunctionCode`.)

Chaque entrée du journal contient des informations sur la personne qui a généré la demande. Les informations d'identité d'utilisateur figurant dans le journal vous aident à déterminer si la demande a été effectuée à l'aide d'informations d'identification racine ou d'utilisateur IAM, à l'aide d'informations d'identification de sécurité temporaires pour un rôle ou un utilisateur fédéré, ou par un autre service AWS. Pour plus d'informations, reportez-vous au champ `userIdentity` dans le [Guide de référence des événements CloudTrail](#).

Vous pouvez stocker vos fichiers journaux dans votre compartiment aussi longtemps que vous le souhaitez, mais vous pouvez également définir des règles de cycle de vie Amazon S3 pour archiver ou supprimer automatiquement les fichiers journaux. Par défaut, vos fichiers journaux sont chiffrés à l'aide du chiffrement côté serveur (SSE) d'Amazon S3.

Vous pouvez décider d'utiliser CloudTrail pour publier des notifications Amazon SNS lorsque de nouveaux fichiers journaux sont fournis, si vous voulez effectuer une action rapide lors de la livraison des fichiers journaux. Pour plus d'informations, consultez [Configuration des notifications Amazon SNS pour CloudTrail](#).

Vous pouvez également regrouper des fichiers journaux AWS Lambda provenant de plusieurs régions AWS et de plusieurs comptes AWS dans un compartiment S3 unique. Pour plus d'informations, consultez [Utilisation des fichiers journaux CloudTrail](#).

Présentation des entrées des fichiers journaux AWS Lambda

Les fichiers journaux CloudTrail contiennent une ou plusieurs entrées de journal et chaque entrée est composée de plusieurs événements au format JSON. Une entrée de journal représente une demande individuelle à partir d'une source quelconque et comprend des informations sur l'action demandée, sur tous les paramètres, sur la date et l'heure de l'action, etc. Les entrées de journal ne sont garanties dans aucun ordre particulier. Cela signifie qu'il ne s'agit pas d'une arborescence des appels de procédure ordonnée des appels d'API publics.

L'exemple suivant montre les entrées de journal CloudTrail pour les actions `GetFunction` et `DeleteFunction`.

```
    "errorMessage": "User: arn:aws:iam::999999999999:user/myUserName" is
not authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:999999999999:function:other-acct-function",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
    "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
    "eventType": "AwsApiCall",
    "recipientAccountId": "999999999999"
},
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::999999999999:user/myUserName",
        "accountId": "999999999999",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
    },
    "eventTime": "2015-03-18T19:04:42Z",
    "eventSource": "lambda.amazonaws.com",
    "eventName": "DeleteFunction",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "Python-httplib2/0.8 (gzip)",
    "requestParameters": {
        "functionName": "basic-node-task"
    },
    "responseElements": null,
    "requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
    "eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
    "eventType": "AwsApiCall",
    "recipientAccountId": "999999999999"
}
]
}
```

Note

La valeur `eventName` peut inclure des informations de date et de version d'informations, comme "GetFunction20150331", mais il s'agit toujours de la même API publique. Pour plus d'informations, consultez [Services pris en charge par l'historique des événements CloudTrail](#) dans le Guide de l'utilisateur AWS CloudTrail.

Utilisation de CloudTrail pour suivre les appels de fonctions

CloudTrail enregistre également les événements de données. Vous pouvez activer la journalisation des événements de données afin de pouvoir enregistrer un événement à chaque appel des fonctions Lambda. Vous pourrez ainsi comprendre quelles identités appellent les fonctions, ainsi que la fréquence de ces appels. Pour l'activer, utilisez la console AWS CloudTrail ou l'opération d'interface de ligne de commande [Invoke \(p. 452\)](#). Pour en savoir plus sur cette option, consultez [Configuration des événements de données et de gestion pour des suivis](#).

Balisage des fonctions Lambda

Les fonctions Lambda peuvent couvrir plusieurs applications sur différentes régions. Pour simplifier le processus de suivi de la fréquence et du coût de chaque fonction d'appel, vous pouvez utiliser des balises.

Les balises sont des paires clé-valeur que vous associez aux ressources AWS afin de mieux les organiser. Elles sont particulièrement utiles lorsque vous avez de nombreuses ressources du même type, une fonction dans le cas d'AWS Lambda. Grâce aux balises, les clients ayant des centaines de fonctions Lambda peuvent facilement accéder à un ensemble spécifique, et l'analyser, en filtrant sur ceux qui contiennent la même balise. Le balisage des fonctions Lambda présente les deux avantages suivants :

- Regroupement et filtrage : En appliquant des balises, vous pouvez utiliser la console Lambda ou l'interface de ligne de commande pour isoler une liste des fonctions Lambda contenues dans une application spécifique ou un service de facturation. Pour plus d'informations, consultez [Filtrage sur les fonctions Lambda balisées \(p. 278\)](#).
- Répartition des coûts : Dans la mesure où la prise en charge du balisage par Lambda est intégrée à AWS Billing, vous pouvez diviser les factures en catégories dynamiques et mapper les fonctions à certains centres de coûts. Par exemple, si vous balisez toutes les fonctions Lambda avec une clé « Service », tous les coûts AWS Lambda peuvent être ventilés par service. Vous pouvez ensuite fournir une valeur de service, par exemple « Service 1 » ou « Service 2 » pour diriger la fonction d'appel de coût vers le centre de coûts approprié. La répartition des coûts apparaît dans des rapports de facturation détaillés, ce qui vous permet de catégoriser et de suivre plus facilement vos coûts AWS.

Rubriques

- [Balisage des fonctions Lambda pour la facturation \(p. 276\)](#)
- [Application des balises aux fonctions Lambda \(p. 276\)](#)
- [Filtrage sur les fonctions Lambda balisées \(p. 278\)](#)
- [Restrictions liées aux balises \(p. 279\)](#)

Balisage des fonctions Lambda pour la facturation

Vous pouvez utiliser des balises pour organiser votre facture AWS afin de refléter votre propre structure de coût. Pour ce faire, vous pouvez ajouter des clés de balise dont les valeurs seront incluses dans le rapport de répartition des coûts. Pour plus d'informations sur la configuration d'un rapport de répartition des coûts qui inclut les clés de balise à faire figurer en tant que lignes d'articles dans le rapport, consultez le [rapport mensuel de répartition des coûts](#) dans À propos de la facturation des comptes AWS.

Pour voir le coût de vos ressources combinées, vous pouvez organiser vos informations de facturation selon les fonctions possédant les mêmes valeurs de clé de balise. Par exemple, vous pouvez baliser plusieurs fonctions Lambda avec un nom d'application spécifique, puis organiser vos informations de facturation pour afficher le coût total de cette application dans plusieurs services. Pour plus d'informations, consultez [Utilisation des balises de répartition des coûts](#) dans le Guide de l'utilisateur AWS Billing and Cost Management.

Important

Dans AWS Lambda, la seule ressource qui peuvent être balisée est une fonction. Vous ne pouvez pas baliser un alias ou une version de fonction spécifique. Tout appel de l'alias ou de la version d'une fonction sera facturé comme un appel de la fonction d'origine.

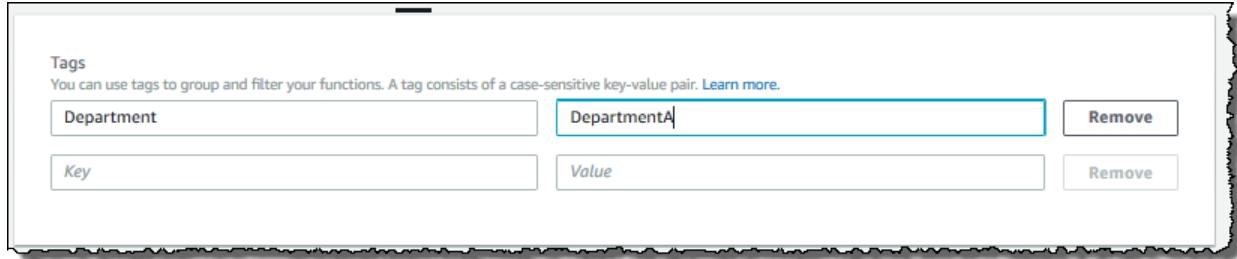
Application des balises aux fonctions Lambda

Le balisage des fonctions Lambda dépend de la façon dont vous créez la fonction. Vous pouvez les appliquer à l'aide de l'interface de ligne de commande ou de la console Lambda, comme expliqué dans les sections suivantes :

- [Application des balises aux fonctions Lambda à l'aide de la console \(p. 277\)](#)
- [Application des balises aux fonctions Lambda à l'aide de l'interface de ligne de commande \(p. 277\)](#)

Application des balises aux fonctions Lambda à l'aide de la console

Vous pouvez ajouter des balises à votre fonction dans la section Tags de l'onglet de configuration.



Pour supprimer des balises d'une fonction existante, ouvrez la fonction, choisissez la section Tags, puis le bouton Remove en regard de la paire clé-valeur.



Application des balises aux fonctions Lambda à l'aide de l'interface de ligne de commande

Lorsque vous créez une nouvelle fonction Lambda à l'aide de la commande [CreateFunction \(p. 410\)](#), vous pouvez ajouter des balises en remplissant le paramètre Tags. Spécifiez plusieurs valeurs de balise en les plaçant entre guillemets, comme indiqué ci-dessous :

Note

Si vous n'avez pas déjà créé le profil adminuser, consultez [Configuration de l'AWS CLI \(p. 12\)](#).

```
$ aws lambda create-function \
--region region \
--function-name function-name
--role role-arn \
--handler handler-name \
--runtime runtime-value \
--runtime runtime \
--tags "DEPARTMENT=Department A, Department B" \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

Pour appliquer ou ajouter des balises à une fonction existante, vous pouvez utiliser l'API [TagResource \(p. 496\)](#) et l'approvisionner avec l'ARN (Amazon Resource Name) de la fonction Lambda, ainsi que les paires clé-valeur comprenant vos balises.

```
$ aws lambda tag-resource \
--resource function arn \
--tags DEPARTMENT="Department C, Department D"
```

À l'inverse, si vous souhaitez supprimer tout ou une partie des balises à partir d'une fonction Lambda, vous utilisez l'API [UntagResource \(p. 498\)](#) et vous fournissez à nouveau l'ARN (Amazon Resource Name) de la fonction, ainsi qu'une liste des clés de balise à supprimer de la fonction.

```
$ aws lambda untag-resource \
--resource function arn \
--tagkeys list of tag keys to be removed
```

Filtrage sur les fonctions Lambda balisées

Une fois que vous avez regroupé vos fonctions Lambda en utilisant des balises, vous pouvez exploiter les fonctionnalités de filtrage fournies par la console ou l'AWS CLI Lambda pour les afficher en fonction de vos besoins spécifiques.

Filtrage des fonctions Lambda à l'aide de la console

La console Lambda contient un champ de recherche qui vous permet de filtrer la liste des fonctions à partir d'un ensemble d'attributs de fonctions, y compris Balises. Supposons que vous ayez deux fonctions nommées MyFunction et MyFunction2 et ayant une clé Tags appelée Department. Pour afficher ces fonctions, choisissez le champ de recherche et observez le filtrage automatique qui inclut une liste de clés Tags :

The screenshot shows the AWS Lambda 'Functions' list page with 25 results. A search bar at the top contains the query 'tag:Department'. To the left of the search bar is a sidebar with filtering options: 'Function attributes' (Description, Function name, Runtime, Tags, Department), 'Runtime' (Node.js 6.10, Python 2.7), and 'Code size' (333 bytes, 360 bytes). The main table lists three functions:

Function name	Runtime	Code size
mbda function.	Node.js 6.10	333 bytes
mbda function.	Node.js 6.10	333 bytes
mbda function.	Python 2.7	360 bytes

Choisissez la clé Service. Lambda renverra une fonction qui contient cette clé.

Supposons maintenant que la valeur de la clé de la balise MyFunction soit « Department A » et la valeur de clé MyFunction2 soit « Department B ». Vous pouvez affiner votre recherche en choisissant la valeur de la clé Department, dans ce cas Department A, comme indiqué ci-dessous.

The screenshot shows the AWS Lambda 'Functions' list page with 25 results. The search bar now displays 'tag:Department:DepartmentA'. The sidebar remains the same. The main table lists two functions:

Function name	Runtime	Code size
mbda function.	Node.js 6.10	333 bytes
mbda function.	Node.js 6.10	333 bytes

Cela renverra uniquement MyFunction.

Vous pouvez affiner votre recherche en incluant les autres attributs Function attributes acceptés, notamment Description, Function name ou Runtime.

Note

Vous êtes limité à un maximum de 50 balises par fonction Lambda. Si vous supprimez la fonction Lambda, les balises associées seront également supprimées.

Filtrage des fonctions Lambda à l'aide de l'interface de ligne de commande

Si vous souhaitez afficher les balises qui sont appliquées à une fonction Lambda spécifique, vous pouvez utiliser l'une des commandes d'API Lambda suivantes :

- [ListTags \(p. 473\)](#) : vous fournissez l'ARN (Amazon Resource Name) de la fonction Lambda pour afficher la liste des balises associées à cette fonction :

```
$ aws lambda list-tags \
--resource function arn \
--region region \
--profile adminuser
```

- [GetFunction \(p. 437\)](#) : vous fournissez le nom de votre fonction Lambda pour afficher la liste des balises associées à cette fonction :

```
$ aws lambda get-function \
--function-name function name \
--region region \
--profile adminuser
```

Vous pouvez également utiliser l'API [GetResources](#) du service de balisage AWS pour filtrer vos ressources par balise. L'API GetResources reçoit jusqu'à 10 filtres, chaque filtre contenant une clé de balise et jusqu'à 10 valeurs de balise. Vous fournissez GetResources avec un ResourceType pour filtrer par certains types de ressources. Pour plus d'informations sur le service de balisage AWS, consultez [Utilisation des groupes de ressources](#).

Restrictions liées aux balises

Les restrictions suivantes s'appliquent aux balises :

- Nombre maximal de balises par ressource : 50
- Longueur de clé maximale : 128 caractères Unicode en UTF-8
- Longueur de valeur maximale : 256 caractères Unicode en UTF-8
- Les clés et valeurs de balise sont sensibles à la casse.
- N'utilisez pas le préfixe `aws` : dans les noms ou valeurs de vos balises car celui-ci est réservé pour être utilisé par AWS. Vous ne pouvez pas modifier ou supprimer des noms ou valeurs de balise ayant ce préfixe. Les balises avec ce préfixe ne sont pas comptabilisées comme vos balises pour la limite de ressources.
- Si votre schéma de balisage doit être utilisé pour plusieurs services et ressources, n'oubliez pas que d'autres services peuvent avoir des restrictions concernant les caractères autorisés. Les caractères généralement autorisés sont les lettres, les espaces et les chiffres représentables en UTF-8, ainsi que les caractères spéciaux suivants : `+ - = . _ : / @`.

Gestion de la simultanéité

L'unité de dimensionnement pour AWS Lambda est une exécution simultanée (consultez [Présentation du comportement de dimensionnement \(p. 79\)](#) pour plus de détails). Cependant, le dimensionnement à l'infini n'est pas souhaitable dans tous les cas. Vous pouvez, par exemple, vouloir contrôler la simultanéité pour des raisons de coûts ou afin de réguler le temps nécessaire au traitement d'un lot d'événements, ou

simplement pour la faire correspondre à une ressource en aval. Dans cette optique, Lambda permet de contrôler la limite des exécutions simultanées aux niveaux du compte et de la fonction.

Limite des exécutions simultanées au niveau du compte

Par défaut, AWS Lambda limite à 1000 le nombre total d'exécutions simultanées de toutes les fonctions au sein d'une région donnée. Vous pouvez consulter les paramètres au niveau du compte à l'aide de l'API [GetAccountSettings \(p. 429\)](#) et de l'affichage de l'objet `AccountLimit`. Cette limite peut être augmentée comme décrit ci-dessous :

Pour demander une augmentation de limite pour les exécutions simultanées

1. Ouvrez la page [AWS Support Center](#), connectez-vous (si nécessaire), puis choisissez Create case (Créer une demande de support).
2. Pour Regarding, sélectionnez Service Limit Increase.
3. Pour Limit Type (Type de limite), choisissez Lambda, remplissez les champs nécessaires du formulaire, puis choisissez le bouton correspondant à votre mode de contact préféré en bas de la page.

Limite des exécutions simultanées au niveau de la fonction

Par défaut, la limite des exécutions simultanées est appliquée par rapport à la somme des exécutions simultanées de toutes les fonctions. Le groupe d'exécutions simultanées partagé est désigné sous le terme d'allocation de simultanéité non réservée. Si vous n'avez pas défini de limite de simultanéité au niveau de la fonction, la limite de simultanéité non réservée est identique à celle définie au niveau du compte. À toute augmentation de la limite au niveau du compte correspond une augmentation de la limite de simultanéité non réservée. Vous pouvez afficher l'allocation de simultanéité non réservée pour une fonction à l'aide de l'API [GetAccountSettings \(p. 429\)](#) ou en utilisant la console AWS Lambda. Les fonctions prises en compte par rapport au groupe d'exécutions simultanées partagé ne présentent aucune valeur de simultanéité lorsqu'elles sont interrogées à l'aide de l'API [GetFunctionConfiguration \(p. 440\)](#).

Vous pouvez éventuellement définir la limite des exécutions simultanées pour une fonction, pour plusieurs raisons :

- Le comportement par défaut implique qu'une hausse des exécutions simultanées dans une fonction empêche la fonction que vous avez isolée avec une limite d'exécutions d'être limitée. En paramétrant une limite des exécutions simultanées au niveau d'une fonction, vous réservez pour cette fonction la valeur d'exécution simultanée spécifiée.
- Les fonctions se mettent automatiquement à l'échelle en fonction du taux de demandes entrantes, mais il se peut que certaines ressources de votre architecture ne puissent pas le faire. Par exemple, pour les bases de données relationnelles, il existe des limites quant au nombre de connexions simultanées pouvant être gérées. Vous pouvez définir la limite des exécutions simultanées pour une fonction d'après les valeurs prises en charge par ses ressources en aval.
- Si la fonction se connecte aux ressources VPC, vous devez vous assurer que vos sous-réseaux disposent d'une capacité d'adresse adéquate pour prendre en charge les exigences de mise à l'échelle de l'interface réseau Elastic (ENI) de votre fonction. Vous pouvez estimer la capacité approximative de l'interface réseau Elastic avec la formule suivante :

Concurrent executions * (Memory in GB / 3 GB)

Où :

- Concurrent execution – Il s'agit de la simultanéité projetée de votre charge de travail. Utilisez les informations de [Présentation du comportement de dimensionnement \(p. 79\)](#) pour déterminer cette valeur.
- Memory in GB (Mémoire en Go) – Quantité de mémoire que vous avez configurée pour la fonction Lambda.

Vous pouvez définir la limite des exécutions simultanées pour une fonction d'après les limites de la taille du sous-réseau.

Note

Pour qu'une fonction arrête de traiter les appels, vous pouvez définir à 0 la simultanéité et restreindre toutes les exécutions entrantes.

En définissant une limite de simultanéité au niveau d'une fonction, Lambda garantit que cette allocation s'appliquera spécifiquement à ladite fonction, quelle que soit l'importance du trafic pour le traitement des fonctions restantes. Si cette limite est dépassée, la fonction sera restreinte. Le comportement de cette fonction, lorsqu'elle est restreinte, dépend de la source d'événements. Pour plus d'informations, consultez [Comportement de limitation \(p. 282\)](#).

Note

Les limites de simultanéité peuvent être définies uniquement au niveau de la fonction, et non pour des versions individuelles. Tous les appels à toutes les versions et à tous les alias d'une fonction donnée sont comptabilisés dans la limite de fonction.

Limites de simultanéité réservée ou non réservée

Si vous définissez la limite des exécutions simultanées pour une fonction, cette valeur est déduite du groupe de simultanéité non réservée. Si, par exemple, votre compte est limité à 1 000 exécutions simultanées et que vous avez 10 fonctions, vous pouvez définir la limite d'une fonction à 200 et celle d'une autre à 100. Les 700 autres exécutions seront partagées entre les 8 dernières fonctions.

Note

AWS Lambda maintiendra à 100 exécutions simultanées minimum le groupe de simultanéité non réservée, de sorte que les fonctions qui n'ont pas de limites spécifiques puissent continuer à traiter les demandes. Ainsi, concrètement, si votre compte est limité à 1 000 exécutions, vous pouvez allouer seulement 900 exécutions à des fonctions individuelles.

Définition de limites de simultanéité par fonction (console)

Pour définir une limite de simultanéité pour la fonction Lambda à l'aide de la console Lambda, procédez comme suit :

1. Connectez-vous à AWS Management Console et ouvrez la console AWS Lambda.
2. Le paramétrage d'une limite de simultanéité s'effectue de la même façon, que vous créeiez une nouvelle fonction Lambda ou mettiez à jour une fonction existante. Si vous n'êtes pas encore familiarisé avec Lambda ni avec la création d'une fonction, consultez [Créer une fonction Lambda avec la console \(p. 6\)](#).
3. Sous l'onglet Configuration, choisissez Concurrency. Sous Reserve concurrency, définissez le nombre maximum d'exécutions simultanées à réserver pour la fonction. Remarque : Lorsque vous définissez cette valeur, la valeur Unreserved account concurrency est automatiquement mise à jour afin d'afficher

le nombre restant d'exécutions simultanées disponibles pour toutes les autres fonctions du compte. Si, par ailleurs, vous souhaitez bloquer l'appel de cette fonction, définissez la valeur sur 0. Pour supprimer une valeur d'allocation dédiée pour ce compte, sélectionnez Use unreserved account concurrency.

Définition de limites de simultanéité par fonction (interface de ligne de commande)

Pour définir une limite de simultanéité pour la fonction Lambda à l'aide de l'AWS CLI, procédez comme suit :

- Utilisez l'opération [PutFunctionConcurrency \(p. 488\)](#) et transmettez le nom de la fonction et la limite de simultanéité à allouer à cette fonction :

```
aws lambda put-function-concurrency --function-name function-name
--reserved-concurrent-executions limit value
```

Pour supprimer une limite de simultanéité pour la fonction Lambda à l'aide de l'AWS CLI, procédez comme suit :

- Utilisez l'opération [DeleteFunctionConcurrency \(p. 425\)](#) et transmettez le nom de la fonction :

```
aws lambda delete-function-concurrency --function-name function-name
```

Pour afficher une limite de simultanéité pour la fonction Lambda à l'aide de l'AWS CLI, procédez comme suit :

- Utilisez l'opération [GetFunction \(p. 437\)](#) et transmettez le nom de la fonction :

```
aws lambda get-function --function-name function-name
```

Note

Le fait de définir la simultanéité par fonction peut avoir des conséquences sur le groupe de simultanéité disponible pour d'autres fonctions. Il est recommandé de restreindre les autorisations d'accès aux API [PutFunctionConcurrency \(p. 488\)](#) et [DeleteFunctionConcurrency \(p. 425\)](#) aux seuls administrateurs, afin de limiter le nombre d'utilisateurs pouvant apporter ces modifications.

Comportement de limitation

Lorsque la limite de simultanéité associée à une fonction est atteinte, toute nouvelle demande d'appel à cette fonction est restreinte, c'est-à-dire que l'appel n'exécute pas la fonction. Chaque appel restreint entraîne une augmentation de la métrique Throttles dans Amazon CloudWatch pour cette fonction. AWS Lambda gère différemment les demandes d'appel restreintes, en fonction de leur source :

- Sources d'événements non basées sur les flux : certaines de ces sources d'événements appellent une fonction Lambda de manière synchrone, d'autres de façon asynchrone. Le traitement diffère selon la méthode employée :
 - Appel synchrone : si la fonction est appelée de façon synchrone et qu'elle est restreinte, Lambda renvoie une erreur 429 et le service à l'origine de l'appel est chargé de réessayer. Le code d'erreur ThrottledReason indique s'il s'agit d'une restriction au niveau de la fonction (si spécifiée) ou au niveau du compte (voir la remarque ci-dessous). À chaque service peut correspondre sa propre

stratégie de nouvelle tentative. Pour obtenir une liste des sources d'événements et leur type d'appel, consultez [Sources d'événements prises en charge \(p. 82\)](#).

Note

Si vous appelez la fonction directement avec les kits SDK AWS via le mode d'appel `RequestResponse`, le client reçoit l'erreur 429 et vous pouvez réessayer l'appel.

- Appel asynchrone : si la fonction Lambda est appelée de façon asynchrone et qu'elle est restreinte, AWS Lambda réessaie automatiquement de lancer l'événement pendant une durée pouvant aller jusqu'à six heures, avec des temps d'attente entre les tentatives. Par exemple, CloudWatch Logs réessaie jusqu'à cinq fois avec un temps d'attente entre les tentatives. Les événements asynchrones sont ajoutés à une file d'attente avant d'être utilisés pour appeler la fonction Lambda. Vous pouvez configurer une file d'attente de lettres mortes pour enquêter sur les raisons pour lesquelles votre fonction a été limitée. Pour plus d'informations, consultez [Files d'attente Lettre Morte \(p. 81\)](#).
- Sources d'événements basées sur les interrogations qui sont aussi basées sur les flux : par exemple [Amazon Kinesis](#), [Amazon DynamoDB](#), AWS Lambda interroge votre flux et appelle votre fonction Lambda. Si votre fonction Lambda est restreinte, Lambda tente de traiter le lot d'enregistrements correspondant tant que les données n'expirent pas. Cela peut aller jusqu'à sept jours pour Amazon Kinesis. La demande restreinte est considérée comme un blocage de partition. Dès lors, Lambda cesse de lire les enregistrements de cette partition jusqu'à ce que le lot d'enregistrements restreint aboutisse ou expire. Si le flux comporte plusieurs partitions, Lambda continue les appels sur les partitions non restreintes jusqu'à ce que l'un d'eux fonctionne.
- Sources d'événements basées sur les interrogations qui ne sont pas basées sur les flux : par exemple [Amazon Simple Queue Service](#), AWS Lambda interroge votre file d'attente et appelle votre fonction Lambda. Lorsque votre fonction Lambda est limitée, Lambda essaie de traiter le lot d'enregistrement limité jusqu'à être invoqué avec succès (auquel cas le message est automatiquement supprimé de la file d'attente) ou jusqu'à ce que la valeur `MessageRetentionPeriod` (durée de rétention du message) définie expire.

Surveillance de votre utilisation de la simultanéité

AWS Lambda fournit les métriques suivantes, qui vous aideront à comprendre l'usage que vous faites des exécutions simultanées :

- `ConcurrentExecutions` : cette métrique présente les exécutions simultanées au niveau d'un compte, pour n'importe quelle fonction avec une limite de simultanéité personnalisée.
- `UnreservedConcurrentExecutions` : cette métrique présente le nombre total d'exécutions simultanées pour les fonctions attribuées au groupe de simultanéité non réservée par défaut.

Pour en savoir plus sur ces métriques et sur la façon d'y accéder, consultez [Utiliser Amazon CloudWatch \(p. 223\)](#).

Bonnes pratiques d'utilisation des fonctions AWS Lambda

Les bonnes pratiques suivantes sont recommandées pour l'utilisation d'AWS Lambda :

Rubriques

- [Code de fonction \(p. 284\)](#)
- [Configuration de fonctions \(p. 285\)](#)

- [Alarmes et métriques \(p. 285\)](#)
- [Appels d'événements de flux \(p. 286\)](#)
- [Appels asynchrones \(p. 286\)](#)
- [VPC Lambda \(p. 286\)](#)

Code de fonction

- Séparez le gestionnaire Lambda (point d'entrée) de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire. Dans Node.js, vous obtenez ce qui suit :

```
exports.myHandler = function(event, context, callback) {
  var foo = event.foo;
  var bar = event.bar;
  var result = MyLambdaFunction (foo, bar);

  callback(null, result);
}

function MyLambdaFunction (foo, bar) {
  // MyLambdaFunction logic here
}
```

- Tirez parti de la réutilisation du contexte d'exécution pour améliorer les performances de votre fonction. Assurez-vous que les configurations externalisées et les dépendances que votre code extrait sont stockées et référencées localement après l'exécution initiale. Limitez la réinitialisation des variables/objets sur chaque appel. Utilisez à la place l'initialisation statique/le constructeur statique, les variables globales/statiques et les singletons. Maintenez et réutilisez les connexions (HTTP, base de données, etc.) qui ont été établies lors d'un précédent appel.
- Utilisez [Variables d'environnement AWS Lambda \(p. 30\)](#) pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3, au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Contrôlez les dépendances de votre package de déploiement des fonctions. L'environnement d'exécution AWS Lambda contient un certain nombre de bibliothèques telles que le kit AWS SDK pour les runtimes Node.js et Python (la liste complète est disponible à l'adresse [Environnement d'exécution Lambda et bibliothèques disponibles \(p. 104\)](#)). Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour que vous ayez le contrôle total des dépendances que votre fonction utilise, il est recommandé d'empaqueter toutes vos dépendances avec votre package de déploiement.
- Réduisez la taille de votre package de déploiement à ses strictes nécessités de runtime. Vous réduirez ainsi le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'appel. Pour les fonctions créées dans Java ou .NET Core, évitez de charger la totalité de la bibliothèque du SDK AWS comme partie intégrante de votre package de déploiement. À la place, appuyez-vous de façon sélective sur les modules qui sélectionnent les composants du kit SDK dont vous avez besoin (par exemple, les modules SDK Amazon S3, DynamoDB et les [bibliothèques Lambda principales](#)).
- Réduisez le temps mis par Lambda pour décompresser les packages de déploiement créés dans Java en plaçant vos fichiers de dépendance .jar dans un répertoire /lib distinct. Cette solution est plus rapide que le placement de tout le code de votre fonction dans un seul fichier .jar comprenant une multitude de fichiers .class.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage du [contexte d'exécution](#). Par exemple, préférez les frameworks d'injection (IoC) de dépendances Java plus simples comme [Dagger](#) ou [Guice](#), à des plus complexes comme [Spring Framework](#).

- Évitez d'utiliser du code récursif dans votre fonction Lambda, dans lequel la fonction s'appelle elle-même automatiquement jusqu'à ce que certains critères arbitraires soient satisfaits. Cela peut entraîner un volume involontaire d'appels de fonction et des coûts accrus. Si vous le faites par inadvertance, définissez immédiatement la limite des exécutions simultanées de la fonction sur 0 afin de bloquer tous les appels à la fonction pendant que vous mettez à jour le code.

Configuration de fonctions

- Le test de performance de votre fonction Lambda est une partie cruciale pour garantir que vous choisissez la configuration de taille mémoire optimale. Toute augmentation de la taille mémoire déclenche une augmentation équivalente de l'UC disponible pour votre fonction. L'utilisation de la mémoire pour votre fonction est déterminée par appel et peut s'afficher dans les [joumnaux AWS CloudWatch](#). À chaque appel une entrée REPORT: est créée, comme indiqué ci-dessous :

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

En analysant le champ `Max Memory Used:`, vous pouvez déterminer si votre fonction a besoin de plus de mémoire ou si vous avez surdimensionné la taille mémoire de votre fonction.

- Effectuez un test de charge de votre fonction Lambda pour déterminer une valeur de délai d'expiration. Il importe d'analyser comment votre fonction s'exécute afin que vous puissiez mieux déterminer les problèmes de service de dépendance qui peuvent accroître la simultanéité de la fonction au-delà de ce que vous attendez. Cet aspect est particulièrement important quand votre fonction Lambda effectue des appels réseau aux ressources qui peuvent ne pas gérer la mise à l'échelle de Lambda.
- Utilisez les autorisations les plus restrictives lors de la définition des stratégies IAM. Maîtrisez les ressources et les opérations dont votre fonction Lambda a besoin et limitez le rôle d'exécution à ces autorisations. Pour plus d'informations, consultez [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).
- Familiarisez-vous avec [Limites AWS Lambda \(p. 288\)](#). La taille de la charge utile, les descripteurs de fichiers et l'espace /tmp sont souvent ignorés lors de la détermination des limites des ressources.
- Supprimez les fonctions Lambda que vous n'utilisez plus. En procédant ainsi, les fonctions inutilisées n'interviendront plus inutilement dans la limite de taille de votre package de déploiement.
- Si vous utilisez Amazon Simple Queue Service en tant que source d'événement, assurez-vous que la valeur de la durée d'exécution prévue de la fonction ne dépasse pas la valeur [Délai de visibilité](#) de la file d'attente. Cela s'applique à [CreateFunction \(p. 410\)](#) et [UpdateFunctionConfiguration \(p. 514\)](#).
 - Dans le cas de CreateFunction, AWS Lambda échouera lors de l'exécution du processus de création de la fonction.
 - Dans le cas de UpdateFunctionConfiguration, cela peut entraîner des appels en double de la fonction.

Alarmes et métriques

- Utilisez les [Indicateurs AWS Lambda \(p. 228\)](#) et les [alarmes CloudWatch](#) au lieu de créer ou de mettre à jour une métrique depuis le code de votre fonction Lambda. Le suivi de l'état de vos fonctions Lambda est une solution plus efficace et vous permet d'intercepter de façon précoce les problèmes du processus de développement. Par exemple, vous pouvez configurer une alarme basée sur la durée prévue de l'exécution de votre fonction Lambda afin de pouvoir prendre en compte les goulets d'étranglement ou les latences du code de votre fonction.
- Mettez à profit votre bibliothèque de journalisation et les [dimensions et métriques AWS Lambda](#) pour intercepter les erreurs d'application (par exemple, ERR, ERROR, WARNING, etc.)

Appels d'événements de flux

- Testez différentes tailles de lot et d'enregistrement de telle sorte que la fréquence d'interrogation de chaque source d'événement soit réglée sur la vitesse à laquelle votre fonction peut exécuter sa tâche. **BatchSize** contrôle le nombre maximal d'enregistrements qui peuvent être envoyés à votre fonction avec chaque appel. Une taille de lot plus grande peut souvent absorber plus efficacement l'appel sur un plus large ensemble d'enregistrements, ce qui accroît votre débit.

Note

Quand il n'y a plus assez d'enregistrements à traiter, au lieu d'attendre, la fonction de traitement de flux est appelé avec un plus petit nombre d'enregistrements.

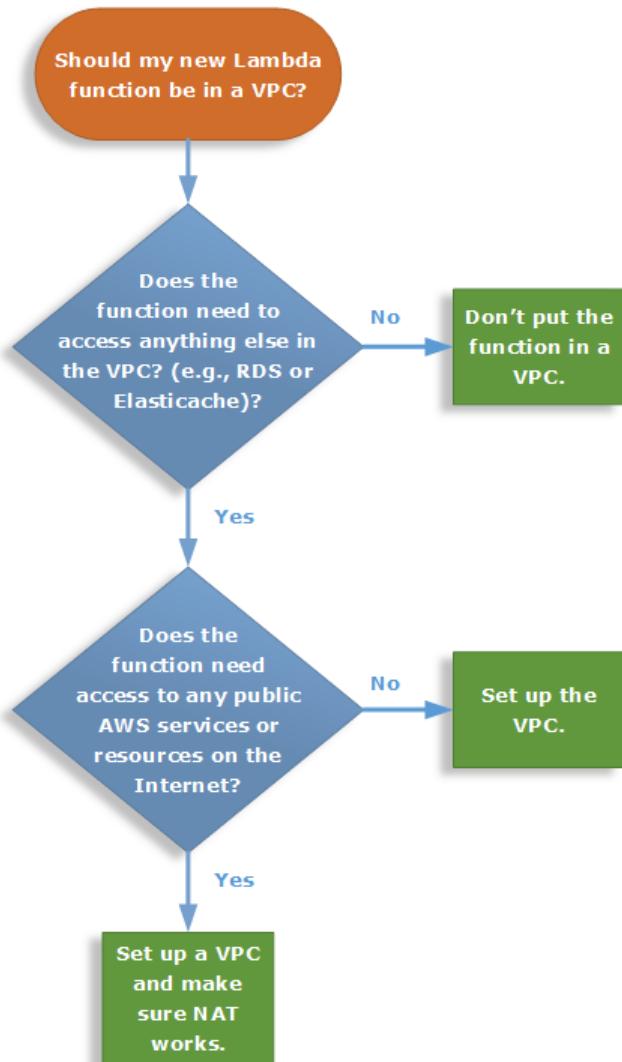
- Augmentez le débit du traitement de flux Kinesis en ajoutant des partitions. Un flux Kinesis est composé d'une ou plusieurs partitions. Lambda interroge chaque partition avec au plus un appel simultané. Par exemple, si le flux contient 100 partitions actives, il y aura au plus 100 appels de fonctions Lambda s'exécutant simultanément. L'augmentation du nombre de partitions entraîne directement celle du nombre d'appels maximaux simultanés de fonctions Lambda et peut accroître le débit de traitement des flux Kinesis. Si vous augmentez le nombre de partitions d'un flux Kinesis, assurez-vous d'avoir choisi une bonne clé de partition (consultez [Clés de partition](#)) pour vos données, de telle sorte que les enregistrements associés se retrouvent sur les mêmes partitions et que vos données soient bien distribuées.
- Utilisez [Amazon CloudWatch](#) sur IteratorAge pour déterminer si votre flux Kinesis est en cours de traitement. Par exemple, configurez une alarme CloudWatch avec une valeur maximale de 30000 (30 secondes).

Appels asynchrones

- Créez et utilisez [Files d'attente Lettre Morte \(p. 81\)](#) pour traiter et relire les erreurs de fonctions asynchrones.

VPC Lambda

- Le schéma suivant vous guide à travers un arbre de décision comme si vous deviez utiliser un VPC (Virtual Private Cloud) :



- Ne placez pas votre fonction Lambda dans un VPC à moins d'y être obligé. Il n'y a aucun avantage à cela, si ce n'est de l'utiliser pour accéder aux ressources que vous ne pouvez pas exposer publiquement, comme une instance [Amazon Relational Database](#) privée. Les services comme Amazon Elasticsearch Service peuvent être sécurisés sur IAM avec les stratégies d'accès ; par conséquent, l'exposition publique du point de terminaison est sécurisée et ne nécessite pas que vous exécutez votre fonction dans le VPC pour le sécuriser.
- Lambda crée des interfaces réseau Elastic ([ENI](#)) dans votre VPC pour accéder à vos ressources internes. Avant de demander une augmentation simultanée, assurez-vous d'avoir une capacité ENI suffisante (la formule correspondante est disponible à l'adresse [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#)) et un espace d'adressage IP. Si vous n'avez pas une capacité ENI suffisante, vous devrez demander une augmentation. Si vous n'avez pas un espace d'adressage IP suffisant, vous pouvez demander à créer un plus vaste sous-réseau.
- Créez des sous-réseaux Lambda dédiés dans votre VPC :
 - Il vous sera ainsi plus facile d'appliquer une table de routage personnalisée pour le trafic NAT Gateway sans avoir à modifier vos autres sous-réseaux privés/publics. Pour plus d'informations, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#)
- Vous pouvez ainsi dédier un espace d'adressage à Lambda sans le partager avec d'autres ressources.

Limites AWS Lambda

AWS Lambda limite la quantité de ressources de calcul et de stockage que vous pouvez utiliser pour exécuter et stocker des fonctions. Les limites suivantes s'appliquent par région et peuvent être augmentées. Pour demander une augmentation, utilisez la [console Support Center](#).

Ressource	Limite par défaut
Exécutions simultanées (p. 279)	1000
Stockage de couche et fonction	75 GB

Pour plus d'informations sur la façon dont Lambda met à l'échelle votre simultanéité de fonction en réponse au trafic, consultez [Présentation du comportement de dimensionnement \(p. 79\)](#).

Les limites suivantes s'appliquent à la configuration, aux déploiements et à l'exécution des fonctions. Elles ne peuvent pas être modifiées.

Ressource	Limite
Allocation de mémoire (p. 29) des fonctions	De 128 Mo à 3008 MB, par incrément de 64 Mo
Délai d'expiration (p. 29) des fonctions	900 seconds (15 minutes)
Variables d'environnement (p. 30) des fonctions	4 KB
Couches de fonctions (p. 56)	5 layers
Charge utile d'appel (p. 70) (demande et réponse)	6 MB (synchrone) 256 KB (asynchrone)
Taille du package de déploiement (p. 25)	50 MB (compressé) 250 MB (décompressé, y compris les couches) 3 MB (éditeur de console)
Événements de test (éditeur de console)	10
Stockage dans le répertoire /tmp	512 MB
Descripteurs de fichier	1024
Processus/threads d'exécution	1024

Les limites pour les autres services, tels que AWS Identity and Access Management, Amazon CloudFront (Lambda@Edge) et Amazon Virtual Private Cloud, peuvent avoir un impact sur vos fonctions Lambda. Pour plus d'informations, consultez [Limites de service AWS](#).

Création de fonctions Lambda avec Node.js

AWS Lambda prend en charge les environnements d'exécution Node.js suivants :

Runtimes Node.js

Nom	Identificateur
Node.js 8.10	nodejs8.10
Node.js 6.10	nodejs6.10

Lorsque vous créez une fonction Lambda, vous spécifiez l'environnement d'exécution que vous souhaitez utiliser. Pour plus d'informations, consultez le paramètre `runtime` de [CreateFunction \(p. 410\)](#).

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lors de la création du code d'une fonction Lambda en Node.js. Sauf mention contraire, le modèle de programmation décrit dans les sections suivantes s'applique à toutes les versions d'environnement d'exécution prises en charge.

Rubriques

- [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Node.js \(p. 291\)](#)
- [Objet de contexte AWS Lambda dans Node.js \(p. 293\)](#)
- [Journalisation des fonctions AWS Lambda dans Node.js \(p. 294\)](#)
- [Erreurs de fonction AWS Lambda dans Node.js \(p. 296\)](#)
- [Instrumentation du code Node.js dans AWS Lambda \(p. 299\)](#)

Package de déploiement AWS Lambda dans Node.js

Pour créer une fonction Lambda, vous devez d'abord concevoir un package de déploiement de cette fonction, à savoir un fichier .zip qui se compose de votre code et de toutes les dépendances. Comme indiqué précédemment, vous devez définir les autorisations de sécurité appropriées pour le package zip. Pour plus d'informations, consultez les stratégies [Authentification et contrôle d'accès pour AWS Lambda \(p. 238\)](#).

Vous pouvez créer un package de déploiement vous-même ou écrire votre code directement dans la console Lambda. Dans ce cas, la console conçoit le package de déploiement pour vous et l'importe afin de créer la fonction Lambda. Notez les points suivants pour déterminer si vous pouvez utiliser la console pour créer la fonction Lambda :

- Scénario simple : si votre code personnalisé nécessite uniquement la bibliothèque AWS SDK, vous pouvez utiliser l'éditeur intégré dans la console AWS Lambda. Cette dernière vous permet de modifier et d'importer le code dans AWS Lambda. La console compresse le code avec les informations de configuration appropriées dans un package de déploiement au format .zip que le service Lambda peut exécuter.

Vous pouvez également tester le code dans la console en l'appelant manuellement à l'aide d'un échantillon de données d'événement.

Note

Le service Lambda a préinstallé le kit AWS SDK pour Node.js.

- Scénario avancé : si vous écrivez du code qui exploite des autres ressources (par exemple, une bibliothèque pour le traitement d'images) ou que vous voulez utiliser l'interface de ligne de commande AWS au lieu de la console, vous devez d'abord créer le package de déploiement de la fonction Lambda, puis utiliser la console ou l'interface de ligne de commande pour importer ce package.

Note

Une fois que vous avez généré un package de déploiement, vous pouvez soit l'importer directement, soit importer le fichier .zip dans un compartiment Amazon S3 de la région AWS dans laquelle vous voulez créer la fonction Lambda, puis spécifier le nom de compartiment et le nom de la clé d'objet lors de la création de la fonction Lambda via la console ou l'interface de ligne de commande AWS.

La procédure suivante illustre comment créer un package de déploiement (en dehors de la console). Supposons que vous vouliez créer un package de déploiement qui inclut un fichier de code `filename.js` et que le code utilise la bibliothèque `async`.

1. Ouvrez un éditeur de texte et écrivez le code. Enregistrez le fichier (par exemple, `filename.js`).

Vous allez utiliser le nom du fichier pour spécifier le gestionnaire au moment de la création de la fonction Lambda.

2. Dans le même répertoire, utilisez npm pour installer les bibliothèques dont dépend votre code. Par exemple, si votre code utilise la bibliothèque `async`, exécutez la commande npm suivante.

```
npm install async
```

3. Votre répertoire aura ensuite la structure suivante :

```
filename.js
node_modules/async
node_modules/async/lib
node_modules/async/lib/async.js
node_modules/async/package.json
```

4. Compressez le contenu du dossier dans un fichier zip, lequel sera votre package de déploiement (par exemple, `sample.zip`).

Ensuite, spécifiez le nom du fichier .zip de votre package de déploiement lors de la création de la fonction Lambda.

Si vous souhaitez inclure vos propres fichiers binaires, y compris ceux qui sont natifs, ajoutez-les au fichier zip que vous importez, puis référez-les (en incluant le chemin d'accès relatif dans le fichier zip que vous avez créé) lorsque vous les appelez à partir de Node.js ou d'autres processus que vous avez lancés précédemment. Veillez à inclure les éléments suivants au début du code de la fonction : `process.env['PATH'] = process.env['PATH'] + ':' + process.env['LAMBDA_TASK_ROOT']`

Pour plus d'informations sur l'ajout de fichiers binaires natifs dans votre package de fonctions Lambda, consultez [Utilisation des fichiers exécutables dans AWS Lambda](#). Notez également que vous devez fournir

les autorisations nécessaires au contenu du fichier Zip. Pour plus d'informations, consultez [Stratégies d'autorisation sur les packages de déploiement Lambda \(p. 26\)](#).

Gestionnaire de fonctions AWS Lambda dans Node.js

AWS Lambda appelle la fonction Lambda via un objet `handler`. Un élément `handler` représente le nom de la fonction Lambda (et sert de point d'entrée utilisé par AWS Lambda pour exécuter le code de votre fonction). Par exemple :

```
exports.myHandler = function(event, context, callback) {
    ... function code
    callback(null, "some success message");
    // or
    // callback("some error type");
}
```

- **`myHandler`** : il s'agit du nom de la fonction qu'appelle AWS Lambda. Supposons que vous enregistriez ce code sous le nom de fichier `helloworld.js`. Ensuite, `myHandler` est la fonction qui contient le code de votre fonction Lambda et `helloworld` est le nom du fichier qui représente votre package de déploiement. Pour plus d'informations, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 289\)](#).

AWS Lambda prend en charge deux types d'appel :

- RequestResponse ou exécution synchrone : AWS Lambda renvoie le résultat de l'appel de fonction au client appelant la fonction Lambda. Si le code du gestionnaire de la fonction Lambda ne spécifie pas de valeur de retour, AWS Lambda renvoie automatiquement `null` pour cette valeur. Pour voir un exemple simple, consultez [exemple \(p. 292\)](#).
- Événement ou exécution asynchrone : AWS Lambda élimine tous les résultats de l'appel de fonction.

Note

Si vous découvrez que la fonction Lambda ne traite pas l'événement utilisant un appel asynchrone, vous pouvez faire des recherches sur cette défaillance en utilisant [Files d'attente Lettre Morte \(p. 81\)](#).

Les sources d'événements sont d'un service AWS pris en charge à des applications personnalisées qui appellent votre fonction Lambda. Pour obtenir des exemples, consultez [Exemples d'événements publiés en fonction des sources \(p. 91\)](#). Pour voir un exemple simple, consultez [exemple \(p. 292\)](#).

- `context` : AWS Lambda utilise ce paramètre pour fournir des détails sur l'exécution de votre fonction Lambda. Pour plus d'informations, consultez [Objet de contexte AWS Lambda dans Node.js \(p. 293\)](#).
- `callback` (facultatif) – Consultez [Utilisation du paramètre de rappel \(p. 291\)](#).

Utilisation du paramètre de rappel

Les environnements d'exécution Node.js v6.10 et v8.10 prennent en charge le paramètre `callback` facultatif. Vous pouvez l'utiliser explicitement pour renvoyer les informations au mandataire. La syntaxe générale est la suivante :

```
callback(Error error, Object result);
```

Où :

- **error** : paramètre facultatif que vous pouvez utiliser pour fournir les résultats d'une exécution de fonction Lambda ayant échoué. Lorsqu'une fonction Lambda réussit, vous pouvez transmettre la valeur « null » comme premier paramètre.
- **result** : paramètre facultatif que vous pouvez utiliser pour fournir le résultat de l'exécution d'une fonction ayant réussi. Le résultat fourni doit être compatible avec `JSON.stringify`. Si une erreur est indiquée, ce paramètre est ignoré.

Si vous n'utilisez pas `callback` dans le code, AWS Lambda l'appelle implicitement et la valeur de retour indique `null`.

Lorsque le rappel est appelé (explicitement ou implicitement), AWS Lambda poursuit l'appel de la fonction Lambda jusqu'à ce que la boucle d'événements soit vide.

Voici quelques exemples de rappels :

```
callback();      // Indicates success but no information returned to the caller.  
callback(null); // Indicates success but no information returned to the caller.  
callback(null, "success"); // Indicates success with information returned to the caller.  
callback(error); // Indicates error with error information returned to the caller.
```

AWS Lambda traite comme exception gérée toutes les valeurs qui ne sont pas `null` pour le paramètre `error`.

Remarques :

- Quel que soit le type d'appel spécifié pour la fonction Lambda (consultez [Invoke \(p. 452\)](#)), la méthode de rappel consigne automatiquement la représentation de la chaîne des valeurs non null de type `error` dans le flux Amazon CloudWatch Logs associé à la fonction Lambda.
- Si la fonction Lambda a été appelée de façon synchrone (avec le type d'appel `RequestResponse`), le rappel renvoie le corps de la réponse suivant :
 - Si la valeur `error` est `null`, le corps de la réponse affiche la représentation de chaîne `result`.
 - Si la valeur `error` n'est pas `null`, la valeur `error` est indiquée dans le corps de la réponse.

Note

Lorsque le paramètre `callback(error, null)` (et `callback(error)`) est appelé, Lambda consigne les 256 premiers Ko de l'objet d'erreur. Lorsque l'objet d'erreur est plus grand, AWS Lambda tronque le journal et affiche le texte `Truncated by Lambda` à côté de cet objet.

Si vous utilisez la version d'exécution 8.10, vous pouvez inclure le mot-clé `async` :

```
exports.myHandler = async function(event, context) {  
    ...  
    // return information to the caller.  
}
```

exemple

Prenez l'exemple de code Node.js suivant.

```
exports.myHandler = function(event, context, callback) {  
    console.log("value1 = " + event.key1);  
    console.log("value2 = " + event.key2);  
    callback(null, "some success message");  
    // or
```

```
// callback("some error type");
}
```

Cet exemple possède un fonction, *myHandler*

Dans cette fonction, les instructions `console.log()` consignent certaines des données d'événement entrantes dans CloudWatch Logs. Lorsque le paramètre `callback` est appelé, la fonction Lambda se termine uniquement une fois que la boucle d'événements transmise est vide.

Si vous souhaitez utiliser la fonctionnalité `async` fournie par l'environnement d'exécution v8.10, vous pouvez utiliser l'exemple de code suivant :

```
exports.myHandler = async function(event, context) {
  console.log("value1 = " + event.key1);
  console.log("value2 = " + event.key2);
  return "some success message";
  // or
  // throw new Error("some error type");
}
```

Pour importer et tester ce code en tant que fonction Lambda (console)

1. Dans la console, créez une fonction Lambda sur la base des informations suivantes :
 - Utilisez le plan hello-world.
 - L'exemple utilise nodejs6.10 comme runtime mais vous pouvez également sélectionner nodejs8.10. Les exemples de code fournis fonctionnent pour chaque version.
2. Remplacez le code du modèle par le code fourni dans cette section, puis créez la fonction.
3. Testez la fonction Lambda avec l'exemple de modèle d'événement Hello World fourni dans la console Lambda.

Objet de contexte AWS Lambda dans Node.js

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 291\)](#). Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'appel, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `getRemainingTimeInMillis()` – Returns the number of milliseconds left before the execution times out.

Propriétés du contexte

- `functionName` – The name of the Lambda function.
- `functionVersion` – The [version \(p. 38\)](#) of the function.
- `invokedFunctionArn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `memoryLimitInMB` – The amount of memory configured on the function.
- `awsRequestId` – The identifier of the invocation request.

- `logGroupName` – The log group for the function.
- `logStreamName` – The log stream for the function instance.
- `identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
 - `cognitoIdentityId` – The authenticated Amazon Cognito identity.
 - `cognitoIdentityPoolId` – The Amazon Cognito identity pool that authorized the invocation.
- `clientContext` – (mobile apps) Client context provided to the Lambda invoker by the client application.
 - `client.installation_id`
 - `client.app_title`
 - `client.app_version_name`
 - `client.app_version_code`
 - `client.app_package_name`
 - `env.platform_version`
 - `env.platform`
 - `env.make`
 - `env.model`
 - `env.locale`
 - `Custom` – Valeurs personnalisées définies par l'application mobile.
- `callbackWaitsForEmptyEventLoop` – Définissez ce paramètre sur faux pour envoyer la réponse immédiatement lorsque le [rappel \(p. 291\)](#) s'exécute, au lieu d'attendre que la boucle d'événement Node.js soit vide. Si ce paramètre est faux, les événements restants continueront de s'exécuter lors du prochain appel.

L'exemple suivant montre une fonction de gestionnaire qui consigne les informations de contexte.

Example index.js

```
exports.handler = function(event, context, callback) {
    console.log('remaining time =', context.getRemainingTimeInMillis());
    console.log('functionName =', context.functionName);
    console.log('AWSrequestID =', context.awsRequestId);
    callback(null, context,functionName);
};
```

Journalisation des fonctions AWS Lambda dans Node.js

Votre fonction Lambda peut contenir des instructions de journalisation. AWS Lambda écrit ces journaux dans CloudWatch. Si vous utilisez la console Lambda pour appeler la fonction Lambda, la console affiche les mêmes journaux.

Les instructions Node.js suivantes génèrent des entrées de journal :

- `console.log()`
- `console.error()`
- `console.warn()`
- `console.info()`

Prenons les exemples de code Node.js suivants :

- Le premier exemple peut être écrit à l'aide de la version d'exécution 6.10 ou 4.3.

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, event.key1); // Echo back the first key value
};
```

- Le second exemple utilise la fonctionnalité Node.js `async`, disponible uniquement dans les versions d'exécution 8.10 ou ultérieures.

```
console.log('Loading function');

exports.handler = async function(event) {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    return event.key1 // Echo back the first key value
};
```

Dans les deux cas, la capture d'écran suivante illustre un exemple de section Sortie de journal dans la console Lambda. Vous pouvez examiner les mêmes informations dans CloudWatch Logs. Pour plus d'informations, consultez [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 227\)](#).

The screenshot shows the AWS Lambda execution results for a successful function call. It includes sections for the execution summary, log output, and resource metrics.

Execution result: succeeded (logs)

Summary

- Code SHA-256: yhxWtSs9l4Kb+WPTJ+VLnUAoItyzuctEOuevh8ZIlg=
- Request ID: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d
- Duration: 472.47 ms
- Billed duration: 500 ms
- Resources configured: 128 MB

Log output

The log output shows the following entries:

```

START RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d Version: $LATEST
2017-07-20T18:24:06.610Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value1 = value1
2017-07-20T18:24:06.629Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value2 = value2
2017-07-20T18:24:06.629Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value3 = value3
END RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d
REPORT RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d Duration: 472.47 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 18 MB

```

La console utilise le type d'appel `RequestResponse` (appel synchrone) lors de l'appel de la fonction. Par conséquent, AWS Lambda lui renvoie la valeur de retour (`value1`) qu'affiche la console.

Pour tester le code Node.js précédent dans la console AWS Lambda

1. Dans la console, créez une fonction Lambda avec le plan hello-world. Veillez à sélectionner Node.js sous runtime. Pour obtenir des instructions sur la façon de procéder, consultez [Créer une fonction Lambda avec la console \(p. 6\)](#).
2. Testez la fonction Lambda avec l'exemple de modèle d'événement Hello World fourni dans la console Lambda. Vous pouvez également mettre à jour le code et tester les autres méthodes de journalisation et les autres propriétés présentées dans cette section.

Pour obtenir des instructions détaillées, consultez [Mise en route \(p. 3\)](#).

Recherche des journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Dans la console AWS Lambda : la section Sortie de journal de la console AWS Lambda affiche les journaux.
- Dans l'en-tête de la réponse, lorsque vous appelez une fonction Lambda par programmation : si vousappelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les quatre derniers Ko de données de journal écrites dans CloudWatch Logs. AWS Lambda renvoie ces informations de journal dans l'en-tête `x-amz-log-results` de la réponse. Pour plus d'informations, consultez [Invoke](#).

Si vous utilisez AWS CLI pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type parameter` avec la valeur `Tail` pour récupérer les mêmes informations.

- Dans CloudWatch Logs : pour trouver vos journaux dans CloudWatch, vous devez connaître le nom du groupe de journaux, ainsi que le nom de leur flux. Pour obtenir ces informations, ajoutez les méthodes `context.logGroupName` et `context.logStreamName` dans le code. Lorsque vous exécutez la fonction Lambda, les journaux générés dans la console ou l'interface de ligne de commande indiquent le nom du groupe de journaux et le nom de leur flux.

Erreurs de fonction AWS Lambda dans Node.js

Si la fonction Lambda avertit AWS Lambda qu'elle n'a pas pu être exécutée correctement, Lambda tente de convertir l'objet d'erreur en chaîne. Prenez l'exemple suivant :

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    // This example code only throws error.
    var error = new Error("something is wrong");
    callback(error);
};
```

Lorsque vousappelez cette fonction Lambda, celle-ci informe AWS Lambda que son exécution a généré une erreur et transmet les informations correspondantes à AWS Lambda. AWS Lambda renvoie les informations d'erreur au client :

```
{
```

```
"errorMessage": "something is wrong",
"errorType": "Error",
"stackTrace": [
    "exports.handler (/var/task/index.js:10:17)"
]
}
```

Vous obtenez le même résultat si vous écrivez la fonction à l'aide de la fonctionnalité asynchrone de l'environnement d'exécution Node.js version 8.10. Par exemple :

```
exports.handler = async function(event, context) {
    function AccountAlreadyExistsError(message) {
        this.name = "AccountAlreadyExistsError";
        this.message = message;
    }
    AccountAlreadyExistsError.prototype = new Error();

    const error = new AccountAlreadyExistsError("Account is in use!");
    throw error
};
```

De nouveau, lorsque cette fonction Lambda est appelée, celle-ci informe AWS Lambda que son exécution a généré une erreur et transmet les informations correspondantes à AWS Lambda. AWS Lambda renvoie les informations d'erreur au client :

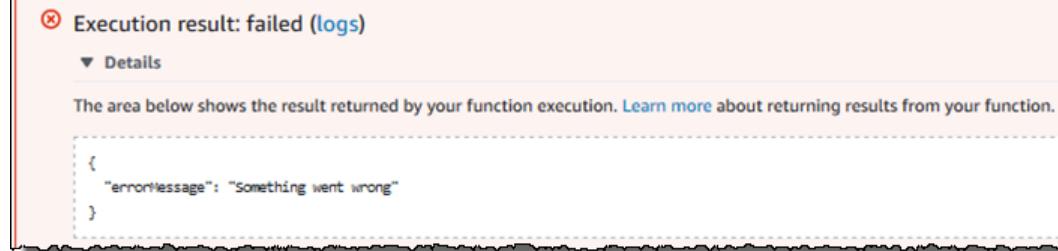
```
{
    "errorMessage": "Acccount is in use!",
    "errorType": "Error",
    "stackTrace": [
        "exports.handler (/var/task/index.js:10:17)"
    ]
}
```

Notez que les informations d'erreur sont renvoyées sous la forme du tableau JSON stackTrace constitué d'éléments de trace de pile.

La façon dont vous recevez les informations d'erreur varie selon le type d'appel que le client spécifie au moment de l'appel d'une fonction :

- Si le client spécifie le type d'appel RequestResponse (c'est-à-dire, l'exécution synchrone), elle renvoie le résultat au client qui est à l'origine de l'appel « invoke ».

Par exemple, la console utilise toujours le type d'appel RequestResponse. Par conséquent, elle affiche l'erreur comme suit dans la section Execution result :



Les mêmes informations sont également envoyées à CloudWatch, et la section Log output affiche les mêmes journaux.

Summary	Log output
Code SHA-256 U4b2T1IAJ6JHw7VXNT W02s3RXctox6Ph3Jw3 7xQfO6g=	The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.
Request ID 093bf2a1-6cba-11e7-b9ad-850729ae85a8	<pre>START RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Version: \$LATEST 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value3 = undefined 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value4 = undefined 2017-07-19T19:39:53.488Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value5 = undefined 2017-07-19T19:39:53.489Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 {"errorMessage": "Something went wrong"} END RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 REPORT RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Duration: 41.24 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 17 MB</pre>
Duration 41.24 ms	
Billed duration 100 ms	

- Si le client spécifie le type d'appel **Event** (c'est-à-dire, l'exécution asynchrone), AWS Lambda ne renvoie rien. Au lieu de cela, il consigne les informations d'erreur dans [CloudWatch Logs](#). Vous pouvez également voir les métriques d'erreur dans [Métriques CloudWatch](#).

En fonction de la source d'événement, AWS Lambda réessaie parfois d'exécuter la fonction Lambda qui a échoué. Par exemple, si Kinesis est la source d'événement, AWS Lambda réessaie d'exécuter l'appel qui a échoué jusqu'à ce que la fonction Lambda aboutisse ou jusqu'à ce que les enregistrements du flux expirent. Pour plus d'informations sur les nouvelles tentatives, consultez la section [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Pour tester le code Node.js précédent (console)

- Dans la console, créez une fonction Lambda avec le plan hello-world. Dans runtime, sélectionnez Node.js et, dans Role, sélectionnez Basic execution role. Pour obtenir des instructions sur la façon de procéder, consultez [Créer une fonction Lambda avec la console \(p. 6\)](#).
- Remplacez le code de modèle par le code fourni dans cette section.
- Testez la fonction Lambda avec l'exemple de modèle d'événement Hello World fourni dans la console Lambda.

Gestion des erreurs de fonction

Vous pouvez créer la gestion des erreurs personnalisées pour déclencher une exception directement depuis la fonction Lambda et la gérer directement (Réessayer ou CATCH) dans le cadre d'une machine d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

Prenons l'exemple d'un état `CreateAccount` qui est une tâche écrivant les détails d'un client dans une base de données à l'aide d'une fonction Lambda.

- Si la tâche réussit, un compte est créé et un e-mail de bienvenue est envoyé.
- Si un utilisateur essaie de créer un compte pour un nom d'utilisateur qui existe déjà, la fonction Lambda génère une erreur, ce qui pousse la machine d'état à suggérer un autre nom d'utilisateur et à recommencer le processus de création de compte.

L'extrait de code suivant montre comment procéder. Notez que les erreurs personnalisées dans Node.js doivent étendre le prototype d'erreur.

```
exports.handler = function(event, context, callback) {
    function AccountAlreadyExistsError(message) {
        this.name = "AccountAlreadyExistsError";
        this.message = message;
    }
    AccountAlreadyExistsError.prototype = new Error();

    const error = new AccountAlreadyExistsError("Account is in use!");
    callback(error);
};
```

Vous pouvez configurer Step Functions de façon à intercepter l'erreur à l'aide d'une règle `Catch` :

```
{
  "StartAt": "CreateAccount",
  "States": {
    "CreateAccount": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
      "Next": "SendWelcomeEmail",
      "Catch": [
        {
          "ErrorEquals": ["AccountAlreadyExistsError"],
          "Next": "SuggestAccountName"
        }
      ]
    },
    ...
  }
}
```

Lors de l'exécution, AWS Step Functions identifie l'erreur et [transitionne](#) jusqu'à l'état `SuggestAccountName` comme spécifié dans la transition `Next`.

Note

Le nom de propriété de l'objet `Error` doit correspondre à la valeur `ErrorEquals`.

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda, ce qui vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Instrumentation du code Node.js dans AWS Lambda

Dans Node.js, Lambda peut émettre des sous-segments vers X-Ray afin de vous montrer des informations sur les appels en aval effectués par votre fonction vers d'autres services AWS. Pour cela, commencez par inclure le [kit SDK AWS X-Ray pour Node.js](#) dans votre package de déploiement. De plus, encapsulez l'instruction AWS SDK `require` de la manière suivante :

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

Ensuite, utilisez la variable AWS définie dans l'exemple précédent pour initialiser n'importe quel service client que vous souhaitez suivre avec X-Ray, par exemple :

```
s3Client = AWS.S3();
```

Une fois ces étapes terminées, tous les appels effectués à partir de votre fonction à l'aide des résultats s3Client produisent un sous-segment X-Ray qui représente cet appel. Par exemple, vous pouvez exécuter la fonction Node.js suivante pour voir à quoi ressemble le suivi dans X-Ray :

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));

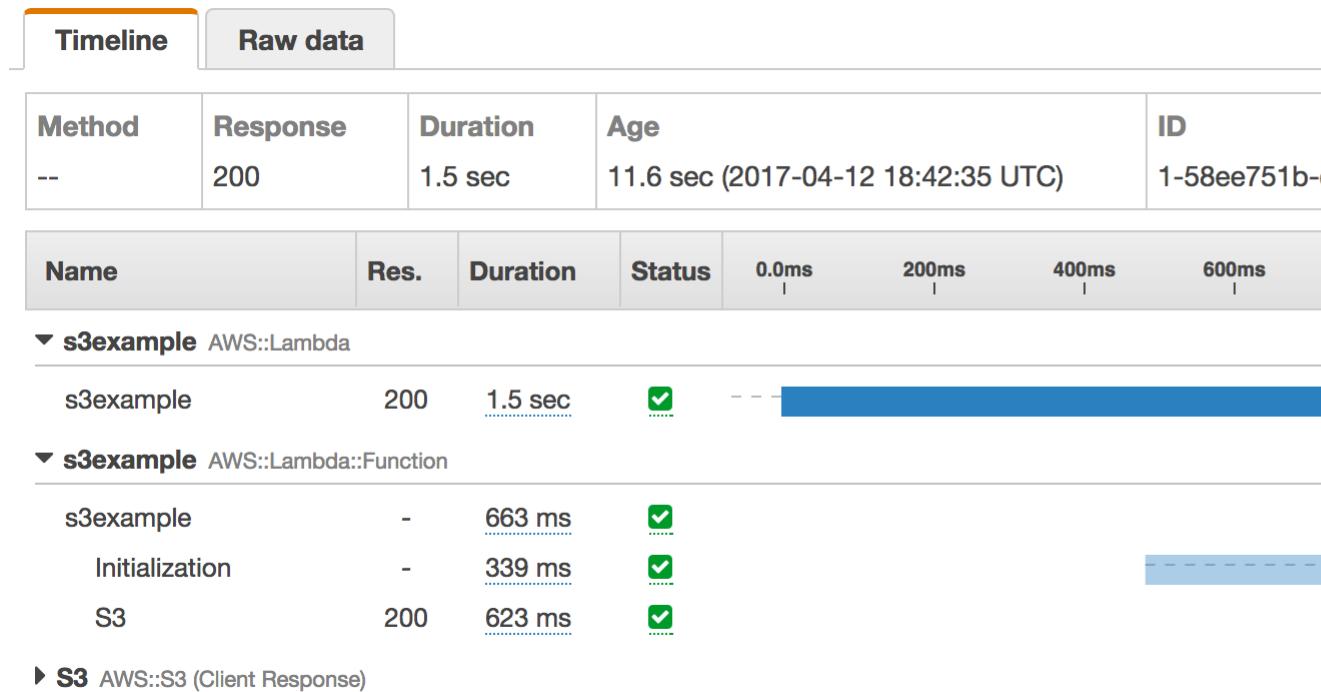
s3 = new AWS.S3({signatureVersion: 'v4'});

exports.handler = (event, context, callback) => {

    var params = {Bucket: BUCKET_NAME, Key: BUCKET_KEY, Body: BODY};

    s3.putObject(params, function(err, data) {
        if (err)
            { console.log(err) }
        else {
            console.log('success!')
        }
    });
};
```

Voici un exemple de suivi émis par le code précédent (appel asynchrone) :



Création de fonctions Lambda avec Python

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lors de la création du code d'une fonction Lambda en Python.

Runtimes Python

Nom	Identificateur
Python 3.6	<code>python3.6</code>
Python 3.7	<code>python3.7</code>
Python 2.7	<code>python2.7</code>

Rubriques

- [Package de déploiement AWS Lambda dans Python \(p. 301\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Python \(p. 305\)](#)
- [Objet de contexte AWS Lambda dans Python \(p. 306\)](#)
- [Journalisation des fonctions AWS Lambda dans Python \(p. 307\)](#)
- [Erreurs de fonction AWS Lambda dans Python \(p. 309\)](#)
- [Instrumentation du code Python dans AWS Lambda \(p. 311\)](#)

Package de déploiement AWS Lambda dans Python

A deployment package is a ZIP archive that contains your function code and dependencies. You need to create a deployment package if you use the Lambda API to manage functions, or if your code uses libraries other than the AWS SDK. Other libraries and dependencies need to be included in the deployment package. You can upload the package directly to Lambda, or you can use an Amazon S3 bucket, and then upload it to Lambda.

If you use the Lambda [console editor \(p. 17\)](#) to author your function, the console manages the deployment package. You can use this method as long as you don't need to add any libraries. You can also use it to update a function that already has libraries in the deployment package, as long as the total size doesn't exceed 3 MB.

Note

Vous pouvez utiliser la commande `build` de l'interface de ligne de commande AWS SAM pour créer un package de déploiement pour le code et les dépendances de votre fonction Python. Pour obtenir des instructions, consultez [Création d'applications avec des dépendances](#) dans le Guide du développeur AWS SAM.

Sections

- [Sans dépendances supplémentaires \(p. 302\)](#)

- [Avec des dépendances supplémentaires \(p. 302\)](#)
- [Avec un environnement virtuel \(p. 303\)](#)

Sans dépendances supplémentaires

Pour créer ou mettre à jour une fonction à l'aide de l'API Lambda, créez une archive contenant le code de votre fonction et chargez-la avec l'AWS CLI.

Pour mettre à jour une fonction Python sans dépendances

1. Créez une archive ZIP.

```
~/my-function$ zip function.zip function.py
```

2. Utilisez la commande update-function-code pour charger le package.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
    "Runtime": "python3.7",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 815,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-20T20:41:16.647+0000",
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "d1e983e3-ca8e-434b-8dc1-7add83d72ebd"
}
```

Avec des dépendances supplémentaires

Si votre fonction dépend de bibliothèques autres que le Kit SDK pour Python (Boto3), installez-les dans un répertoire local avec [pip](#), et incluez-les dans votre package de déploiement.

Pour mettre à jour une fonction Python avec des dépendances

1. Créez un répertoire pour les dépendances.

```
~/my-function$ mkdir package
```

2. Installez les bibliothèques dans le répertoire package avec l'option --target.

```
~/my-function$ cd package
~/my-function/package$ pip install Pillow --target .
```

```
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
  Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
  Installing collected packages: Pillow
  Successfully installed Pillow-5.3.0
```

3. Créez une archive ZIP.

```
package$ zip -r9 ../function.zip .
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
  adding: PIL/.libs/liblcms2-a6801db4.so.2.0.8 (deflated 67%)
  ...
```

4. Ajoutez le code de votre fonction dans l'archive.

```
~/my-function/package$ cd ../
~/my-function$ zip -g function.zip function.py
  adding: function.py (deflated 56%)
```

5. Mettez à jour le code de la fonction.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file
fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
    "Runtime": "python3.7",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 2269409,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-20T20:51:35.871+0000",
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "a9c05ffd-8ad6-4d22-b6cd-d34a00c1702c"
}
```

Avec un environnement virtuel

Dans certains cas, vous pouvez être amené à utiliser un [environnement virtuel](#) pour installer les dépendances de votre fonction. Cela peut se produire si votre fonction ou ses dépendances ont des dépendances sur les bibliothèques natives, ou si vous avez utilisé Homebrew pour installer Python.

Pour mettre à jour une fonction Python avec un environnement virtuel

1. Créez un environnement virtuel.

```
~/my-function$ virtualenv v-env
Using base prefix '/.local/python-3.7.0'
New python executable in v-env/bin/python3.7
Also creating executable in v-env/bin/python
Installing setuptools, pip, wheel...
done.
```

2. Activez l'environnement.

```
~/my-function$ source v-env/bin/activate
(v-env) ~/my-function$
```

Pour la ligne de commande Windows, le script d'activation se trouve dans le répertoire Scripts.

```
> v-env\Scripts\activate.bat
```

3. Installez les bibliothèques avec pip.

```
~/my-function$ pip install Pillow
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
  Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
  Installing collected packages: Pillow
  Successfully installed Pillow-5.3.0
```

4. Désactivez l'environnement virtuel.

```
(v-env)~/my-function$ deactivate
```

5. Créez une archive ZIP avec le contenu du répertoire site-packages.

```
~/my-function$ cd v-env/lib/python3.7/site-packages/
~/my-function/v-env/lib/python3.7/site-packages$ zip -r9 ../function.zip .
  adding: easy_install.py (deflated 17%)
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
...
```

Note

Dans certains cas, les bibliothèques peuvent également être installées dans le répertoire dist-packages.

6. Ajoutez le code de votre fonction dans l'archive.

```
~/my-function/v-env/lib/python3.7/site-packages$ cd ../../..
~/my-function$ zip -g function.zip function.py
  adding: function.py (deflated 56%)
```

7. Mettez à jour le code de la fonction.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file
fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
```

```
"Runtime": "python3.7",
"Role": "arn:aws:iam::123456789012:role/lambda-role",
"Handler": "function.handler",
"CodeSize": 5912988,
"Description": "",
"Timeout": 3,
"MemorySize": 128,
"LastModified": "2018-11-20T21:08:26.326+0000",
"CodeSha256": "A2P0NUWq1J+LtSbkuP8tm9uNYqs1TAa3M76ptmZCw5g=",
"Version": "$LATEST",
"VpcConfig": {
    "SubnetIds": [],
    "SecurityGroupIds": [],
    "VpcId": ""
},
"TracingConfig": {
    "Mode": "Active"
},
"RevisionId": "5afdc7dc-2fcb-4ca8-8f24-947939ca707f"
}
```

Gestionnaire de fonctions AWS Lambda dans Python

Lorsque vous créez une fonction Lambda, vous spécifiez un gestionnaire qui est une fonction de votre code qu'AWS Lambda peut appeler lorsque le service exécute votre code. Utilisez la structure syntaxique générale suivante lorsque vous créez une fonction de gestionnaire en Python.

```
def handler_name(event, context):
    ...
    return some_value
```

Dans la syntaxe, notez les éléments suivants :

- **event** – AWS Lambda utilise ce paramètre pour transmettre les données d'événement au gestionnaire. Ce paramètre est généralement du type Python `dict`. Il peut également s'agir du type `list`, `str`, `int`, `float` ou `NoneType`.
- **context** – AWS Lambda utilise ce paramètre pour fournir les informations d'exécution au gestionnaire. Ce paramètre est de type `LambdaContext`.
- Le cas échéant, le gestionnaire peut renvoyer une valeur. Ce qu'il advient de la valeur renvoyée dépend du type d'appel utilisé pour la fonction Lambda :
 - Si vous utilisez le type d'appel `RequestResponse` (exécution synchrone), AWS Lambda renvoie le résultat de l'appel de la fonction Python au client qui appelle la fonction Lambda (dans la réponse HTTP à la demande d'appel, sérialisée au format JSON). Par exemple, la console AWS Lambda utilise le type d'appel `RequestResponse`. Dès lors, lorsque vous appelez la fonction à l'aide de la console, cette dernière affiche la valeur renvoyée.

Si le gestionnaire renvoie `NONE`, AWS Lambda renvoie `null`.

- Si vous utilisez le type d'appel `Event` (exécution asynchrone), la valeur est ignorée.

Prenons l'exemple de code Python suivant.

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
```

```
        event['last_name'])  
    return {  
        'message' : message  
    }
```

Cet exemple possède une fonction appelée `my_handler`. Cette fonction renvoie un message contenant les données d'événement qu'elle a reçues comme entrée.

Objet de contexte AWS Lambda dans Python

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 305\)](#). Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'appel, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `get_remaining_time_in_millis` – Returns the number of milliseconds left before the execution times out.

Propriétés du contexte

- `function_name` – The name of the Lambda function.
- `function_version` – The [version \(p. 38\)](#) of the function.
- `invoked_function_arn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `memory_limit_in_mb` – The amount of memory configured on the function.
- `aws_request_id` – The identifier of the invocation request.
- `log_group_name` – The log group for the function.
- `log_stream_name` – The log stream for the function instance.
- `deadline_ms` – The date that the execution times out, in Unix time milliseconds.
- `identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
 - `cognito_identity_id` – The authenticated Amazon Cognito identity.
 - `cognito_identity_pool_id` – The Amazon Cognito identity pool that authorized the invocation.
- `client_context` – (mobile apps) Client context provided to the Lambda invoker by the client application.
 - `client.installation_id`
 - `client.app_title`
 - `client.app_version_name`
 - `client.app_version_code`
 - `cclient.app_package_name`
 - `custom` – dict de valeurs personnalisées définies par l'application client mobile.
 - `env` – dict des informations d'environnement fournies par le kit SDK AWS.

L'exemple suivant montre une fonction de gestionnaire qui consigne les informations de contexte.

Example handler.py

```
import time  
def get_my_log_stream(event, context):  
    print("Log stream name:", context.log_stream_name)
```

```
print("Log group name:", context.log_group_name)
print("Request ID:", context.aws_request_id)
print("Mem. limits(MB):", context.memory_limit_in_mb)
# Code will execute quickly, so we add a 1 second intentional delay so you can see that
in time remaining value.
time.sleep(1)
print("Time remaining (MS):", context.get_remaining_time_in_millis())
```

En plus des options ci-dessus, vous pouvez également utiliser le kit SDK AWS X-Ray pour [Instrumentation du code Python dans AWS Lambda \(p. 311\)](#) afin d'identifier les chemins de code critiques, suivre leurs performances et capturer les données pour l'analyse.

Journalisation des fonctions AWS Lambda dans Python

Votre fonction Lambda peut contenir des instructions de journalisation. AWS Lambda écrit ces journaux dans CloudWatch. Si vous utilisez la console Lambda pour appeler la fonction Lambda, la console affiche les mêmes journaux.

Les instructions Python suivantes génèrent des entrées de journal :

- Instructions `print`
- Fonctions `Logger` du module `logging` (par exemple, `logging.Logger.info` et `logging.Logger.error`)

Les fonctions `print` et `logging.*` écrivent toutes les deux des journaux dans CloudWatch Logs, mais la fonction `logging.*` écrit des informations supplémentaires pour chaque entrée, telles que l'horodatage et le niveau de journalisation.

Prenons l'exemple de code Python suivant.

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def my_logging_handler(event, context):
    logger.info('got event{}'.format(event))
    logger.error('something went wrong')
    return 'Hello from Lambda!'
```

Etant donné que l'exemple de code utilise le module `logging` pour écrire des messages dans les journaux, vous obtenez également des informations supplémentaires dans les journaux, comme l'horodatage et les niveaux de journalisation. Le niveau de journalisation identifie le type de journal ([`INFO`], [`ERROR`] et [`DEBUG`], par exemple).

Vous trouverez également ces journaux dans CloudWatch. Pour plus d'informations, consultez [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 227\)](#).

Au lieu du module `logging`, vous pouvez utiliser les instructions `print` de votre code, comme illustré dans l'exemple Python suivant :

```
from __future__ import print_function
def lambda_handler(event, context):
    print('this will also show up in cloud watch')
    return 'Hello World!'
```

Dans ce cas, seul le texte transmis à la méthode d'impression est envoyé à CloudWatch. Les entrées de journal n'ont pas d'informations supplémentaires renvoyées par la fonction `logging.*`. L'instruction `from __future__` vous permet d'écrire du code qui est compatible avec Python 2 ou 3.

The screenshot shows the AWS Lambda function execution details page. At the top, a green checkmark indicates "Execution result: succeeded (logs)". Below this, a "Details" section shows the returned value: "Hello World!". The "Summary" section includes the following information:

- Code SHA-256: Mj6sNoYuSX2UjGoXUy
64+R/UvzWB0xJc7VdX
HRN6hqY=
- Request ID: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559
- Duration: 0.18 ms
- Billed duration: 100 ms

The "Log output" section displays the following log entries:

```
START RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559 Version: $LATEST
this will also show up in cloud watch
END RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559
REPORT RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559 Duration: 0.18 ms      Billed
Duration: 100 ms      Memory Size: 128 MB      Max Memory Used: 17 MB
```

La console utilise le type d'appel `RequestResponse` (appel synchrone) lors de l'appel de la fonction. Par conséquent, il obtient la valeur de retour (« Hello from Lambda! ») d'AWS Lambda qu'affiche la console.

Pour tester le code Python précédent (console)

1. Dans la console, créez une fonction Lambda avec le plan hello-world-python. Dans runtime, sélectionnez Python 2.7. Dans Handler, remplacez `lambda_function.lambda_handler` par `lambda_function.my_other_logging_handler` et, dans Role, sélectionnez Basic execution role. Vous devez également remplacer le code fourni par le plan par le code de cette section. Pour obtenir des instructions détaillés sur la création d'une fonction Lambda à l'aide de la console, consultez [Créer une fonction Lambda avec la console \(p. 6\)](#).
2. Remplacez le code de modèle par le code fourni dans cette section.
3. Testez la fonction Lambda avec l'exemple de modèle d'événement Hello World fourni dans la console Lambda.

Recherche des journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Dans la console AWS Lambda – La section Sortie de journal de la console AWS Lambda affiche les journaux.
- Dans l'en-tête de la réponse, lorsque vous appelez une fonction Lambda par programmation – Si vousappelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les quatre derniers kilooctets de données de journal écrites dans CloudWatch Logs. AWS Lambda renvoie ces informations de journal dans l'en-tête `x-amz-log-results` de la réponse. Pour plus d'informations, consultez [Invoke \(p. 452\)](#).

Si vous utilisez AWS CLI pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type parameter` avec la valeur `Tail` pour récupérer les mêmes informations.

- Dans CloudWatch Logs : pour trouver vos journaux dans CloudWatch, vous devez connaître le nom du groupe de journaux, ainsi que le nom de leur flux. Vous pouvez utiliser les propriétés `context.log_group_name` et `context.log_stream_name` de votre code pour obtenir ces informations. Lorsque vous exécutez la fonction Lambda, les journaux générés dans la console ou l'interface de ligne de commande indiquent le nom du groupe de journaux et le nom de leur flux.

Erreurs de fonction AWS Lambda dans Python

Si la fonction Lambda déclenche une exception, AWS Lambda reconnaît l'échec, puis sérialise les informations correspondantes dans JSON et les renvoie. Prenez l'exemple suivant :

```
def always_failed_handler(event, context):
    raise Exception('I failed!')
```

Lorsque vous appelez la fonction Lambda, celle-ci déclenche une exception et AWS Lambda renvoie le message d'erreur suivant :

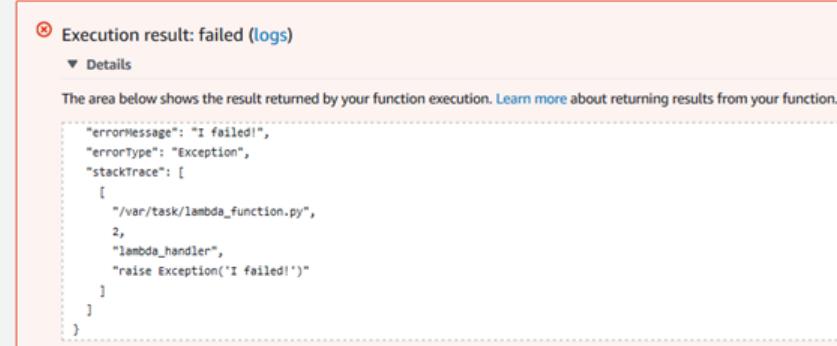
```
{
  "errorMessage": "I failed!",
  "stackTrace": [
    [
      "/var/task/lambda_function.py",
      3,
      "my_always_fails_handler",
      "raise Exception('I failed!')"
    ]
  ],
  "errorType": "Exception"
}
```

Notez que la trace de la pile est renvoyée sous la forme du tableau JSON `stackTrace` constitué des éléments de cette trace.

La façon dont vous recevez les informations d'erreur varie selon le type d'appel que le client spécifie au moment de l'appel d'une fonction :

- Si le client spécifie le type d'appel `RequestResponse` (c'est-à-dire, l'exécution synchrone), elle renvoie le résultat au client qui est à l'origine de l'appel « `invoke` ».

Par exemple, la console utilise toujours le type d'appel `RequestResponse`. Par conséquent, elle affiche l'erreur comme suit dans la section `Execution result` :



Les mêmes informations sont également envoyées à CloudWatch et la section Sortie de journal affiche les mêmes journaux.

The screenshot shows the AWS Lambda function details page. At the top, there's a summary table with metrics like Request ID, Duration, and Max memory used. Below the summary is a 'Log output' section containing a log entry from CloudWatch. The log entry starts with 'START' and ends with 'END'. It shows an exception being raised in the lambda_handler function of the lambda_function.py file. The log entry is as follows:

```
START RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46 Version: $LATEST
I failed: Exception
Traceback (most recent call last):
  File "/var/task/lambda_function.py", line 2, in lambda_handler
    raise Exception('I failed!')
Exception: I failed!

END RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46
REPORT RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46 Duration: 0.82 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 22 MB
```

- Si le client spécifie le type d'appel Event (c'est-à-dire, l'exécution asynchrone), AWS Lambda ne renvoie rien. Au lieu de cela, il consigne les informations d'erreur dans CloudWatch Logs. Vous pouvez également voir les métriques d'erreur dans les métriques CloudWatch.

En fonction de la source d'événement, AWS Lambda réessaie parfois d'exécuter la fonction Lambda qui a échoué. Par exemple, si Kinesis est la source d'événement, AWS Lambda réessaie d'exécuter l'appel qui a échoué jusqu'à ce que la fonction Lambda aboutisse ou jusqu'à ce que les enregistrements du flux expirent.

Pour tester le code Python précédent (console)

1. Dans la console, créez une fonction Lambda avec le plan hello-world. Dans runtime, choisissez Python 3.7. Dans Handler, remplacez `lambda_function.lambda_handler` par `lambda_function.always_failed_handler`. Pour obtenir des instructions sur la façon de procéder, consultez [Créer une fonction Lambda avec la console \(p. 6\)](#).
2. Remplacez le code de modèle par le code fourni dans cette section.
3. Testez la fonction Lambda avec l'exemple de modèle d'événement Hello World fourni dans la console Lambda.

Gestion des erreurs de fonction

Vous pouvez créer la gestion des erreurs personnalisées pour déclencher une exception directement depuis la fonction Lambda et la gérer directement (Réessayer ou CATCH) dans le cadre d'une machine d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

Prenons l'exemple d'un état `CreateAccount` qui est une tâche écrivant les détails d'un client dans une base de données à l'aide d'une fonction Lambda.

- Si la tâche réussit, un compte est créé et un e-mail de bienvenue est envoyé.
- Si un utilisateur essaie de créer un compte pour un nom d'utilisateur qui existe déjà, la fonction Lambda génère une erreur, ce qui pousse la machine d'état à suggérer un autre nom d'utilisateur et à recommencer le processus de création de compte.

L'extrait de code suivant montre comment procéder. Notez que les erreurs personnalisées dans Python doivent étendre la classe `Exception`.

```
class AccountAlreadyExistsException(Exception): pass

def create_account(event, context):
    raise AccountAlreadyExistsException('Account is in use!')
```

Vous pouvez configurer Step Functions de façon à intercepter l'erreur à l'aide d'une règle `Catch`. Lambda définit automatiquement le nom de l'erreur comme étant le nom de classe simple de l'exception lors de l'exécution :

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": [ "AccountAlreadyExistsException" ],
                    "Next": "SuggestAccountName"
                }
            ]
        },
        ...
    }
}
```

Lors de l'exécution, AWS Step Functions identifie l'erreur et [transitionne](#) jusqu'à l'état `SuggestAccountName` comme spécifié dans la transition `Next`.

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda, ce qui vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Instrumentation du code Python dans AWS Lambda

Dans Python, Lambda peut émettre des sous-segments vers X-Ray afin de vous montrer des informations sur les appels en aval effectués par votre fonction vers d'autres services AWS. Pour cela, commencez par inclure le [kit SDK AWS X-Ray pour Python](#) dans votre package de déploiement. De plus, vous pouvez corriger le module `boto3` (ou `botocore` si vous utilisez des sessions) pour que tout client que vous créez pour accéder à d'autres services AWS soit suivi par X-Ray.

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])
```

Une fois que vous avez corrigé le module que vous utilisez pour créer des clients, vous pouvez l'utiliser pour créer vos clients suivis, dans le cas ci-dessous Amazon S3 :

```
s3_client = boto3.client('s3')
```

Le kit SDK X-Ray pour Python crée un sous-segment pour l'appel et enregistre les informations provenant de la demande et de la réponse. Vous pouvez utiliser `aws_xray_sdk.core.xray_recorder` pour créer des sous-segments, automatiquement en configurant vos fonctions Lambda ou manuellement en appelant `xray_recorder.begin_subsegment()` et `xray_recorder.end_subsegment()` à l'intérieur de la fonction, comme indiqué dans la fonction Lambda suivante.

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    bucket_name = event['bucket_name']
    bucket_key = event['bucket_key']
    body = event['body']

    put_object_into_s3(bucket_name, bucket_key, body)
    get_object_from_s3(bucket_name, bucket_key)

# Define subsegments manually
def put_object_into_s3(bucket_name, bucket_key, body):
    try:
        xray_recorder.begin_subsegment('put_object')
        response = s3_client.put_object(Bucket=bucket_name, Key=bucket_key, Body=body)
        status_code = response['ResponseMetadata']['HTTPStatusCode']
        xray_recorder.current_subsegment().put_annotation('put_response', status_code)
    finally:
        xray_recorder.end_subsegment()

# Use decorators to automatically set the subsegments
@xray_recorder.capture('get_object')
def get_object_from_s3(bucket_name, bucket_key):
    response = s3_client.get_object(Bucket=bucket_name, Key=bucket_key)
    status_code = response['ResponseMetadata']['HTTPStatusCode']
    xray_recorder.current_subsegment().put_annotation('get_response', status_code)
```

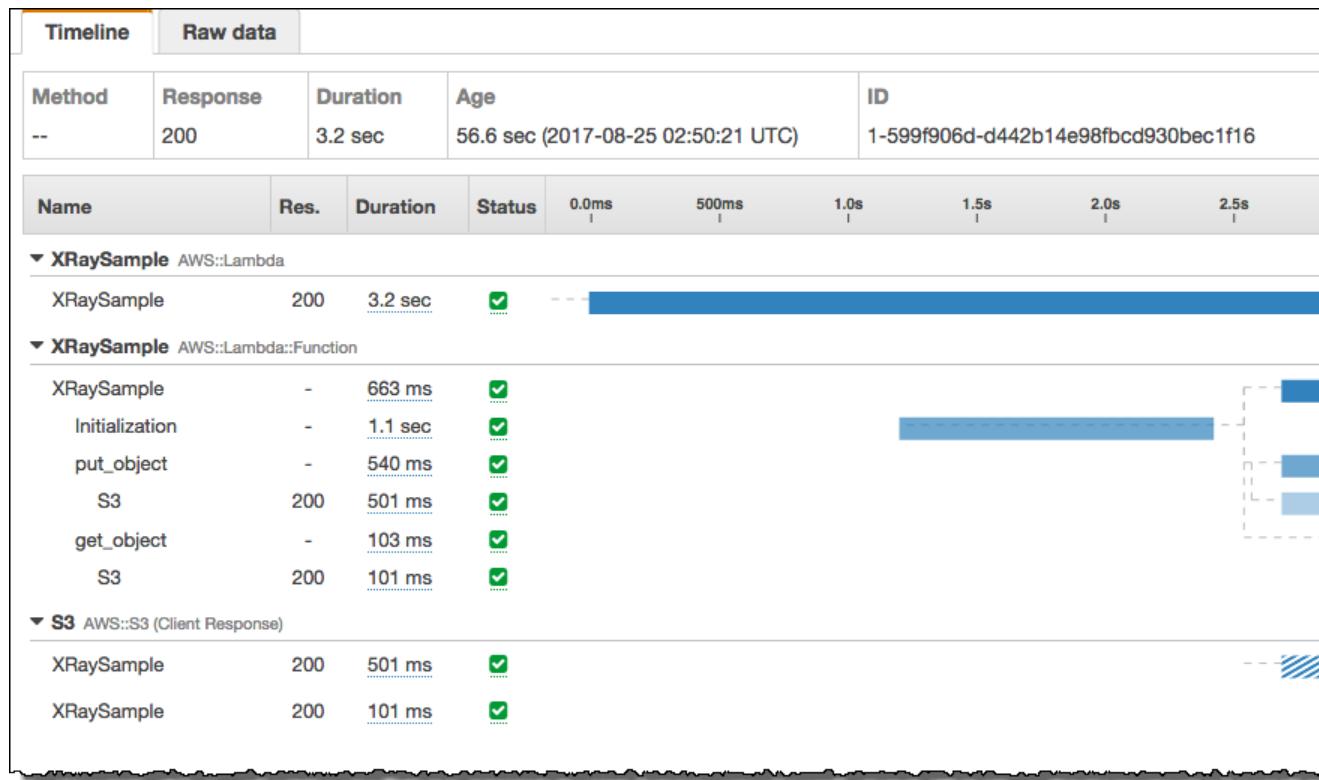
Note

Le kit SDK X-Ray pour Python vous permet de corriger les modules suivants :

- botocore
- boto3
- requêtes
- sqlite3
- mysql

Vous pouvez utiliser `patch_all()` pour tous les corriger en une seule fois.

Voici un exemple de suivi émis par le code précédent (appel synchrone) :



Création de fonctions Lambda avec Java

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lors de la création du code d'une fonction Lambda en Java.

Runtimes Java

Nom	Identificateur	JDK
Java 8	java8	java-1.8.0-openjdk

Rubriques

- [Package de déploiement AWS Lambda dans Java \(p. 315\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Java \(p. 322\)](#)
- [Objet de contexte AWS Lambda dans Java \(p. 333\)](#)
- [Journalisation des fonctions AWS Lambda dans Java \(p. 334\)](#)
- [Erreurs de fonction AWS Lambda dans Java \(p. 338\)](#)
- [Instrumentation du code Java dans AWS Lambda \(p. 340\)](#)
- [Création d'une fonction Lambda en Java \(p. 342\)](#)

En outre, notez qu'AWS Lambda fournit les bibliothèques suivantes :

- aws-lambda-java-core : cette bibliothèque fournit l'objet de contexte, `RequestStreamHandler`, et les interfaces `RequestHandler`. L'objet Context ([Objet de contexte AWS Lambda dans Java \(p. 333\)](#)) indique les informations d'exécution concernant la fonction Lambda. Les interfaces prédéfinies offrent un moyen de définir le gestionnaire de la fonction Lambda. Pour en savoir plus, consultez [Utilisation des interfaces prédéfinies pour la création d'un gestionnaire \(Java\) \(p. 329\)](#).
- aws-lambda-java-events : cette bibliothèque fournit des types prédéfinis que vous pouvez utiliser lors de l'écriture de fonctions Lambda pour traiter les événements publiés par Amazon S3, Kinesis, Amazon SNS et Amazon Cognito. Ces classes vous aident à traiter l'événement sans avoir à écrire votre propre logique de sérialisation personnalisée.
- Appender personnalisé pour Log4j2 : vous pouvez utiliser l'appender Log4j personnalisé (voir [Apache Log4j 2](#)) fourni par AWS Lambda pour la journalisation à partir des fonctions Lambda. Chaque appel aux méthodes Log4j, comme `log.info()` ou `log.error()`, entraîne un événement CloudWatch Logs. L'appender personnalisé est appelé `LambdaAppender` et doit être utilisé dans le fichier `log4j2.xml`. Vous devez inclure l'artefact `aws-lambda-java-log4j2` (`artifactId:aws-lambda-java-log4j2`) dans le package de déploiement (fichier `.jar`). Pour en savoir plus, consultez [Journalisation des fonctions AWS Lambda dans Java \(p. 334\)](#).
- Appender personnalisé pour Log4j1.2 : vous pouvez utiliser l'appender Log4j personnalisé (voir [Apache Log4j 1.2](#)) fourni par AWS Lambda pour la journalisation à partir des fonctions Lambda. Pour plus d'informations, consultez [Journalisation des fonctions AWS Lambda dans Java \(p. 334\)](#).

Note

La prise en charge de l'appender personnalisé Log4j v1.2 sera prochainement en fin de vie. Cet appender ne sera plus mis à jour, et son utilisation est déconseillée.

Ces bibliothèques sont disponibles via le [référentiel Maven Central](#) et sont également disponibles sur [GitHub](#).

Package de déploiement AWS Lambda dans Java

Votre package de déploiement peut être un fichier .zip ou un fichier .jar autonome, à votre convenance. Vous pouvez utiliser n'importe quel outil de création et de compression de fichier que vous connaissez pour créer un package de déploiement.

Nous fournissons des exemples d'utilisation de Maven et de Gradle pour créer des fichiers jar autonomes et des fichiers .zip, respectivement. Pour plus d'informations, consultez les rubriques suivantes :

Rubriques

- [Création d'un package de déploiement .jar via Maven sans IDE \(Java\) \(p. 315\)](#)
- [Création d'un package de déploiement .jar via Maven et l'IDE Eclipse \(Java\) \(p. 317\)](#)
- [Création d'un package de déploiement .zip \(Java\) \(p. 320\)](#)
- [Création de fonctions Lambda à l'aide de l'IDE Eclipse et du plug-in AWS SDK \(Java\) \(p. 322\)](#)

Création d'un package de déploiement .jar via Maven sans IDE (Java)

Cette section décrit comment compresser le code Java dans un package de déploiement via Maven au niveau de la ligne de commande.

Rubriques

- [Avant de commencer \(p. 315\)](#)
- [Présentation de la structure d'un projet \(p. 315\)](#)
- [Étape 1 : Création du projet \(p. 316\)](#)
- [Étape 2 : Création du package de déploiement \(p. 317\)](#)

Avant de commencer

Vous devez installer l'outil de développement de ligne de commande Maven. Pour en savoir plus, consultez la page [Maven](#). Si vous utilisez Linux, vérifiez votre gestionnaire de package.

```
sudo apt-get install mvn
```

Si vous utilisez Homebrew

```
brew install maven
```

Présentation de la structure d'un projet

Une fois que vous configurez le projet, vous devez avoir la structure de dossiers suivante :

```
project-dir/pom.xml
```

```
project-dir/src/main/java/ (your code goes here)
```

Le code sera ensuite dans le dossier /java. Par exemple, si le nom du package est `exemple` et qu'il contient une classe `Hello.java`, la structure sera la suivante :

```
project-dir/src/main/java/exemple/Hello.java
```

Une fois que vous créez le projet, le fichier .jar généré (autrement dit, votre package de déploiement) se trouvera dans le sous-répertoire `project-dir/target`.

Étape 1 : Création du projet

Suivez les étapes de cette section pour créer un projet Java.

1. Créez un répertoire de projet (`project-dir`).
2. Dans le répertoire `project-dir`, créez les éléments suivants :
 - Fichier de modèle d'objet du projet, `pom.xml`. Ajoutez les informations de projet et de configuration suivantes pour que Maven crée le projet.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>doc-examples</groupId>
  <artifactId>lambda-java-example</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>lambda-java-example</name>

  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.1.0</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Note

- Dans la section `dependencies`, la valeur `groupId` (`com.amazonaws`) correspond à l'ID du groupe Amazon AWS pour les objets Maven du référentiel centralisé de Maven. La valeur `artifactId` (`aws-lambda-java-core`) correspond à la bibliothèque AWS Lambda principale qui définit les interfaces `AWS Lambda RequestHandler`, `RequestStreamHandler` et `Context` à utiliser dans votre application Java. Maven résout ces dépendances au moment de la création du package de déploiement.
- Dans la section des plug-ins, `maven-shade-plugin` est un plug-in Apache que Maven télécharge et utilise au cours du processus de création du package de déploiement. Ce plug-in est utilisé pour compresser les fichiers .jar dans un fichier .jar autonome (fichier .zip). Ce fichier représentera votre package de déploiement.
- Si vous suivez d'autres rubriques de didacticiel dans ce guide, vous devrez peut-être ajouter des dépendances supplémentaires. Assurez-vous d'ajouter toutes les dépendances requises.

3. Dans le répertoire `project-dir`, créez la structure suivante :

```
project-dir/src/main/java
```

4. Le cas échéant, ajoutez vos fichiers Java et la structure du dossier dans le sous-répertoire / java. Par exemple, si le nom de votre package Java est `example` et si le code source est `Hello.java`, la structure de votre répertoire ressemblera à ceci :

```
project-dir/src/main/java/example/Hello.java
```

Étape 2 : Création du package de déploiement

Désormais, vous pouvez créer le package de déploiement de votre projet via Maven au niveau de la ligne de commande.

1. A l'invite de commande, indiquez le répertoire du projet (`project-dir`).
2. Exécutez la commande `mvn` suivante pour créer le package de déploiement :

```
$ mvn package
```

Le fichier .jar généré est enregistré sous `project-dir/target/lambda-java-example-1.0-SNAPSHOT.jar`. Son nom a été créé en concaténant les valeurs `artifactId` et `version` dans le fichier `pom.xml`.

Le processus crée le fichier .jar généré à l'aide des informations du fichier `pom.xml` pour effectuer les transformations nécessaires. Il s'agit d'un fichier .jar autonome (fichier .zip) qui inclut toutes les dépendances. C'est le package de déploiement que vous pourrez importer dans AWS Lambda pour créer une fonction Lambda.

Création d'un package de déploiement .jar via Maven et l'IDE Eclipse (Java)

Cette section montre comment compresser votre code Java dans un package de déploiement à l'aide de l'IDE Eclipse et du plug-in Maven pour Eclipse.

Rubriques

- [Avant de commencer \(p. 318\)](#)
- [Étape 1 : Création et développement d'un projet \(p. 318\)](#)

Avant de commencer

Installez le plug-in Maven pour Eclipse.

1. Démarrez Eclipse. Dans le menu Help d'Eclipse, sélectionnez Install New Software.
2. Dans la fenêtre Install (Installer), saisissez <http://download.eclipse.org/technology/m2e/releases> dans la zone Work with : (Travailler avec :), puis choisissez Add (Ajouter).
3. Suivez les étapes indiquées pour terminer l'installation.

Étape 1 : Création et développement d'un projet

Dans cette étape, vous démarrez Eclipse et créez un projet Maven. Vous ajoutez les dépendances nécessaires et générer le projet. Cette opération générera un fichier.jar, qui sera votre package de déploiement.

1. Créez un projet Maven dans Eclipse.
 - a. Dans le menu File, sélectionnez New, puis Project.
 - b. Dans la fenêtre New Project, sélectionnez Maven Project.
 - c. Dans la fenêtre New Maven Project, sélectionnez Create a simple project et conservez les autres sélections par défaut.
 - d. Dans les fenêtres New Maven Project, Configure project, tapez les informations Artifact suivantes :
 - Group Id : doc-examples
 - Artifact Id : lambda-java-example
 - Version : 0.0.1-SNAPSHOT
 - Packaging : jar
 - Name : lambda-java-example
2. Ajoutez la dépendance `aws-lambda-java-core` dans le fichier `pom.xml`.

Elle fournit les définitions des interfaces `RequestHandler`, `RequestStreamHandler` et `Context`. Cela vous permet de compiler le code que vous pouvez utiliser avec AWS Lambda.

- a. Ouvrez le menu contextuel (via un clic droit) du fichier `pom.xml`, puis sélectionnez Maven et Add Dependency.
- b. Dans les fenêtres Add Dependency, entrez les valeurs suivantes :

Group Id : com.amazonaws

Artifact Id : aws-lambda-java-core

Version : 1.1.0

Note

Si vous suivez d'autres rubriques de didacticiel dans ce guide, vous devrez peut-être ajouter des dépendances supplémentaires. Assurez-vous d'ajouter toutes les dépendances requises.

3. Ajoutez la classe Java au projet.

- a. Ouvrez le menu contextuel (via un clic droit) du sous-répertoire de projet `src/main/java`, puis sélectionnez New et Class.
- b. Dans la fenêtre New Java Class, entrez les valeurs suivantes :

- Package : **example**
- Name : **Hello**

Note

Si vous suivez les autres didacticiels de ce guide, les noms de package et de classe peuvent différer.

- c. Ajoutez le code Java. Si vous suivez d'autres didacticiels de ce guide, ajoutez le code spécifique qu'ils indiquent.
4. Générez le projet.

Ouvrez le menu contextuel (via un clic droit) du projet dans Package Explorer, puis sélectionnez Run As et Maven Build. Dans la fenêtre Edit Configuration (Modifier la configuration), tapez **package** dans la case Goals (Objectifs).

Note

Le fichier .jar généré, `lambda-java-example-0.0.1-SNAPSHOT.jar`, n'est pas le fichier .jar autonome final que vous pouvez utiliser comme package de déploiement. Dans l'étape suivante, vous ajouterez le plug-in Apache maven-shade-plugin pour créer le fichier .jar autonome. Pour en savoir plus, consultez la section [Plug-in Apache Maven Shade](#).

5. Ajoutez le plug-in maven-shade-plugin générez le projet à nouveau.

Le plug-in Maven Shade utilise les objets (fichiers jar) générés par le package (qui génère un fichier .jar de code client) et crée un fichier un .jar autonome contenant le code client compilé et les dépendances résolues à partir du fichier `pom.xml`.

- a. Ouvrez le menu contextuel (via un clic droit) du fichier `pom.xml`, puis sélectionnez Maven et Add Plugin.
- b. Dans la fenêtre Add Plugin, entrez les valeurs suivantes :
 - Group Id : `org.apache.maven.plugins`
 - Artifact Id : `maven-shade-plugin`
 - Version : `2.3`
- c. Générez le projet à nouveau.

Cette fois-ci, nous crérons le fichier jar comme précédemment, puis nous utiliserons `maven-shade-plugin` pour extraire les dépendances afin de générer le fichier .jar autonome.

- i. Ouvrez le menu contextuel (via un clic droit) du projet, puis sélectionnez Run As et Maven build.
- ii. Dans les fenêtres Edit Configuration (Modifier la configuration), saisissez **package shade : shade** dans la case Goals (Objectifs).
- iii. Choisissez Run.

Vous trouverez le fichier .jar autonome généré (autrement dit, votre package de déploiement) dans le sous-répertoire `/target` .

Ouvrez le menu contextuel (via un clic droit) du sous-répertoire `/target`, puis sélectionnez Show In et System Explorer pour trouver le fichier `lambda-java-example-0.0.1-SNAPSHOT.jar`.

Création d'un package de déploiement .zip (Java)

Cette section fournit des exemples de création d'un fichier .zip comme package de déploiement. Vous pouvez utiliser n'importe quel l'outil de création et de compression de fichier pour créer ce fichier zip. Quels que soient les outils que vous utilisez, le fichier .zip généré doit avoir la structure suivante :

- Tous les fichiers de ressources et les fichiers de classe compilés au niveau racine.
- Tous les fichiers jar requis pour exécuter le code dans le répertoire /lib.

Note

Vous pouvez aussi créer un fichier .jar autonome (qui constitue également un fichier compressé) comme package de déploiement. Pour obtenir des exemples de création d'un fichier .jar autonome à l'aide de Maven, consultez la section [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Les exemples suivants utilisent l'outil de création et de déploiement Gradle pour créer le fichier .zip.

Important

Gradle 2.0 ou version ultérieure est obligatoire.

Avant de commencer

Vous devez télécharger Gradle. Pour en savoir plus, consultez le site web de Gradle, <https://gradle.org/>.

Exemple 1 : Création d'un fichier .zip via Gradle et le référentiel Maven Central

A la fin de cette procédure, vous aurez un répertoire de projet (*project-dir*) avec la structure suivante :

```
project-dir/build.gradle  
project-dir/src/main/java/
```

Le dossier /java contiendra le code. Par exemple, si le nom du package est `example` et qu'il contient une classe `Hello.java`, la structure sera la suivante :

```
project-dir/src/main/java/example/Hello.java
```

Une fois que vous créez le projet, le fichier .zip généré (autrement dit, votre package de déploiement) se trouvera dans le sous-répertoire *project-dir*/build/distributions.

1. Créez un répertoire de projet (*project-dir*).
2. Dans le répertoire *project-dir*, créez le fichier `build.gradle` et ajoutez le contenu suivant :

```
apply plugin: 'java'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    compile ('com.amazonaws:aws-lambda-java-core:1.1.0',  
            'com.amazonaws:aws-lambda-events:1.1.0')
```

```
        )
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.compileClasspath
    }
}

build.dependsOn buildZip
```

Note

- La section des référentiels fait référence au référentiel centralisé Maven. Lors de la création du package de déploiement, il récupère les dépendances (c'est-à-dire, les deux bibliothèques AWS Lambda) à partir de Maven Central.
- La tâche `buildZip` décrit comment créer le fichier .zip du package de déploiement.

Par exemple, si vous décompressez le fichier .zip généré, vous devriez trouver tous les fichiers de classe compilés et tous les fichiers de ressources du niveau racine. Vous devriez également voir un répertoire /lib avec les fichiers jar nécessaires pour exécuter le code.

- Si vous suivez d'autres rubriques de didacticiel dans ce guide, vous devrez peut-être ajouter des dépendances supplémentaires. Assurez-vous d'ajouter toutes les dépendances requises.

3. Dans le répertoire `project-dir`, créez la structure suivante :

```
project-dir/src/main/java/
```

4. Le cas échéant, ajoutez vos fichiers Java et la structure du dossier dans le sous-répertoire / java. Par exemple, si le nom de votre package Java est `example` et si le code source est `Hello.java`, la structure de votre répertoire ressemblera à ceci :

```
project-dir/src/main/java/example/Hello.java
```

5. Exécutez la commande Gradle suivante pour générer et compresser le projet dans un fichier .zip.

```
project-dir> gradle build
```

6. Vérifiez le fichier `project-dir.zip` obtenu dans le sous-répertoire `project-dir/build/distributions`.
7. Vous pouvez désormais importer le fichier .zip (votre package de déploiement) dans AWS Lambda pour créer une fonction Lambda et l'appeler manuellement via l'échantillon de données d'événement afin de la tester. Pour obtenir des instructions, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#).

Exemple 2 : Création d'un fichier zip via Gradle à l'aide des fichiers jar locaux

Vous pouvez choisir de ne pas utiliser le référentiel Maven Central. Au lieu de cela, vous pouvez inclure toutes les dépendances dans le dossier du projet. Dans ce cas, le dossier du projet (`project-dir`) aura la structure suivante :

```
project-dir/jars/          (all jars go here)
project-dir/build.gradle
```

```
project-dir/src/main/java/ (your code goes here)
```

Donc, si le code Java définit le package `exemple` et la classe `Hello.java`, le code sera dans le sous-répertoire suivant :

```
project-dir/src/main/java/exemple/Hello.java
```

Le fichier `build.gradle` devrait être comme suit :

```
apply plugin: 'java'

dependencies {
    compile fileTree(dir: 'jars', include: '*.jar')
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.compileClasspath
    }
}

build.dependsOn buildZip
```

Notez que les dépendances spécifient `fileTree`, qui identifie `project-dir/jars` tel le sous-répertoire qui comprendra tous les fichiers jar requis.

Vous pouvez maintenant créer le package. Exécutez la commande Gradle suivante pour générer et compresser le projet dans un fichier `.zip`.

```
project-dir> gradle build
```

Création de fonctions Lambda à l'aide de l'IDE Eclipse et du plug-in AWS SDK (Java)

AWS Toolkit pour Eclipse fournit un plug-in Eclipse pour vous permettre de créer un package de déploiement et de l'importer pour créer une fonction Lambda. Si vous pouvez utiliser l'IDE Eclipse en tant qu'environnement de développement, ce plug-in vous permet de concevoir le code Java, de créer et d'importer un package de déploiement et de créer la fonction Lambda. Pour plus d'informations, consultez le [Guide de démarrage AWS Toolkit for Eclipse](#). Pour voir un exemple d'utilisation de la boîte à outils de création de fonctions Lambda, consultez [Utilisation d'AWS Lambda avec AWS Toolkit pour Eclipse](#).

Gestionnaire de fonctions AWS Lambda dans Java

Lorsque vous créez une fonction Lambda, vous spécifiez un gestionnaire qu'AWS Lambda peut appeler lorsque ce service exécute la fonction Lambda en votre nom.

Lambda prend en charge deux approches pour la création d'un gestionnaire :

- Le chargement direct de la méthode de gestionnaire sans avoir à mettre en œuvre une interface. Cette section décrit cette approche.
- Mise en œuvre des interfaces standard fournies dans le cadre de la bibliothèque `aws-lambda-java-core` (approche basée sur des interfaces). Pour en savoir plus, consultez [Utilisation des interfaces prédefinies pour la création d'un gestionnaire \(Java\) \(p. 329\)](#).

Voici la syntaxe générale du gestionnaire :

```
outputType handler-name(inputType input, Context context) {  
    ...  
}
```

Pour permettre à AWS Lambda d'appeler avec succès un gestionnaire, il doit être appelé avec des données d'entrée qui peuvent être sérialisées dans le type de données du paramètre `input`.

Dans la syntaxe, notez les éléments suivants :

- **`inputType`** – Le premier paramètre du gestionnaire sont les données d'entrée, qui peuvent correspondre à des données d'événement (publiées par une source d'événement) ou à des données d'entrée personnalisées que vous spécifiez, telles qu'une chaîne ou n'importe quel objet de données personnalisé. Pour permettre à AWS Lambda d'appeler avec succès ce gestionnaire, la fonction doit être appelée avec des données d'entrée qui peuvent être sérialisées dans le type de données du paramètre `input`.
- **`outputType`** : si vous avez l'intention d'appeler la fonction Lambda de manière synchrone (via le type d'appel `RequestResponse`), vous pouvez renvoyer la sortie de la fonction via l'un des types de données pris en charge. Par exemple, si vous utilisez une fonction Lambda en tant que back-end pour application mobile, vous lappelez de façon synchrone. Le type de données de sortie sera sérialisé dans JSON.

Si vous envisagez d'appeler la fonction Lambda de manière asynchrone (en utilisant le type d'appel `Event`), le paramètre `outputType` doit être `void`. Par exemple, si vous utilisez AWS Lambda avec des sources d'événements comme Amazon S3 ou Amazon SNS, ces sources d'événements appellent la fonction Lambda en utilisant le type d'appel `Event`.

- Les paramètres `inputType` et `outputType` peuvent être les suivants :

- Types primitifs Java (par exemple, `String` ou `int`).
- Types d'événement AWS prédéfinis dans la bibliothèque `aws-lambda-java-events`.

Par exemple, `S3Event` est l'un des objets POJO prédéfinis dans la bibliothèque et qui fournit des méthodes vous permettant de lire facilement les informations à partir de l'événement Amazon S3 entrant.

- Vous pouvez également écrire votre propre classe POJO. AWS Lambda sérialise et déserialise automatiquement le code JSON d'entrée et de sortie en fonction du type POJO.

Pour plus d'informations, consultez [Types d'entrée et de sortie du gestionnaire \(Java\) \(p. 324\)](#).

- Vous pouvez ignorer l'objet `Context` provenant de la signature de la méthode de gestionnaire s'il n'est pas nécessaire. Pour en savoir plus, consultez [Objet de contexte AWS Lambda dans Java \(p. 333\)](#).

Prenons l'exemple de code Java suivant.

```
package example;  
  
import com.amazonaws.services.lambda.runtime.Context;  
import com.amazonaws.services.lambda.runtime.RequestHandler;  
  
public class Hello implements RequestHandler<Integer, String>{  
    public String myHandler(int myCount, Context context) {  
        return String.valueOf(myCount);  
    }  
}
```

Dans cet exemple, l'entrée est de type `Integer` et la sortie est de type `String`. Si vous compressez et ses dépendances, et créez la fonction Lambda, vous spécifiez `example.Hello::myHandler` (`package.class::method-reference`) comme gestionnaire.

Dans l'exemple de code Java, le premier paramètre de gestionnaire est l'entrée (`myHandler`), qui peut correspondre à des données d'événement (publiées par une source d'événement comme Amazon S3), à une entrée personnalisée que vous fournissez, par exemple un objet `Integer` (comme dans cet exemple), ou n'importe quel objet de données.

Pour en savoir plus sur la création d'une fonction Lambda avec ce code Java, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#).

Résolution de la surcharge du gestionnaire

Si votre code Java contient plusieurs méthodes avec le même nom que celui du `handler`, AWS Lambda utilise les règles suivantes pour sélectionner une méthode à appeler :

1. Sélectionnez la méthode avec le plus grand nombre de paramètres.
2. Si deux méthodes ou plus ont le même nombre de paramètres, AWS Lambda sélectionne celle qui utilise `Context` comme dernier paramètre.

Si aucune méthode n'a le paramètre `Context` ou si elles l'ont toutes, le comportement n'est pas défini.

Informations supplémentaires

Les rubriques suivantes fournissent plus d'informations sur le gestionnaire.

- Pour plus d'informations sur les types d'entrée et de sortie du gestionnaire, consultez la page [Types d'entrée et de sortie du gestionnaire \(Java\) \(p. 324\)](#).
- Pour plus d'informations sur l'utilisation des interfaces prédéfinies pour créer un gestionnaire, consultez la section [Utilisation des interfaces prédéfinies pour la création d'un gestionnaire \(Java\) \(p. 329\)](#).

Si vous implémentez ces interfaces, vous pouvez valider la signature de la méthode de gestionnaire lors de la compilation.

- Si la fonction Lambda génère une exception, AWS Lambda enregistre les métriques dans CloudWatch en indiquant qu'une erreur s'est produite. Pour plus d'informations, consultez [Erreurs de fonction AWS Lambda dans Java \(p. 338\)](#).

Types d'entrée et de sortie du gestionnaire (Java)

Lorsqu'AWS Lambda exécute la fonction Lambda, il appelle le gestionnaire. Le premier paramètre de gestionnaire est l'entrée, qui peut correspondre à des données d'événement (publiées par une source d'événement) ou à une entrée personnalisée que vous fournissez, telle qu'une chaîne ou n'importe quel objet de données personnalisé.

AWS Lambda prend en charge les types d'entrée et de sortie suivants pour un gestionnaire :

- Types Java simples (AWS Lambda prend en charge les types `String`, `Integer`, `Boolean`, `Map` et `List`)
- Type POJO (Plain Old Java Object)
- Type de flux (si vous ne souhaitez pas utiliser POJO ou si l'approche de sérialisation de Lambda ne répond pas à vos besoins, vous pouvez opter pour la mise en œuvre de flux d'octets. Pour en savoir plus, consultez [Exemple : Utilisation de flux d'entrée/sortie pour le gestionnaire \(Java\) \(p. 328\)](#).)

Type d'entrée et de sortie du gestionnaire : type String

La classe Java suivante affiche un gestionnaire appelé `myHandler` qui utilise le type `String` pour l'entrée et la sortie.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class Hello {
    public String myHandler(String name, Context context) {
        return String.format("Hello %s.", name);
    }
}
```

Vous pouvez avoir des fonctions de gestionnaire similaires pour d'autres types Java simples.

Note

Lorsque vous appelez une fonction Lambda de façon asynchrone, les valeurs de retour de cette fonction Lambda sont ignorées. Par conséquent, nous vous recommandons de définir un type de retour « void » dans votre code pour que cela soit clair. Pour en savoir plus, consultez [Invoke \(p. 452\)](#).

Pour tester un exemple complet, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#).

Type d'entrée et de sortie du gestionnaire : type POJO

La classe Java suivante affiche un gestionnaire appelé `myHandler` qui utilise le type POJO pour l'entrée et la sortie.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class HelloPojo {

    // Define two classes/POJOs for use with Lambda function.
    public static class RequestClass {
        ...
    }

    public static class ResponseClass {
        ...
    }

    public static ResponseClass myHandler(RequestClass request, Context context) {
        String greetingString = String.format("Hello %s, %s.", request.getFirstName(),
        request.getLastName());
        return new ResponseClass(greetingString);
    }
}
```

AWS Lambda sérialise les informations sur des conventions de dénomination bean standard (voir le [didacticiel Java EE 6](#)). Vous devez utiliser des objets POJO mutables avec des méthodes getter et setter publiques.

Note

Ne vous fiez pas aux autres fonctionnalités des structures de sérialisation, telles que les annotations. Si vous devez personnaliser le comportement de sérialisation, vous pouvez utiliser le flux d'octets bruts afin de définir votre propre sérialisation.

Si vous utilisez des objets POJO pour l'entrée et la sortie, vous devez fournir la mise en œuvre des types `RequestClass` et `ResponseClass`. Pour obtenir un exemple, consultez [Exemple : Utilisation de POJO pour l'entrée et la sortie du gestionnaire \(Java\) \(p. 326\)](#).

Exemple : Utilisation de POJO pour l'entrée et la sortie du gestionnaire (Java)

Supposons que les événements de votre application génèrent des données qui incluent le prénom et le nom de famille, comme illustré ci-après :

```
{ "firstName": "John", "lastName": "Doe" }
```

Pour cet exemple, le gestionnaire reçoit ce code JSON et renvoie la chaîne "Hello John Doe".

```
public static ResponseClass handleRequest(RequestClass request, Context context){
    String greetingString = String.format("Hello %s, %s.", request.firstName,
    request.lastName);
    return new ResponseClass(greetingString);
}
```

Pour créer une fonction Lambda avec ce gestionnaire, vous devez assurer la mise en œuvre des types d'entrée et de sortie, comme illustré dans l'exemple Java suivant. La classe HelloPojo définit la méthode handler.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class HelloPojo implements RequestHandler<RequestClass, ResponseClass>{

    public ResponseClass handleRequest(RequestClass request, Context context){
        String greetingString = String.format("Hello %s, %s.", request.firstName,
        request.lastName);
        return new ResponseClass(greetingString);
    }
}
```

Afin de mettre en œuvre le type d'entrée, ajoutez le code suivant dans un fichier séparé que vous appellerez RequestClass.java. Placez-le à côté de la classe HelloPojo.java dans la structure du répertoire :

```
package example;

public class RequestClass {
    String firstName;
    String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

```
public RequestClass(String firstName, String lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
}  
  
public RequestClass() {  
}  
}
```

Afin de mettre en œuvre le type de sortie, ajoutez le code suivant dans un fichier séparé que vous appellerez ResponseClass.java. Placez-le à côté de la classe HelloPojo.java dans la structure du répertoire :

```
package example;  
  
public class ResponseClass {  
    String greetings;  
  
    public String getGreetings() {  
        return greetings;  
    }  
  
    public void setGreetings(String greetings) {  
        this.greetings = greetings;  
    }  
  
    public ResponseClass(String greetings) {  
        this.greetings = greetings;  
    }  
  
    public ResponseClass() {  
    }  
}
```

Note

Les méthodes `get` et `set` sont obligatoires pour que POJO fonctionne avec l'outil de sérialisation intégré d'AWS Lambda. Les constructeurs qui n'utilisent aucun argument ne sont généralement pas requis. Toutefois, dans cet exemple, nous avons fourni d'autres constructeurs et nous devons donc indiquer explicitement les constructeurs sans argument.

Vous pouvez importer ce code en tant que fonction Lambda et le tester comme suit :

- Utilisez les fichiers de code précédents pour créer un package de déploiement.
- Importez le package de déploiement dans AWS Lambda et créez votre fonction Lambda. Pour ce faire, passez par la console ou par AWS CLI.
- Appelez la fonction Lambda manuellement à l'aide de la console ou de l'interface de ligne de commande. Vous pouvez fournir un échantillon de données d'événement JSON lorsque vousappelez manuellement la fonction Lambda. Exemples :

```
{ "firstName": "John", "lastName": "Doe" }
```

Pour en savoir plus, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#). Notez les différences suivantes :

- Lorsque vous créez un package de déploiement, n'oubliez pas la dépendance de la bibliothèque `aws-lambda-java-core`.

- Lorsque vous créez la fonction Lambda, spécifiez `example.HelloPojo::handleRequest` (`package.class::method`) comme valeur de gestionnaire.

Exemple : Utilisation de flux d'entrée/sortie pour le gestionnaire (Java)

Si vous ne souhaitez pas utiliser POJO ou si l'approche de sérialisation de Lambda ne répond pas à vos besoins, vous pouvez opter pour la mise en œuvre de flux d'octets. Dans ce cas, vous pouvez utiliser `InputStream` et `OutputStream` comme types d'entrée et de sortie pour le gestionnaire. Voici un exemple de fonction de gestionnaire :

```
public void handler(InputStream inputStream, OutputStream outputStream, Context context)
    throws IOException{
    ...
}
```

Notez que dans ce cas, la fonction du gestionnaire utilise des paramètres pour les flux de requête et de réponse.

Voici un exemple de fonction Lambda mettant en œuvre le gestionnaire qui utilise les types `InputStream` et `OutputStream` pour les paramètres `input` et `output`.

Note

La charge utile de l'entrée doit respecter un format JSON valide, mais le flux de sortie n'est pas soumis à cette restriction. Tous les octets sont pris en charge.

```
package example;

import java.io.InputStream;
import java.io.OutputStream;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler{
    public void handler(InputStream inputStream, OutputStream outputStream, Context
context) throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

Pour tester le code, procédez comme suit :

- Utilisez le code précédent pour créer un package de déploiement.
- Importez le package de déploiement dans AWS Lambda et créez votre fonction Lambda. Pour ce faire, passez par la console ou par AWS CLI.
- Vous pouvez manuellement appeler le code en fournissant un échantillon d'entrée. Exemples :

```
test
```

Suivez les instructions indiquées dans la mise en route. Pour en savoir plus, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#). Notez les différences suivantes :

- Lorsque vous créez un package de déploiement, n'oubliez pas la dépendance de la bibliothèque aws-lambda-java-core.
- Lorsque vous créez la fonction Lambda, spécifiez `example.Hello::handler` (`package.class::method`) comme valeur de gestionnaire.

Utilisation des interfaces prédéfinies pour la création d'un gestionnaire (Java)

Vous pouvez utiliser l'une des interfaces prédéfinies fournies par la bibliothèque de base AWS Lambda Java (aws-lambda-java-core) pour créer le gestionnaire de fonctions Lambda, au lieu d'écrire votre propre méthode de gestionnaire avec un nom et des paramètres arbitraires. Pour plus d'informations sur les gestionnaires, consultez la section [Gestionnaire de fonctions AWS Lambda dans Java \(p. 322\)](#).

Vous pouvez mettre en œuvre l'une des interfaces prédéfinies, `RequestStreamHandler` ou `RequestHandler`, et fournir cette mise en œuvre pour la méthode `handleRequest` que les interfaces proposent. Vous mettez en œuvre une de ces interfaces selon que vous souhaitez utiliser des types Java standard ou des types POJO personnalisés pour l'entrée ou la sortie de votre gestionnaire (où AWS Lambda sérialise et déserialise automatiquement les données d'entrée et de sortie pour correspondre à votre type de données), ou vous personnalisez la sérialisation avec le type `Stream`.

Note

Ces interfaces sont disponibles dans la bibliothèque aws-lambda-java-core.

Lorsque vous mettez en œuvre des interfaces standards, elles vous aident à valider la signature de la méthode au moment de la compilation.

Si vous mettez en œuvre l'une des interfaces, vous spécifiez `package.classe` dans le code Java en tant que gestionnaire lorsque vous créez la fonction Lambda. Par exemple, voici ci-dessous la commande `create-function` modifiée (CLI) provenant de la mise en route. Notez que le paramètre `--handler` spécifie la valeur « `example.Hello` » :

```
aws lambda create-function \
--region region \
--function-name getting-started-lambda-function-in-java \
--zip-file fileb://deployment-package (zip or jar)
  path \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler example.Hello \
--runtime java8 \
--timeout 15 \
--memory-size 512
```

Les sections suivantes fournissent des exemples de mise en œuvre de ces interfaces.

Exemple 1 : Création d'un gestionnaire avec une entrée/sortie POJO personnalisée (utilisation de l'interface RequestHandler)

L'exemple de classe `Hello` de cette section met en œuvre l'interface `RequestHandler`. Cette interface définit la méthode `handleRequest()` qui recueille les données d'événement en tant que paramètre d'entrée de type `Request` et qui renvoie un objet POJO de type `Response` :

```
public Response handleRequest(Request request, Context context) {
  ...
}
```

```
}
```

La classe Hello avec un exemple de mise en œuvre de la méthode handleRequest() est affichée. Pour cet exemple, nous supposons que les données d'événement se composent du prénom et du nom de famille.

```
package example;

import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestHandler<Request, Response> {

    public Response handleRequest(Request request, Context context) {
        String greetingString = String.format("Hello %s %s.", request.firstName,
request.lastName);
        return new Response(greetingString);
    }
}
```

Par exemple, si les données d'événement de l'objet Request sont :

```
{
    "firstName": "value1",
    "lastName" : "value2"
}
```

La méthode renvoie un objet Response comme suit :

```
{
    "greetings": "Hello value1 value2."
}
```

Ensuite, vous devez mettre en œuvre les classes Request et Response. Vous pouvez utiliser la mise en œuvre suivante pour les tests :

La classe Request :

```
package example;

public class Request {
    String firstName;
    String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

```
public Request(String firstName, String lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
}  
  
public Request() {  
}  
}
```

La classe Response :

```
package example;  
  
public class Response {  
    String greetings;  
  
    public String getGreetings() {  
        return greetings;  
    }  
  
    public void setGreetings(String greetings) {  
        this.greetings = greetings;  
    }  
  
    public Response(String greetings) {  
        this.greetings = greetings;  
    }  
  
    public Response() {  
    }  
}
```

Vous pouvez créer une fonction Lambda à partir de ce code et tester l'expérience complète, comme suit :

- Utilisez le code précédent pour créer un package de déploiement. Pour en savoir plus, consultez [Package de déploiement AWS Lambda dans Java \(p. 315\)](#)
- Importez le package de déploiement dans AWS Lambda et créez votre fonction Lambda.
- Testez la fonction Lambda à l'aide de la console ou de l'interface de ligne de commande. Vous pouvez spécifier n'importe quel échantillon de données JSON qui respecte les méthodes getter et setter dans votre classe Request, par exemple :

```
{  
    "firstName": "John",  
    "lastName" : "Doe"  
}
```

La fonction Lambda renvoie le JSON suivant en réponse.

```
{  
    "greetings": "Hello John, Doe."  
}
```

Suivez les instructions indiquées dans la mise en route (voir [Création d'une fonction Lambda en Java \(p. 342\)](#)). Notez les différences suivantes :

- Lorsque vous créez un package de déploiement, n'oubliez pas la dépendance de la bibliothèque aws-lambda-java-core.

- Lorsque vous créez la fonction Lambda, spécifiez `example.Hello` (`package.class`) comme valeur de gestionnaire.

Exemple 2 : Création de gestionnaire avec une entrée/sortie de flux (utilisation de l'interface RequestStreamHandler)

La classe `Hello` de cet exemple met en œuvre l'interface `RequestStreamHandler`. Cette interface définit la méthode `handleRequest` comme suit :

```
public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
    throws IOException {
    ...
}
```

La classe `Hello` avec un exemple de mise en œuvre du gestionnaire `handleRequest()` est affichée. Le gestionnaire traite les données d'événement entrantes (par exemple, une chaîne « Hello ») en la convertissant simplement en majuscules et en la renvoyant.

```
package example;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler {
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
        throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

Vous pouvez créer une fonction Lambda à partir de ce code et tester l'expérience complète, comme suit :

- Utilisez le code précédent pour créer le package de déploiement.
- Importez le package de déploiement dans AWS Lambda et créez votre fonction Lambda.
- Testez la fonction Lambda à l'aide de la console ou de l'interface de ligne de commande. Vous pouvez spécifier n'importe quel échantillon de données de chaîne, par exemple :

```
"test"
```

La fonction Lambda renvoie TEST en réponse.

Suivez les instructions indiquées dans la mise en route (voir [Création d'une fonction Lambda en Java \(p. 342\)](#)). Notez les différences suivantes :

- Lorsque vous créez un package de déploiement, n'oubliez pas la dépendance de la bibliothèque `aws-lambda-java-core`.

- Lorsque vous créez la fonction Lambda, spécifiez `example.Hello` ([package.class](#)) comme valeur de gestionnaire.

Objet de contexte AWS Lambda dans Java

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 322\)](#). Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'appel, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `getRemainingTimeInMillis()` – Returns the number of milliseconds left before the execution times out.
- `getFunctionName()` – Returns the name of the Lambda function.
- `getFunctionVersion()` – Returns the [version \(p. 38\)](#) of the function.
- `getInvokedFunctionArn()` – Returns the Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `getMemoryLimitInMB()` – Returns the amount of memory configured on the function.
- `getAwsRequestId()` – Returns the identifier of the invocation request.
- `getLogGroupName()` – Returns the log group for the function.
- `getLogStreamName()` – Returns the log stream for the function instance.
- `getIdentity()` – (mobile apps) Returns information about the Amazon Cognito identity that authorized the request.
- `getClientContext()` – (mobile apps) Returns the client context provided to the Lambda invoker by the client application.
- `getLogger()` – Renvoie l'[objet Logger \(p. 334\)](#) pour la fonction.

L'exemple suivant montre une fonction de gestionnaire qui consigne les informations de contexte.

Example ContextLogger.java

```
package example;
import java.io.InputStream;
import java.io.OutputStream;
import com.amazonaws.services.lambda.runtime.Context;

public class ContextLogger {
    public static void myHandler(InputStream inputStream, OutputStream outputStream,
        Context context) {
        int letter;
        try {
            while((letter = inputStream.read()) != -1)
            {
                outputStream.write(Character.toUpperCase(letter));
            }
            Thread.sleep(3000); // Intentional delay for testing the
        getRemainingTimeInMillis() result.
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        // Print info from the context object
        System.out.println("Function name: " + context.getFunctionName());
        System.out.println("Max mem allocated: " + context.getMemoryLimitInMB());
```

```
        System.out.println("Time remaining in milliseconds: " +
context.getRemainingTimeInMillis());
        System.out.println("CloudWatch log stream name: " + context.getLogStreamName());
        System.out.println("CloudWatch log group name: " + context.getLogGroupName());
    }
}
```

Dependencies

- `aws-lambda-java-core`

Build the code with the Lambda library dependencies to create a deployment package. For instructions, see [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Journalisation des fonctions AWS Lambda dans Java

Votre fonction Lambda peut contenir des instructions de journalisation. AWS Lambda écrit ces journaux dans CloudWatch. Nous vous recommandons d'utiliser l'un des éléments suivants pour écrire des journaux.

Appender personnalisé pour Log4j™ 2

AWS Lambda recommande Log4j 2 afin de fournir un appender personnalisé. Vous pouvez utiliser l'appender personnalisé Log4j (voir [Apache log4j](#)) offert par Lambda pour la journalisation à partir de vos fonctions Lambda. Chaque appel aux méthodes Log4j, comme `log.info()` ou `log.error()`, entraîne un événement CloudWatch Logs. L'appender personnalisé est appelé `LambdaAppender` et doit être utilisé dans le fichier `log4j2.xml`. Vous devez inclure l'objet `aws-lambda-java-log4j2` (`artifactId:aws-lambda-java-log4j2`) dans le package de déploiement (fichier `.jar`). Pour obtenir un exemple, consultez [Exemple 1 : Écrire des journaux via Log4J v2.8 \(p. 335\)](#).

`LambdaLogger.log()`

Chaque appel à `LambdaLogger.log()` entraîne un événement CloudWatch Logs, dans la mesure où la taille de l'événement respecte les limites autorisées. Pour obtenir des informations sur les limites de CloudWatch Logs, consultez [Limites des journaux CloudWatch](#) dans le Guide de l'utilisateur Amazon CloudWatch. Pour obtenir un exemple, consultez [Exemple 2 : Création de journaux à l'aide de LambdaLogger \(Java\) \(p. 337\)](#).

En outre, vous pouvez utiliser les instructions suivantes dans le code de la fonction Lambda pour générer des entrées de journal :

- `System.out()`
- `System.err()`

Cependant, notez qu'AWS Lambda traite chaque ligne renvoyée par `System.out` et `System.err` comme un événement distinct. Cette approche convient lorsque chaque ligne de sortie correspond à une entrée de journal unique. Lorsqu'une entrée de journal comporte plusieurs lignes de sortie, AWS Lambda tente de les analyser en utilisant les sauts de ligne pour identifier les événements distincts. Par exemple, la fonction suivante consigne les deux mots (« Hello » et « Hello ») comme deux événements séparés :

```
System.out.println("Hello \n world");
```

Comment trouver les journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Localisez les journaux dans CloudWatch Logs. L'objet `context` (dans la bibliothèque `aws-lambda-java-core`) fournit les méthodes `getLogStreamName()` et `getLogGroupName()`. Vous pouvez les utiliser pour localiser le flux spécifique dans lequel les journaux sont écrits.
- Si vousappelez une fonction Lambda via la console, le type d'appel est toujours `RequestResponse` (exécution synchrone) et la console affiche les journaux écrits par la fonction Lambda à l'aide de l'objet `LambdaLogger`. AWS Lambda renvoie également les journaux à partir des méthodes `System.out` et `System.err`.
- Si vousappelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les 4 derniers Ko de données de journalisation écrites dans CloudWatch Logs. Pour plus d'informations, consultez [Invoke \(p. 452\)](#). AWS Lambda renvoie ces informations de journalisation dans l'en-tête `x-amz-log-results` de la réponse. Si vous utilisez l'AWS Command Line Interface pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type` avec la valeur `Tail`.

Exemples de journalisation (Java)

Cette section fournit des exemples d'utilisation de l'appender personnalisé pour Log4j et des objets `LambdaLogger` pour la journalisation des informations.

Exemple 1 : Écrire des journaux via Log4J v2.8

- L'exemple suivant montre comment générer votre artefact avec Maven pour inclure correctement les plug-ins Log4j v2.8 :
 - Pour Maven pom.xml :

```
<dependencies>
  ...
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-log4j2</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.8.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.8.2</version>
  </dependency>
  ...
</dependencies>
```

- Si vous utilisez le plug-in Maven Shade, configurez-le comme suit :

```
<plugins>
  ...
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
```

```
<artifactId>maven-shade-plugin</artifactId>
<version>2.4.3</version>
<executions>
    <execution>
        <phase>package</phase>
        <goals>
            <goal>shade</goal>
        </goals>
        <configuration>
            <transformers>
                <transformer

implementation="com.github.edwgiz.mavenShadePlugin.log4j2CacheTransformer.PluginsCacheFileTransformer"
                </transformer>
            </transformers>
        </configuration>
    </execution>
</executions>
<dependencies>
    <dependency>
        <groupId>com.github.edwgiz</groupId>
        <artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
        <version>2.8.1</version>
    </dependency>
</dependencies>
</plugin>
...
</plugins>
```

- L'exemple de code Java suivant montre comment utiliser Log4j avec Lambda :

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Hello {
    // Initialize the Log4j logger.
    static final Logger logger = LogManager.getLogger(Hello.class);

    public String myHandler(String name, Context context) {
        // System.out: One log statement but with a line break (AWS Lambda writes two
        // events to CloudWatch).
        System.out.println("log data from stdout \n this is continuation of
        system.out");

        // System.err: One log statement but with a line break (AWS Lambda writes two
        // events to CloudWatch).
        System.err.println("log data from stderr. \n this is a continuation of
        system.err");

        logger.error("log data from log4j err. \n this is a continuation of
        log4j.err");

        // Return will include the log stream name so you can look
        // up the log later.
        return String.format("Hello %s. log stream = %s", name,
        context.getLogStreamName());
    }
}
```

- L'exemple précédent utilise le fichier log4j2.xml suivant pour charger les propriétés

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration packages="com.amazonaws.services.lambda.runtime.log4j2">
    <Appenders>
        <Lambda name="Lambda">
            <PatternLayout>
                <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1}:%L - %m%n</pattern>
            </PatternLayout>
        </Lambda>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Lambda" />
        </Root>
    </Loggers>
</Configuration>
```

Exemple 2 : Création de journaux à l'aide de LambdaLogger (Java)

L'exemple de code Java suivant crée des journaux via les méthodes de système et l'objet `LambdaLogger` pour illustrer leurs différences quand AWS Lambda consigne des informations dans CloudWatch.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(String name, Context context) {

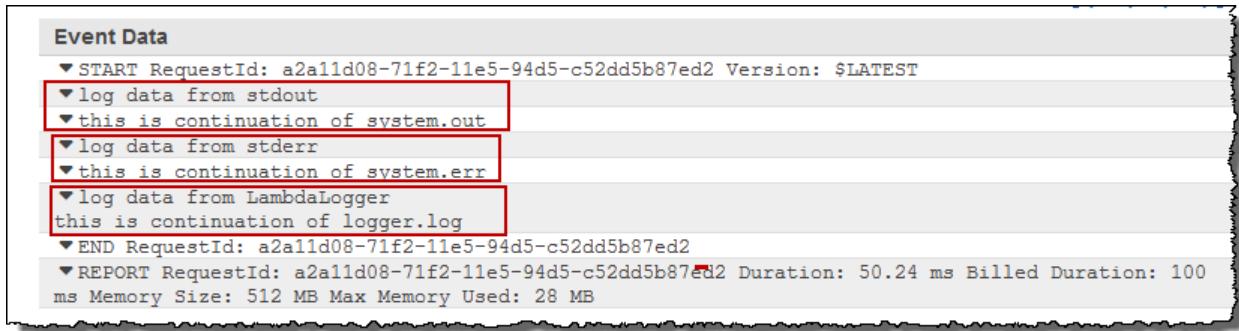
        // System.out: One log statement but with a line break (AWS Lambda writes two
        events to CloudWatch).
        System.out.println("log data from stdout \n this is continuation of system.out");

        // System.err: One log statement but with a line break (AWS Lambda writes two
        events to CloudWatch).
        System.err.println("log data from stderr \n this is continuation of system.err");

        LambdaLogger logger = context.getLogger();
        // Write log to CloudWatch using LambdaLogger.
        logger.log("log data from LambdaLogger \n this is continuation of logger.log");

        // Return will include the log stream name so you can look
        // up the log later.
        return String.format("Hello %s. log stream = %s", name,
        context.getLogStreamName());
    }
}
```

Voici un exemple d'entrées de journal dans CloudWatch Logs.



Remarque:

- AWS Lambda analyse la chaîne de journal dans chacun des journaux d'instructions `System.out.println()` et `System.err.println()` comme deux événements distincts (notez les deux flèches vers le bas dans la capture d'écran) en raison du saut de ligne.
- Le journal `LambdaLogger.log()` génère un événement CloudWatch.

Pour tester le code, procédez comme suit :

- Utilisez le code pour créer un package de déploiement.
- Importez le package de déploiement dans AWS Lambda pour créer votre fonction Lambda.
- Pour tester la fonction Lambda, utilisez une chaîne (« ceci est un test ») comme exemple d'événement. Le code du gestionnaire reçoit l'exemple d'événement, mais ne s'en sert pas. Il montre uniquement comment écrire des journaux.

Suivez les instructions indiquées dans la mise en route. Pour en savoir plus, consultez [Création d'une fonction Lambda en Java \(p. 342\)](#). Notez les différences suivantes :

- Lorsque vous créez un package de déploiement, n'oubliez pas la dépendance de la bibliothèque `aws-lambda-java-core`.
- Lorsque vous créez la fonction Lambda, spécifiez `example.Hello::myHandler` (`package.class::method`) comme valeur de gestionnaire.

Erreurs de fonction AWS Lambda dans Java

Si la fonction Lambda déclenche une exception, AWS Lambda reconnaît l'échec, puis sérialise les informations correspondantes dans JSON et les renvoie. Voici un exemple de message d'erreur :

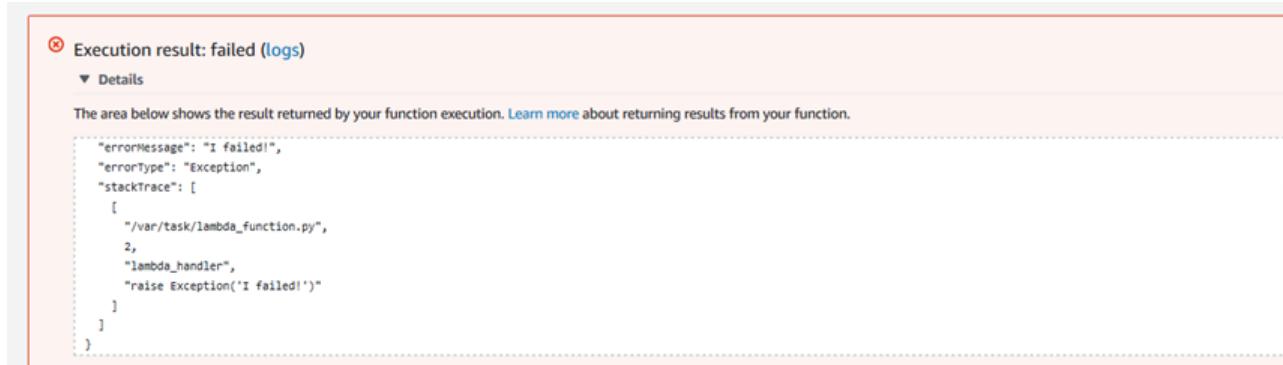
```
{
  "errorMessage": "Name John Doe is invalid. Exception occurred...",
  "errorType": "java.lang.Exception",
  "stackTrace": [
    "example.Hello.handler(Hello.java:9)",
    "sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)",
    "sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)",
    "sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)",
    "java.lang.reflect.Method.invoke(Method.java:497)"
  ]
}
```

Notez que la trace de la pile est renvoyée sous la forme du tableau JSON stackTrace constitué des éléments de cette trace.

La méthode dans laquelle vous récupérez les informations d'erreur varie selon le type d'appel que vous avez spécifié lorsque vous avez appelé la fonction :

- Type d'appel RequestResponse (exécution synchrone) : dans ce cas, vous recevez le message d'erreur.

Par exemple, si vous appelez une fonction Lambda à l'aide de la console Lambda, le type d'appel correspond toujours à RequestResponse, et la console affiche les informations d'erreur renvoyées par AWS Lambda dans la section Execution result, comme illustré dans l'image suivante.



- Type d'appel Event (exécution asynchrone) : dans ce cas, AWS Lambda ne renvoie rien. Au lieu de cela, il consigne les informations d'erreur dans CloudWatch Logs et dans les métriques CloudWatch.

En fonction de la source d'événement, AWS Lambda réessaie parfois d'exécuter la fonction Lambda qui a échoué. Par exemple, si Kinesis est la source d'événement de la fonction Lambda, AWS Lambda retente la fonction qui a échoué jusqu'à ce que la fonction Lambda aboutisse ou jusqu'à ce que les enregistrements du flux expirent.

Gestion des erreurs de fonction

Vous pouvez créer la gestion des erreurs personnalisées pour déclencher une exception directement depuis la fonction Lambda et la gérer directement (Réessayer ou CATCH) dans le cadre d'une machine d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

Prenons l'exemple d'un état CreateAccount qui est une tâche écrivant les détails d'un client dans une base de données à l'aide d'une fonction Lambda.

- Si la tâche réussit, un compte est créé et un e-mail de bienvenue est envoyé.
- Si un utilisateur essaie de créer un compte pour un nom d'utilisateur qui existe déjà, la fonction Lambda génère une erreur, ce qui pousse la machine d'état à suggérer un autre nom d'utilisateur et à recommencer le processus de création de compte.

L'extrait de code suivant montre comment procéder. Notez que les erreurs personnalisées dans Java doivent étendre la classe `Exception`.

```
package com.example;

public static class AccountAlreadyExistsException extends Exception {
    public AccountAlreadyExistsException(String message) {
        super(message);
```

```
    }
}

package com.example;

import com.amazonaws.services.lambda.runtime.Context;

public class Handler {
    public static void CreateAccount(String name, Context context) throws
        AccountAlreadyExistsException {
        throw new AccountAlreadyExistsException ("Account is in use!");
    }
}
```

Vous pouvez configurer Step Functions de façon à intercepter l'erreur à l'aide d'une règle `Catch`. Lambda définit automatiquement le nom de l'erreur pour le nom entièrement qualifié de l'exception lors de l'exécution :

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": [ "com.example.AccountAlreadyExistsException" ],
                    "Next": "SuggestAccountName"
                }
            ]
        },
        ...
    }
}
```

Lors de l'exécution, AWS Step Functions identifie l'erreur et [transitionne](#) jusqu'à l'état `SuggestAccountName` comme spécifié dans la transition `Next`.

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda, ce qui vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Instrumentation du code Java dans AWS Lambda

Dans Java, Lambda peut émettre des sous-segments vers X-Ray afin de vous montrer des informations sur les appels en aval effectués par votre fonction vers d'autres services AWS. Pour profiter de cette fonctionnalité, incluez le [kit SDK AWS X-Ray pour Java](#) dans votre package de déploiement. Aucune modification du code n'est nécessaire. Tant que vous utilisez un AWS SDK version 1.11.48 ou ultérieure, il n'est pas nécessaire d'ajouter des lignes de code pour effectuer le suivi des appels en aval provenant de votre fonction.

AWS SDK importera de façon dynamique le SDK X-Ray afin d'émettre des sous-segments pour les appels en aval effectués par votre fonction. Le kit SDK X-Ray pour Java vous permet d'équiper votre code afin d'émettre des sous-segments personnalisés et/ou d'ajouter des annotations à vos segments X-Ray.

L'exemple suivant utilise le kit SDK X-Ray pour Java afin de permettre à une fonction Lambda d'émettre un sous-segment personnalisé et d'envoyer des annotations personnalisées à X-Ray :

```
package uptime;

import java.io.IOException;
import java.time.Instant;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;

public class Hello {
    private static final Log logger = LogFactory.getLog(Hello.class);

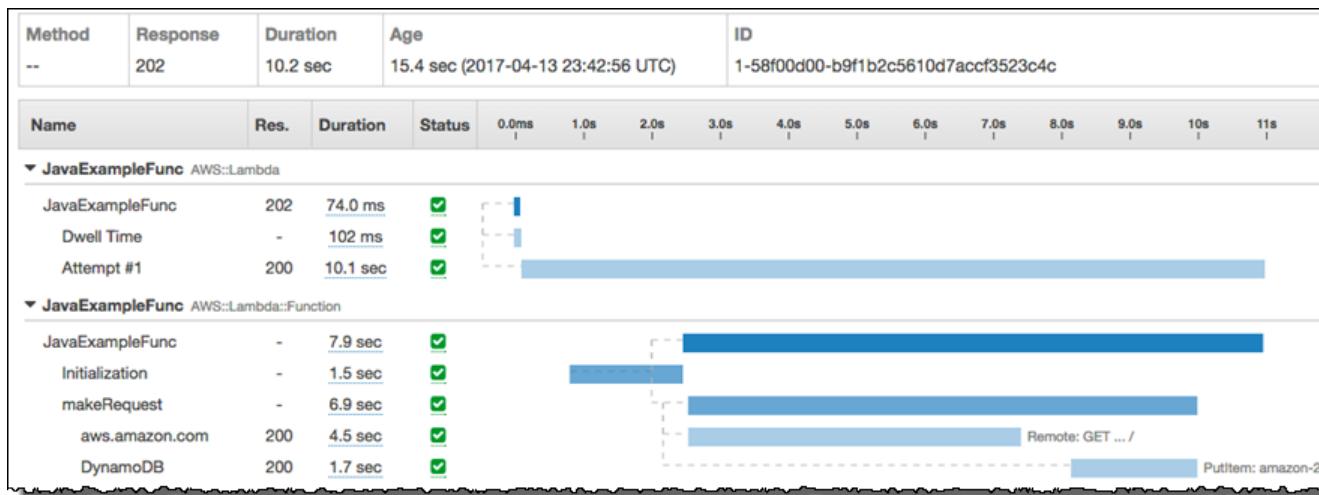
    private static final AmazonDynamoDB dynamoClient;
    private static final HttpClient httpClient;

    static {
        dynamoClient =
AmazonDynamoDBClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
        httpClient = HttpClientBuilder.create().build();
    }
    public void checkUptime(Context context) {
        AWSXRay.createSubsegment("makeRequest", (subsegment) -> {

            HttpGet request = new HttpGet("https://aws.amazon.com/");
            boolean is2xx = false;

            try {
                HttpResponse response = httpClient.execute(request);
                is2xx = (response.getStatusLine().getStatusCode() / 100) == 2;
                subsegment.putAnnotation("statusCode",
response.getStatusLine().getStatusCode());
            } catch (IOException ioe) {
                logger.error(ioe);
            }
            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Timestamp", new AttributeValue().withN("" +
Instant.now().getEpochSecond()));
            item.put("2xx", new AttributeValue().withBOOL(is2xx));
            dynamoClient.putItem("amazon-2xx", item);
        });
    }
}
```

Voici un exemple de suivi émis par le code précédent (appel synchrone) :



Création d'une fonction Lambda en Java

Les plans fournissent un exemple de code créé dans Python ou Node.js. Vous pouvez facilement modifier l'exemple à l'aide de l'éditeur intégré dans la console. Toutefois, si vous souhaitez créer le code de la fonction Lambda en Java, aucun plan n'est fourni à cet effet. En outre, aucun éditeur intégré ne vous permet d'écrire du code Java dans la console AWS Lambda.

Autrement dit, vous devez écrire le code Java et créer le package de déploiement en dehors de la console. Une fois que vous avez généré le package de déploiement, vous pouvez utiliser la console pour l'importer dans AWS Lambda afin de créer la fonction Lambda. Vous pouvez également utiliser la console pour tester la fonction en l'appelant manuellement.

Dans cette section, vous créez une fonction Lambda via l'exemple de code Java suivant.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(int myCount, Context context) {
        LambdaLogger logger = context.getLogger();
        logger.log("received : " + myCount);
        return String.valueOf(myCount);
    }
}
```

Le modèle de programmation explique comment écrire le code Java en détail, notamment les types d'entrée et de sortie pris en charge par AWS Lambda. Pour plus d'informations sur le modèle de programmation, consultez la section [Création de fonctions Lambda avec Java \(p. 314\)](#). Pour l'instant, notez les points suivants concernant ce code :

- Lorsque vous compressez et importez ce code pour créer la fonction Lambda, vous spécifiez la référence de méthode `example.Hello::myHandler` en tant que gestionnaire.
- Le gestionnaire de cet exemple utilise le type d'entrée `int` et le type de sortie `String`.

AWS Lambda prend en charge l'entrée et la sortie des types sérialisables JSON et des types `InputStream/OutputStream`. Lorsque vous appelez cette fonction, vous transmettez un exemple de valeur « int » (par exemple, 123).

- Vous pouvez utiliser la console Lambda pour appeler manuellement cette fonction Lambda. La console utilise toujours le type d'appel `RequestResponse` (synchrone). Par conséquent, vous voyez la réponse dans la console.
- Le gestionnaire inclut le paramètre `Context` facultatif. Dans le code, nous utilisons le `LambdaLogger` fourni par l'objet `Context` pour écrire des entrées dans les journaux CloudWatch. Pour plus d'informations sur l'utilisation de l'objet `Context`, consultez la section [Objet de contexte AWS Lambda dans Java \(p. 333\)](#).

Tout d'abord, vous devez compresser ce code et toutes les dépendances dans un package de déploiement. Ensuite, vous pouvez utiliser l'exercice de mise en route pour importer le package afin de créer la fonction Lambda, puis la tester à l'aide de la console. Pour plus d'informations sur la création d'un package de déploiement, consultez [Package de déploiement AWS Lambda dans Java \(p. 315\)](#).

Création de fonctions Lambda avec Go

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lors de la création du code d'une fonction Lambda dans [Go](#).

Runtimes Go

Nom	Identificateur
Go 1.x	go1.x

Rubriques

- [Package de déploiement AWS Lambda dans Go \(p. 344\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Go \(p. 345\)](#)
- [Objet de contexte AWS Lambda dans Go \(p. 349\)](#)
- [Journalisation des fonctions AWS Lambda dans Go \(p. 351\)](#)
- [Erreurs de fonction AWS Lambda dans Go \(p. 352\)](#)
- [Instrumentation du code Go dans AWS Lambda \(p. 354\)](#)
- [Utilisation des variables d'environnement \(p. 355\)](#)

En outre, notez qu'AWS Lambda fournit les éléments suivants :

- [github.com/aws/aws-lambda-go/lambda](#) : la mise en œuvre du modèle de programmation Lambda pour Go. Ce package est utilisé par AWS Lambda pour appeler votre [Gestionnaire de fonctions AWS Lambda dans Go \(p. 345\)](#).
- [github.com/aws/aws-lambda-go/lambdacontext](#) : assistants pour accéder aux informations du contexte d'exécution depuis le [Objet de contexte AWS Lambda dans Go \(p. 349\)](#).
- [github.com/aws/aws-lambda-go/events](#) : cette bibliothèque fournit les définitions de type pour les intégrations de sources d'événements communes.

Package de déploiement AWS Lambda dans Go

Pour créer une fonction Lambda, vous devez d'abord concevoir un package de déploiement de cette fonction, à savoir un fichier .zip qui se compose de votre code et de toutes les dépendances.

Une fois que vous avez généré un package de déploiement, vous pouvez soit la charger directement, soit charger le fichier .zip dans un compartiment Amazon S3 de la région AWS dans laquelle vous voulez créer la fonction Lambda, puis spécifier le nom de compartiment et le nom de la clé d'objet lors de la création de la fonction Lambda via la console ou l'interface de ligne de commande AWS.

Pour les fonctions Lambda écrites en Go, téléchargez la bibliothèque Lambda pour Go en vous rendant sur le répertoire d'exécution GO, puis entrez la commande suivante : `go get github.com/aws/aws-lambda-go/lambda`

Ensuite, utilisez la commande suivante pour créer, configurer et déployer une fonction Lambda Go via l'interface de ligne de commande. Notez que votre `function-name` doit correspondre au nom de votre [gestionnaire Lambda](#).

```
GOOS=linux go build lambda_handler.go
zip handler.zip ./lambda_handler
# --handler is the path to the executable inside the .zip
aws lambda create-function \
--region region \
--function-name lambda-handler \
--memory 128 \
--role arn:aws:iam::account-id:role/execution_role \
--runtime go1.x \
--zip-file fileb://path-to-your-zip-file/handler.zip \
--handler lambda-handler
```

Note

Si vous utilisez un environnement différent de Linux, comme Windows ou macOS, assurez-vous que la fonction de votre gestionnaire est compatible avec le [contexte d'exécution](#) de la fonction Lambda en définissant la variable d'environnement GOOS du système d'exploitation (système d'exploitation de Go) sur 'linux' lors de la compilation du code de la fonction de votre gestionnaire.

Création d'un package de déploiement sous Windows

Pour créer un fichier .zip qui fonctionne sur AWS Lambda avec Windows, nous vous recommandons d'installer l'outil build-lambda-zip.

Note

Si vous ne l'avez pas déjà fait, vous devrez installer [git](#), puis ajouter l'exécutable git à votre variable d'environnement Windows %PATH%.

Pour télécharger l'outil, exécutez la commande suivante :

```
go.exe get -u github.com/aws/aws-lambda-go/cmd/build-lambda-zip
```

Utilisez l'outil depuis votre GOPATH. Si vous disposez d'une installation de Go par défaut, l'outil est généralement dans le répertoire %USERPROFILE%\Go\bin. Dans le cas contraire, accédez à l'endroit où vous avez installé le runtime Go et procédez comme suit :

Dans cmd.exe, exécutez la commande suivante :

```
set GOOS=linux
go build -o main main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o main.zip main
```

Dans Powershell, exécutez la commande suivante :

```
$env:GOOS = "linux"
go build -o main main.go
~\Go\Bin\build-lambda-zip.exe -o main.zip main
```

Gestionnaire de fonctions AWS Lambda dans Go

Une fonction Lambda écrite en [Go](#) est créée en tant qu'exécutable Go. Dans le code de votre fonction Lambda, vous devez inclure le package github.com/aws/aws-lambda-go/lambda, qui met en œuvre le

modèle de programmation Lambda pour Go. De plus, vous devez mettre en œuvre le code de fonction du gestionnaire et une fonction `main()`.

```
package main

import (
    "fmt"
    "context"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(ctx context.Context, name MyEvent) (string, error) {
    return fmt.Sprintf("Hello %s!", name.Name), nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

Notez bien ce qui suit :

- package `main` : dans Go, le package contenant `func main()` doit toujours être nommé `main`.
- import : utilisez cette instruction pour inclure les bibliothèques exigées par votre fonction Lambda. Dans cette instance, elle inclut :
 - `context` : [Objet de contexte AWS Lambda dans Go \(p. 349\)](#).
 - `fmt` : l'objet Go [Formatting](#) utilisé pour formater la valeur de retour de votre fonction.
 - `github.com/aws/aws-lambda-go/lambda` : comme mentionné précédemment, implémente le modèle de programmation Lambda pour Go.
- `func HandleRequest(ctx context.Context, name MyEvent) (string, error)` : il s'agit de votre signature de gestionnaire Lambda, qui inclut le code qui sera exécuté. En outre, les paramètres inclus désignent les éléments suivants :
 - `ctx context.Context` : fournit des informations d'exécution pour l'appel de votre fonction Lambda. `ctx` est la variable que vous déclarez pour exploiter les informations disponibles par le biais de [Objet de contexte AWS Lambda dans Go \(p. 349\)](#).
 - `name MyEvent` : un type d'entrée avec un nom de variable `name` dont la valeur est renvoyée dans l'instruction `return`.
 - `string, error` : renvoie les informations d'[erreur](#) standard. Pour plus d'informations sur la gestion des erreurs personnalisées, consultez [Erreurs de fonction AWS Lambda dans Go \(p. 352\)](#).
 - `return fmt.Sprintf("Hello %s!", name), nil` : renvoie simplement une salutation formatée « Hello » avec le nom que vous avez fourni dans la signature du gestionnaire. `nil` indique qu'il n'y a pas d'erreurs et que la fonction a été exécutée avec succès.
- `func main()` : le point d'entrée qui exécute le code de votre fonction Lambda. C'est obligatoire.

En ajoutant `lambda.Start(HandleRequest)` entre les accolades du code `func main(){}`, votre fonction Lambda sera exécutée.

Note

Selon les normes du langage Go, l'accolade ouvrante, `{`, doit être placée directement à la fin de la signature de la fonction `main`.

Gestionnaire de fonction Lambda à l'aide des types structurés

Dans l'exemple ci-dessus, le type d'entrée était une chaîne simple. Mais vous pouvez également transmettre des événements structurés à votre gestionnaire de fonction :

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string `json:"What is your name?"`
    Age int     `json:"How old are you?"`
}

type MyResponse struct {
    Message string `json:"Answer:"`
}

func HandleLambdaEvent(event MyEvent) (MyResponse, error) {
    return MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,
event.Age)}, nil
}

func main() {
    lambda.Start(HandleLambdaEvent)
}
```

Votre demande doit se présenter comme suit :

```
# request
{
    "What is your name?": "Jim",
    "How old are you?": 33
}
```

Et la réponse :

```
# response
{
    "Answer": "Jim is 33 years old!"}
```

Pour plus d'informations sur la gestion des événements à partir de sources d'événements AWS, consultez [aws-lambda-go/events](#).

Signatures de gestionnaire valides

Vous disposez de plusieurs options lorsque vous créez un gestionnaire de fonction Lambda dans Go, mais vous devez respecter les règles suivantes :

- Le gestionnaire doit être une fonction.

- Le gestionnaire peut accepter entre 0 et 2 arguments. S'il y a deux arguments, le premier argument doit implémenter `context.Context`.
- Le gestionnaire peut retourner entre 0 et 2 arguments. S'il y a une seule valeur renvoyée, elle doit implémenter `error`. S'il y a deux valeurs renvoyées, la seconde valeur doit implémenter `error`. Pour plus d'informations sur l'implémentation des informations de gestion des erreurs, consultez [Erreurs de fonction AWS Lambda dans Go \(p. 352\)](#).

Voici la liste des signatures de gestionnaire valides. `TIn` et `TOut` représentent des types compatibles avec la bibliothèque `encoding/json` standard. Pour en savoir plus, reportez-vous à la section [func Unmarshal](#) pour savoir comment ces types sont déserialisés.

- `func ()`
- `func () error`
- `func (TIn), error`
- `func () (TOut, error)`
- `func (context.Context) error`
- `func (context.Context, TIn) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) (TOut, error)`

Utilisation de l'état global

Vous pouvez déclarer et modifier les variables globales indépendantes du code de votre gestionnaire de fonction Lambda. De plus, votre gestionnaire peut déclarer une fonction `init` qui est exécutée lorsque votre gestionnaire est chargé. Celui-ci se comporte dans AWS Lambda comme il le fait dans les programmes Go standard. Une instance unique de votre fonction Lambda ne gère jamais plusieurs événements simultanément. Cela signifie, par exemple, que vous pouvez modifier l'état global en toute sécurité, car vous êtes assuré que ces modifications exigent un nouveau contexte d'exécution et qu'ils n'introduiront pas de comportement de blocage ou d'instabilité à partir des appels de fonction adressés au précédent contexte d'exécution. Pour en savoir plus, consultez les éléments suivants :

- [Contexte d'exécution d'AWS Lambda \(p. 105\)](#)
- [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 283\)](#)

```
package main

import (
    "log"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/aws"
)
var invokeCount = 0
```

```
var myObjects []*s3.Object
func init() {
    svc := s3.New(session.New())
    input := &s3.ListObjectsV2Input{
        Bucket: aws.String("examplebucket"),
    }
    result, _ := svc.ListObjectsV2(input)
    myObjects = result.Contents
}

func LambdaHandler() (int, error) {
    invokeCount = invokeCount + 1
    log.Print(myObjects)
    return invokeCount, nil
}

func main() {
    lambda.Start(LambdaHandler)
}
```

Objet de contexte AWS Lambda dans Go

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 345\)](#). Cet objet fournit des méthodes et des propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

La bibliothèque de contexte Lambda fournit les variables globales, méthodes et propriétés suivantes.

Variables globales

- `FunctionName` – The name of the Lambda function.
- `FunctionVersion` – The [version \(p. 38\)](#) of the function.
- `MemoryLimitInMB` – The amount of memory configured on the function.
- `LogGroupName` – The log group for the function.
- `LogStreamName` – The log stream for the function instance.

Méthodes de contexte

- `Deadline` – Returns the date that the execution times out, in Unix time milliseconds.

Propriétés du contexte

- `InvokedFunctionArn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `AwsRequestId` – The identifier of the invocation request.
- `Identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `ClientContext` – (mobile apps) Client context provided to the Lambda invoker by the client application.

Accès aux informations du contexte d'appel

Les fonctions Lambda ont accès aux métadonnées sur leur environnement et la demande d'appel. Elles sont accessibles à l'adresse du [contexte du package](#). Si votre gestionnaire inclut `context.Context`

en tant que paramètre, Lambda insère les informations sur votre fonction dans la propriété `Value` du contexte. Notez que vous devez importer la bibliothèque `lambdacontext` pour accéder au contenu de l'objet `context.Context`.

```
package main

import (
    "context"
    "log"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/lambdacontext"
)

func CognitoHandler(ctx context.Context) {
    lc, _ := lambdacontext.FromContext(ctx)
    log.Print(lc.Identity.CognitoPoolID)
}

func main() {
    lambda.Start(CognitoHandler)
}
```

Dans l'exemple ci-dessus, `lc` est la variable utilisée pour consommer les informations que l'objet de contexte a capturées et `log.Print(lc.Identity.CognitoPoolID)` affiche ces informations, dans ce cas, le `CognitoPoolID`.

Surveillance de la durée d'exécution d'une fonction

L'exemple suivant présente la façon d'utiliser l'objet de contexte pour surveiller le temps nécessaire à l'exécution de votre fonction Lambda. Cela vous permet d'analyser les attentes de performance et d'ajuster le code de votre fonction en conséquence, si nécessaire.

```
package main

import (
    "context"
    "log"
    "time"
    "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

    deadline, _ := ctx.Deadline()
    deadline = deadline.Add(-100 * time.Millisecond)
    timeoutChannel := time.After(time.Until(deadline))

    for {

        select {

        case <- timeoutChannel:
            return "Finished before timing out.", nil

        default:
            log.Print("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
```

```
    lambda.Start(LongRunningHandler)
}
```

Journalisation des fonctions AWS Lambda dans Go

Votre fonction Lambda peut contenir des instructions de journalisation. AWS Lambda écrit ces journaux dans CloudWatch. Si vous utilisez la console Lambda pour appeler la fonction Lambda, la console affiche les mêmes journaux.

Considérons l'exemple suivant.

```
package main

import (
    "log"
    "github.com/aws/aws-lambda-go/lambda"
)

func HandleRequest() {
    log.Print("Hello from Lambda")
}

func main() {
    lambda.Start(HandleRequest)
}
```

Par l'importation du module `log`, Lambda écrit des informations de journalisation supplémentaires, telles que la date et l'heure.

Vous pouvez aussi analyser les journaux dans CloudWatch. Pour en savoir plus, consultez [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 227\)](#).

Au lieu d'utiliser le module `log`, vous pouvez utiliser les instructions `print` dans votre code, comme illustré ci-après :

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

func HandleRequest() {
    fmt.Print("Hello from Lambda")
}

func main() {
    lambda.Start(HandleRequest)
}
```

Dans ce cas, seul le texte transmis à la méthode d'impression est envoyé à CloudWatch. Les entrées de journal n'ont pas d'informations supplémentaires renvoyées par la fonction `log.Print`. De plus, n'importe quel enregistreur d'événements qui écrit sur `stdout` ou `stderr` s'intègrera en toute transparence à une fonction Go et ces journaux seront automatiquement envoyés aux journaux CloudWatch.

La console utilise le type d'appel `RequestResponse` (appel synchrone) lors de l'appel de la fonction. Par conséquent, il obtient la valeur de retour (« Hello from Lambda! ») de AWS Lambda.

Recherche des journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Dans la console AWS Lambda : la section Sortie de journal de la console AWS Lambda affiche les journaux.
- Dans l'en-tête de la réponse, lorsque vous appelez une fonction Lambda par programmation : si vousappelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les quatre derniers kilooctets de données de journal écrits dans les journaux CloudWatch. AWS Lambda renvoie ces informations de journal dans l'en-tête `x-amz-log-results` de la réponse. Pour en savoir plus, consultez [Invoke \(p. 452\)](#).

Si vous utilisez AWS CLI pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type parameter` avec la valeur `Tail` pour récupérer les mêmes informations.

- Dans CloudWatch Logs : pour trouver vos journaux dans CloudWatch, vous devez connaître le nom du groupe de journaux, ainsi que le nom de leur flux. Vous pouvez utiliser les variables globales `context.logGroupName` et `context.logStreamName` dans la bibliothèque [Objet de contexte AWS Lambda dans Go \(p. 351\)](#) pour obtenir ces informations. Lorsque vous exécutez la fonction Lambda, les journaux générés dans la console ou l'interface de ligne de commande indiquent le nom du groupe de journaux et le nom de leur flux.

Erreurs de fonction AWS Lambda dans Go

Vous pouvez créer une gestion des erreurs personnalisée pour lever une exception directement depuis votre fonction Lambda et la gérer directement.

L'extrait de code suivant montre comment procéder. Notez que les erreurs personnalisées dans Go doivent importer le module `errors`.

```
package main

import (
    "errors"
    "github.com/aws/aws-lambda-go/lambda"
)

func OnlyErrors() error {
    return errors.New("something went wrong!")
}

func main() {
    lambda.Start(OnlyErrors)
}
```

Ce qui renvoie :

```
{  
    "errorMessage": "something went wrong!"  
}
```

Gestion des erreurs de fonction

Vous pouvez créer la gestion des erreurs personnalisées pour déclencher une exception directement depuis la fonction Lambda et la gérer directement (Réessayer ou CATCH) dans le cadre d'une machine

d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

Prenons l'exemple d'un état `CreateAccount` qui est une tâche écrivant les détails d'un client dans une base de données à l'aide d'une fonction Lambda.

- Si la tâche réussit, un compte est créé et un e-mail de bienvenue est envoyé.
- Si un utilisateur essaie de créer un compte pour un nom d'utilisateur qui existe déjà, la fonction Lambda génère une erreur, ce qui pousse la machine d'état à suggérer un autre nom d'utilisateur et à recommencer le processus de création de compte.

L'extrait de code suivant montre comment procéder.

```
package main

type CustomError struct {}

func (e *CustomError) Error() string {
    return "bad stuff happened..."
}

func MyHandler() (string, error) {
    return "", &CustomError{}
}
```

Lors de l'exécution, AWS Step Functions identifie l'erreur et [transitionne](#) jusqu'à l'état `SuggestAccountName` comme spécifié dans la transition `Next`.

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda, ce qui vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Gestion des erreurs inattendues

Les fonctions Lambda peuvent échouer pour des raisons indépendantes de votre volonté, comme les pannes réseau. Il s'agit de circonstances exceptionnelles. Dans Go, `panic` répond à ces problèmes. Si votre code panique, Lambda tentera de capturer l'erreur et de la sérialiser selon le format d'erreur standard json. Lambda essaiera également d'insérer la valeur de la panique dans les journaux CloudWatch de la fonction. Après le retour de la réponse, Lambda va recréer la fonction automatiquement. Si nécessaire, vous pouvez inclure la fonction `panic` dans votre code pour personnaliser la réponse d'erreur.

```
package main

import (
    "errors"

    "github.com/aws/aws-lambda-go/lambda"
)

func handler(string) (string, error) {
    panic(errors.New("Something went wrong"))
}

func main() {
    lambda.Start(handler)
}
```

```
}
```

Ce que renvoie la pile suivante dans json :

```
{
    "errorMessage": "Something went wrong",
    "errorType": "errorString",
    "stackTrace": [
        {
            "path": "github.com/aws/aws-lambda-go/lambda/function.go",
            "line": 27,
            "label": "(*Function).Invoke.function"
        },
        ...
    ]
}
```

Instrumentation du code Go dans AWS Lambda

Vous pouvez utiliser le [kit SDK X-Ray pour Go](#) avec votre fonction Lambda. Si votre gestionnaire inclut [Objet de contexte AWS Lambda dans Go \(p. 349\)](#) comme premier argument, cet objet peut être transmis au kit SDK X-Ray. Lambda transmet les valeurs par le biais de ce contexte que le kit SDK peut utiliser pour attacher des sous-segments au segment du service d'appel Lambda. Les sous-segments créés avec le kit SDK apparaissent comme partie intégrante de vos traces Lambda.

Installation du kit SDK X-Ray pour Go

Utilisez la commande suivante pour installer le kit SDK X-Ray pour Go. (Les dépendances du kit SDK extérieures au test seront incluses).

```
go get -u github.com/aws/aws-xray-sdk-go/...
```

Si vous souhaitez inclure les dépendances de test, utilisez la commande suivante :

```
go get -u -t github.com/aws/aws-xray-sdk-go/...
```

Vous pouvez également utiliser [Glide](#) pour gérer les dépendances.

```
glide install
```

Configuration du kit SDK X-Ray pour Go

L'exemple de code suivant montre comment configurer le kit SDK X-Ray pour Go dans votre fonction Lambda :

```
import (
    "github.com/aws/aws-xray-sdk-go/xray"
)
func myHandlerFunction(ctx context.Context, sample string) {
    xray.Configure(xray.Config{
        LogLevel:      "info",           // default
        ServiceVersion: "1.2.3",
    })
}
```

```
    ... //remaining handler code  
}
```

Création d'un sous-segment

Le code suivant illustre comment démarrer un sous-segment :

```
// Start a subsegment  
ctx, subSeg := xray.BeginSubsegment(ctx, "subsegment-name")  
// ...  
// Add metadata or annotation here if necessary  
// ...  
subSeg.Close(nil)
```

Capture

Le code suivant illustre comment tracer et capturer un chemin de code critique :

```
func criticalSection(ctx context.Context) {  
    // This example traces a critical code path using a custom subsegment  
    xray.Capture(ctx, "MyService.criticalSection", func(ctx1 context.Context) error {  
        var err error  
  
        section.Lock()  
        result := someLockedResource.Go()  
        section.Unlock()  
  
        xray.AddMetadata(ctx1, "ResourceResult", result)  
    })  
}
```

Suivi des demandes HTTP

Vous pouvez également utiliser la méthode `xray.Client()` si vous souhaitez suivre un client HTTP, comme indiqué ci-dessous :

```
func myFunction (ctx context.Context) ([]byte, error) {  
    resp, err := ctxhttp.Get(ctx, xray.Client(nil), "https://aws.amazon.com")  
    if err != nil {  
        return nil, err  
    }  
    return ioutil.ReadAll(resp.Body), nil  
}
```

Utilisation des variables d'environnement

Pour accéder à [Variables d'environnement AWS Lambda \(p. 30\)](#) dans Go, utilisez la fonction `Getenv`.

L'exemple suivant illustre la marche à suivre. Notez que la fonction importe le package `fmt` pour mettre en forme les résultats affichés et le package `os`, interface système indépendante de la plateforme qui vous permet d'accéder aux variables d'environnement.

```
package main
```

```
import (
    "fmt"
    "os"
    "github.com/aws/aws-lambda-go/lambda"
)

func main() {
    fmt.Printf("%s is %s years old\n", os.Getenv("NAME"), os.Getenv("AGE"))
}
```

Lambda configure les variables d'environnement suivantes par défaut : [Variables d'environnement disponibles pour les fonctions Lambda \(p. 104\)](#).

Création de fonctions Lambda avec C#

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lors de la création du code d'une fonction Lambda en C#.

Runtimes .NET

Nom	Identificateur	Langages
.NET Core 2.1	dotnetcore2.1	C#
		PowerShell Core 6.0
.NET Core 2.0	dotnetcore2.0	C#
.NET Core 1.0	dotnetcore1.0	C#

Rubriques

- [Package de déploiement AWS Lambda dans C# \(p. 358\)](#)
- [Gestionnaire de fonctions AWS Lambda dans C# \(p. 367\)](#)
- [Objet de contexte AWS Lambda dans C# \(p. 371\)](#)
- [Journalisation des fonctions AWS Lambda dans C# \(p. 372\)](#)
- [Erreurs de fonction AWS Lambda dans C# \(p. 373\)](#)

En outre, notez qu'AWS Lambda fournit les éléments suivants :

- Amazon.Lambda.Core – Cette bibliothèque fournit un enregistreur d'événements Lambda statique, des interfaces de sérialisation et un objet de contexte. L'objet [Context \(Objet de contexte AWS Lambda dans C# \(p. 371\)\)](#) fournit les informations d'exécution concernant votre fonction Lambda.
- Amazon.Lambda.Serialization.Json – Il s'agit d'une implémentation de l'interface de sérialisation dans Amazon.Lambda.Core.
- Amazon.Lambda.Logging.AspNetCore – Fournit une bibliothèque pour la journalisation à partir d'ASP.NET.
- Objets d'événement (POCO) pour plusieurs services AWS, y compris :
 - Amazon.Lambda.APIGatewayEvents
 - Amazon.Lambda.CognitoEvents
 - Amazon.Lambda.ConfigEvents
 - Amazon.Lambda.DynamoDBEvents
 - Amazon.Lambda.KinesisEvents
 - Amazon.Lambda.S3Events
 - Amazon.Lambda.SQSEvents
 - Amazon.Lambda.SNSEvents

Ces packages sont disponibles dans [Packages NuGet](#).

Package de déploiement AWS Lambda dans C#

Un .package de déploiement Lambda .NET Core est un fichier zip de l'assembly compilé de votre fonction, avec l'ensemble de ses dépendances d'assembly. Le package contient également un fichier `proj.deps.json`. Il indique au runtime .NET Core l'ensemble des dépendances de votre fonction et un fichier `proj.runtimeconfig.json`, qui est utilisée pour configurer le runtime .NET Core. La commande `publish` de l'interface de ligne de commande .NET peut créer un dossier avec tous ces fichiers, mais par défaut `proj.runtimeconfig.json` ne sera pas inclus, car un projet Lambda est généralement configuré comme bibliothèque de classe. Pour imposer que `proj.runtimeconfig.json` soit écrit dans le cadre du processus `publish`, transmettez l'argument de ligne de commande : /`p:GenerateRuntimeConfigurationFiles=true` to the `publish` command.

Note

Bien qu'il soit possible de créer le package de déploiement avec la commande `dotnet publish`, nous vous suggérons de créer le package de déploiement avec la commande [AWS Toolkit for Visual Studio \(p. 365\)](#) ou [Interface de ligne de commande .NET Core \(p. 358\)](#). Ces outils sont optimisés spécifiquement pour Lambda afin de garantir que le fichier `lambda-project.runtimeconfig.json` existe et optimise le groupe de package, y compris la suppression de toutes les dépendances autres que celles basées sur Linux.

Interface de ligne de commande .NET Core

L'interface de ligne de commande .NET offre une inter-plateforme qui vous permet de créer des applications Lambda basées sur .NET. Cette section suppose que vous avez installé l'interface de ligne de commande .NET Core. Si tel n'est pas le cas, vous pouvez l'installer [ici](#).

Dans l'interface de ligne de commande .NET, vous utilisez la commande `new` pour créer des projets .NET à partir d'une ligne de commande. Ceci est particulièrement utile si vous souhaitez créer un projet indépendant de la plateforme en dehors de Visual Studio. Pour afficher la liste des types de projets disponibles, ouvrez une ligne de commande et accédez à l'emplacement où vous avez installé le runtime .NET Core et saisissez ce qui suit :

```
dotnet new -all
```

Vous devez voir ce qui suit :

```
dotnet new -all
Usage: new [options]

Options:
  -h, --help           Displays help for this command.
  -l, --list            Lists templates containing the specified name. If no name is
                        specified, lists all templates.
  -n, --name            The name for the output being created. If no name is specified, the
                        name of the current directory is used.
  -o, --output          Location to place the generated output.
  -i, --install         Installs a source or a template pack.
  -u, --uninstall       Uninstalls a source or a template pack.
  --nuget-source        Specifies a NuGet source to use during install.
  --type                Filters templates based on available types. Predefined values are
                        "project", "item" or "other".
  --force               Forces content to be generated even if it would change existing
                        files.
  -lang, --language     Filters templates based on language and specifies the language of the
                        template to create.
```

Templates	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	
Common/Console			
Class library	classlib	[C#], F#, VB	
Common/Library			
Unit Test Project	mstest	[C#], F#, VB	
Test/MSTest			
xUnit Test Project	xunit	[C#], F#, VB	
Test/xUnit			
Razor Page	page	[C#]	Web/
ASP.NET			
MVC ViewImports	viewimports	[C#]	Web/
ASP.NET			
MVC ViewStart	viewstart	[C#]	Web/
ASP.NET			
ASP.NET Core Empty	web	[C#], F#	Web/
Empty			
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/
MVC			
ASP.NET Core Web App	razor	[C#]	Web/
MVC/Razor Pages			
ASP.NET Core with Angular	angular	[C#]	Web/
MVC/SPA			
ASP.NET Core with React.js	react	[C#]	Web/
MVC/SPA			
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/
MVC/SPA			
Razor Class Library	razorclasslib	[C#]	Web/
Razor/Library/Razor Class Library			
ASP.NET Core Web API	webapi	[C#], F#	Web/
WebAPI			
global.json file	globaljson		
Config			
NuGet Config	nugetconfig		
Config			
Web Config	webconfig		
Config			
Solution File	sln		
Solution			
Examples:			
dotnet new mvc --auth Individual			
dotnet new viewstart			
dotnet new --help			

Donc, par exemple, pour créer une console de projet, vous devez effectuer les actions suivantes :

1. Créez un répertoire dans lequel votre projet sera créé à l'aide de la commande suivante : `mkdir example`
2. Accédez à ce répertoire à l'aide de la commande suivante : `cd example`
3. Entrez la commande suivante : `dotnet new console -o myproject`

Cette opération créera les fichiers suivants dans votre répertoire `example` :

- `Program.cs`, où vous écrirez le code de votre fonction Lambda.
- `MyProject.csproj`, fichier XML qui répertorie les fichiers et les dépendances composant votre application .NET.

AWS Lambda propose des modèles supplémentaires via le package nuget [Amazon.Lambda.Templates](#). Pour installer le package, exécutez la commande suivante :

```
dotnet new -i Amazon.Lambda.Templates
```

Une fois l'installation terminée, les modèles Lambda s'affichent comme partie intégrante de `dotnet new`. Pour le vérifier, exécutez la commande suivante :

```
dotnet new -all
```

Vous devez maintenant voir ce qui suit :

```
dotnet new -all
Usage: new [options]

Options:
  -h, --help           Displays help for this command.
  -l, --list            Lists templates containing the specified name. If no name is
                        specified, lists all templates.
  -n, --name             The name for the output being created. If no name is specified, the
                        name of the current directory is used.
  -o, --output           Location to place the generated output.
  -i, --install          Installs a source or a template pack.
  -u, --uninstall        Uninstalls a source or a template pack.
  --nuget-source         Specifies a NuGet source to use during install.
  --type                 Filters templates based on available types. Predefined values are
                        "project", "item" or "other".
  --force                Forces content to be generated even if it would change existing
                        files.
  -lang, --language       Filters templates based on language and specifies the language of the
                        template to create.
```

Templates	Language	Tags	Short Name
Order Flowers Chatbot Tutorial	[C#]	AWS/Lambda/Function	lambda.OrderFlowersChatbot
Lambda Detect Image Labels	[C#, F#]	AWS/Lambda/Function	lambda.DetectImageLabels
Lambda Empty Function	[C#, F#]	AWS/Lambda/Function	lambda.EmptyFunction
Lex Book Trip Sample	[C#]	AWS/Lambda/Function	lambda.LexBookTripSample
Lambda Simple DynamoDB Function	[C#, F#]	AWS/Lambda/Function	lambda.DynamoDB
Lambda Simple Kinesis Firehose Function	[C#]	AWS/Lambda/Function	lambda.KinesisFirehose
Lambda Simple Kinesis Function	[C#, F#]	AWS/Lambda/Function	lambda.Kinesis
Lambda Simple S3 Function	[C#, F#]	AWS/Lambda/Function	lambda.S3
Lambda ASP.NET Core Web API	[C#, F#]	AWS/Lambda/Serverless	serverless.AspNetCoreWebAPI
Lambda ASP.NET Core Web Application with Razor Pages	[C#]	AWS/Lambda/Serverless	serverless.AspNetCoreWebApp
Serverless Detect Image Labels	[C#, F#]	AWS/Lambda/Serverless	serverless.DetectImageLabels
Lambda DynamoDB Blog API	[C#]	AWS/Lambda/Serverless	serverless.DynamoDBBlogAPI
Lambda Empty Serverless	[C#, F#]	AWS/Lambda/Serverless	serverless.EmptyServerless
Lambda Giraffe Web App	F#	AWS/Lambda/Serverless	serverless.Giraffe
Serverless Simple S3 Function	[C#, F#]	AWS/Lambda/Serverless	serverless.S3

Step Functions Hello World			
serverless.StepFunctionsHelloWorld	[C#], F#	AWS/Lambda/Serverless	
Console Application		console	
[C#], F#, VB	Common/Console		
Class library		classlib	
[C#], F#, VB	Common/Library		
Unit Test Project		mstest	
[C#], F#, VB	Test/MSTest		
xUnit Test Project		xunit	
[C#], F#, VB	Test/xUnit		
Razor Page		page	
[C#]	Web/ASP.NET		
MVC ViewImports		viewimports	
[C#]	Web/ASP.NET		
MVC ViewStart		viewstart	
[C#]	Web/ASP.NET		
ASP.NET Core Empty		web	
[C#], F#	Web/Empty		
ASP.NET Core Web App (Model-View-Controller)		mvc	
[C#], F#	Web/MVC		
ASP.NET Core Web App		razor	
[C#]	Web/MVC/Razor Pages		
ASP.NET Core with Angular		angular	
[C#]	Web/MVC/SPA		
ASP.NET Core with React.js		react	
[C#]	Web/MVC/SPA		
ASP.NET Core with React.js and Redux		reactredux	
[C#]	Web/MVC/SPA		
Razor Class Library		razorclasslib	
[C#]	Web/Razor/Library/Razor Class Library		
ASP.NET Core Web API		webapi	
[C#], F#	Web/WebAPI		
global.json file		globaljson	
NuGet Config	Config	nugetconfig	
Web Config	Config	webconfig	
Solution File	Config		
	Solution	sln	
Examples:			
dotnet new mvc --auth Individual			
dotnet new viewimports --namespace			
dotnet new --help			

Utilisez la commande suivante pour afficher des détails sur un modèle particulier :

```
dotnet new lambda.EmptyFunction --help
```

Notez bien ce qui suit :

```
-p|--profile  The AWS credentials profile set in aws-lambda-tools-defaults.json and used
as the default profile when interacting with AWS.
string - Optional

-r|--region   The AWS region set in aws-lambda-tools-defaults.json and used as the default
region when interacting with AWS.
string - Optional
```

Il s'agit de valeurs facultatives que vous pouvez définir lorsque vous créez votre fonction Lambda et qui seront ensuite écrites automatiquement dans le fichier `aws-lambda-tools-defaults.json`, conçu

comme partie intégrante du processus de création de la fonction. L'exemple suivant explique ce qu'elles signifient :

- `--profile` : votre rôle d'exécution.

Pour créer un rôle IAM (rôle d'exécution) :

1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Pour créer un rôle IAM (rôle d'exécution), suivez les étapes [Rôles IAM](#) dans le IAM Guide de l'utilisateur. Lors de la création d'un rôle, prenez note des points suivants :
 - Dans Select Role Type, sélectionnez AWS Service Roles, puis AWS Lambda.
 - Dans Attacher la stratégie, choisissez la stratégie qui convient le mieux aux exigences de votre fonction Lambda. Si elle n'interagit pas avec d'autres services AWS, vous choisirez AWSLambdaBasicExecutionRole. Cependant, supposons que votre fonction Lambda interagisse avec Kinesis, vous devrez choisir AWSLambdaKinesisExecutionRole.
- `--region` : région Amazon dans laquelle votre fonction réside.

Par exemple, pour créer une fonction Lambda, exécutez la commande suivante, en remplaçant les valeurs du paramètre `--region` par la région de votre choix et `--profile` par votre profil IAM :

Note

Pour plus d'informations sur les conditions requises pour la fonction Lambda, consultez [CreateFunction \(p. 410\)](#)

```
dotnet new lambda.EmptyFunction --name MyFunction --profile default --region region
```

Une structure de répertoire similaire à l'exemple suivant est ainsi créée :

```
<dir>myfunction
  /src/myfunction
  /test/myfunction
```

Sous le répertoire `src/myfunction`, examinez les fichiers suivants :

- `aws-lambda-tools-defaults.json` : c'est ici que vous spécifiez les options de ligne de commande lors du déploiement de votre fonction Lambda. Par exemple :

```
"profile": "iam profile",
"region" : "region",
"configuration" : "Release",
"framework" : "netcoreapp2.1",
"function-runtime": "dotnetcore2.1",
"function-memory-size" : 256,
"function-timeout" : 30,
"function-handler" : "MyFunction::MyFunction.Function::FunctionHandler"
```

- `Function.cs` : le code de la fonction de votre gestionnaire Lambda. Il s'agit d'un modèle C # qui inclut la bibliothèque `Amazon.Lambda.Core` par défaut et un attribut `LambdaSerializer` par défaut. Pour plus d'informations sur les conditions de sérialisation et les options, consultez [Sérialisation des fonctions Lambda \(p. 365\)](#). Il y figure également un exemple de fonction que vous pouvez modifier pour appliquer votre code de fonction Lambda.

```
using System;
using System.Collections.Generic;
```

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

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted into
// a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace MyFunction
{
    public class Function
    {

        public string FunctionHandler1(string input, ILambdaContext context)
        {
            return input?.ToUpper();
        }
    }
}
```

- MyFunction.csproj : fichier [MSBuild](#) qui répertorie les fichiers et les assemblies qui composent votre application.

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <TargetFramework>netcoreapp2.1</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
        <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
    </ItemGroup>

</Project>
```

- Readme : utilisez ce fichier pour documenter votre fonction Lambda.

Sous myfunction/test directory, examine the following files:

- myFunction.Tests.csproj : comme indiqué précédemment, il s'agit d'un fichier [MSBuild](#) qui répertorie les fichiers et les assemblies qui composent votre projet de test. Notez également qu'il comprend aussi la bibliothèque `Amazon.Lambda.Core`, ce qui vous permet d'intégrer de façon transparente les modèles Lambda requis pour tester votre fonction.

```
<Project Sdk="Microsoft.NET.Sdk">
    ...
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
    ...

```

- FunctionTest.cs : le même fichier de modèle de code C# que celui inclus dans le répertoire `src`. Modifiez ce fichier pour mettre en miroir le code de production de votre fonction et testez-le avant de télécharger votre fonction Lambda sur un environnement de production.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Xunit;
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {

            // Invoke the lambda function and confirm the string was upper cased.
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

Une fois que votre fonction a réussi ses tests, vous pouvez générer et déployer à l'aide de l'outil .NET Core Global Amazon.Lambda.Tools. Pour installer l'outil .NET Core Global, exécutez la commande suivante.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Si cet outil est déjà installé, vous pouvez vous assurer que vous utilisez la dernière version avec la commande suivante.

```
dotnet tool update -g Amazon.Lambda.Tools
```

Pour plus d'informations sur l'outil .NET Core Global Amazon.Lambda.Tools, consultez son [dépôt GitHub](#).

Lorsque Amazon.Lambda.Tools est installé, vous pouvez déployer votre fonction à l'aide de la commande suivante :

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

Après déploiement, vous pouvez la tester à nouveau dans un environnement de production avec la commande suivante et transmettre une autre valeur à votre gestionnaire de fonction Lambda :

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
```

En supposant que tout a réussi, vous devriez voir les éléments suivants :

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id Duration: 0.99 ms          Billed Duration: 100 ms      Memory Size: 256 MB     Max Memory Used: 12 MB
```

Sérialisation des fonctions Lambda

Pour toutes les fonctions Lambda qui utilisent des types d'entrée ou de sortie autres qu'un objet `Stream`, vous devrez ajouter une bibliothèque de sérialisation à votre application. Vous pouvez effectuer cette opération de différentes manières :

- Utiliser `Json.NET`. Lambda fournira une implémentation pour le sérialiseur JSON à l'aide de `JSON.NET`, sous la forme d'un package [NuGet](#).
- Créer votre propre bibliothèque de sérialisation en implémentant l'interface `ILambdaSerializer`, disponible dans la bibliothèque `Amazon.Lambda.Core`. L'interface définit deux méthodes :

- `T Deserialize<T>(Stream requestStream);`

Vous implémentez cette méthode afin de désérialiser la charge utile de la demande à partir de l'API `Invoke` dans l'objet qui est transféré au gestionnaire de la fonction Lambda.

- `T Serialize<T>(T response, Stream responseStream);`

Vous implémentez cette méthode pour sérialiser le résultat renvoyé depuis le gestionnaire de la fonction Lambda dans la charge utile de réponse renvoyée par l'API `Invoke`.

Vous pouvez utiliser le sérialiseur de votre choix en l'ajoutant en tant que dépendance de votre fichier `MyProject.csproj`.

```
...
<ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
</ItemGroup>
```

Vous l'ajoutez ensuite à votre fichier `AssemblyInfo.cs`. Par exemple, si vous utilisez le sérialiseur `Json.NET` par défaut, voici ce que vous devez ajouter :

```
[assembly:LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
```

Note

Vous pouvez définir un attribut de sérialisation personnalisé au niveau de la méthode, ce qui remplacera le sérialiseur par défaut spécifié au niveau de l'assemblage. Pour plus d'informations, consultez [Gestion des types de données standard \(p. 368\)](#).

AWS Toolkit for Visual Studio

Vous pouvez créer des applications Lambda basées sur .NET à l'aide du plug-in Lambda d'[AWS Toolkit for Visual Studio](#). Le plug-in est disponible dans un package [Nuget](#).

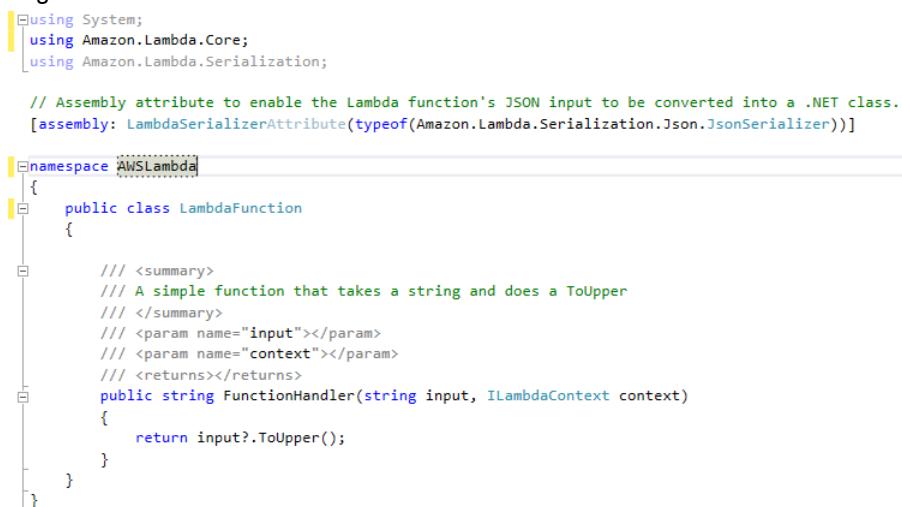
Étape 1 : Crédit et développement d'un projet

1. Lancez Microsoft Visual Studio et choisissez New project.
 - a. Dans le menu File, sélectionnez New, puis Project.
 - b. Dans la fenêtre New Project, choisissez AWS Lambda Project (.NET Core), puis OK.
 - c. La fenêtre Select Blueprint vous permettra de faire votre sélection à partir d'une liste d'exemples d'application, qui vous fourniront un exemple de code afin de démarrer la création d'une application Lambda basée sur .NET.
 - d. Pour créer une application Lambda à partir de zéro, choisissez Blank Function (Fonction vide), puis Terminer.

2. Examinez le fichier `aws-lambda-tools-defaults.json` qui est créé dans le cadre de votre projet. Vous pouvez définir les options dans ce fichier, qui est lu par les outils Lambda par défaut. Les modèles de projet créés dans Visual Studio définissent la plupart de ces champs avec des valeurs par défaut. Notez les champs suivants :
 - profil : rôle IAM requis pour l'exécution de votre fonction Lambda. Si vous n'avez pas encore créé un rôle d'exécution, procédez comme suit :
 1. Connectez-vous à la AWS Management Console et ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
 2. Pour créer un rôle IAM (rôle d'exécution), suivez les étapes fournies dans [Création d'un rôle pour la délégation d'autorisations à un service AWS](#), dans le IAM Guide de l'utilisateur. Lors de la création d'un rôle, prenez note des points suivants :
 - Dans Nom du rôle, utilisez un nom qui est unique au sein de votre compte AWS.
 - Dans Sélectionner un type de rôle, choisissez Rôles de service AWS, puis choisissez un rôle de service qui accorde à ce service des autorisations pour assumer le rôle.
 - Dans Attacher la stratégie, choisissez une stratégie d'autorisation qui convienne pour exécuter votre fonction Lambda.
 - function-handler : emplacement où `function_handler` est spécifié, raison pour laquelle vous n'avez pas à le définir dans l'Assistant. Cependant, chaque fois que vous renommez l'`assembly`, l'`espace de noms`, la `classe` ou la `fonction` dans votre code de fonction, vous devez mettre à jour les champs correspondants dans le fichier `aws-lambda-tools-defaults.json`.

```
{
    "profile": "iam-execution-profile",
    "region" : "region",
    "configuration" : "Release",
    "framework" : "netcoreapp2.1",
    "function-runtime": "dotnetcore2.1",
    "function-memory-size" : 256,
    "function-timeout" : 30,
    "function-handler" : "Assembly::Namespace.Class::Function"
}
```

3. Ouvrez le fichier `Function.cs`. Un modèle vous sera fourni pour vous permettre d'implémenter le code de gestionnaire de votre fonction Lambda.



```
using System;
using Amazon.Lambda.Core;
using Amazon.Lambda.Serialization;

// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializerAttribute(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace AWSLambda
{
    public class LambdaFunction
    {

        /// <summary>
        /// A simple function that takes a string and does a ToUpper
        /// </summary>
        /// <param name="input"></param>
        /// <param name="context"></param>
        /// <returns></returns>
        public string FunctionHandler(string input, ILambdaContext context)
        {
            return input?.ToUpper();
        }
    }
}
```

4. Une fois que vous aurez écrit le code représentant votre fonction Lambda, vous pourrez le charger en cliquant avec le bouton droit sur le nœud Project dans votre application, puis en choisissant Publish to AWS Lambda (Publier dans AWS Lambda).

5. Dans la fenêtre Upload Lambda Function (Charger la fonction Lambda), saisissez un nom pour la fonction ou sélectionnez une fonction précédemment publiée à republier. Ensuite, sélectionnez Next
6. Dans la fenêtre Advanced Function Details, procédez comme suit :
 - Indiquez le nom de Role Name (Nom du rôle), le rôle IAM mentionné précédemment.
 - (Facultatif) Dans Environment:: spécifiez les variables d'environnement que vous souhaitez utiliser. Pour plus d'informations, consultez [Variables d'environnement AWS Lambda \(p. 30\)](#).
 - (Facultatif) Spécifiez les configurations Memory (MB): ou Timeout (Secs):.
 - (Facultatif) Spécifiez les configurations VPC : si votre fonction Lambda a besoin d'accéder aux ressources exécutées à l'intérieur d'un VPC. Pour plus d'informations, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC \(p. 61\)](#).
 - Choisissez Next, puis Upload pour déployer votre application.

Pour plus d'informations, consultez [Déploiement d'un projet AWS Lambda avec l'interface de ligne de commande .NET Core](#).

Gestionnaire de fonctions AWS Lambda dans C#

Lorsque vous créez une fonction Lambda, vous spécifiez un gestionnaire qu'AWS Lambda peut appeler lorsque le service exécute la fonction en votre nom.

Vous définissez un gestionnaire de la fonction Lambda en tant qu'instance ou méthode statique dans une classe. Si vous voulez accéder à l'objet de contexte Lambda, celui-ci est disponible en définissant un paramètre de méthode de type `ILambdaContext`, une interface qui vous permet d'accéder à des informations sur l'exécution actuelle, comme le nom de la fonction en cours d'exécution, la limite de mémoire, le temps d'exécution restant et la journalisation.

```
returnType handler-name(inputType input, ILambdaContext context) {  
    ...  
}
```

Dans la syntaxe, notez les éléments suivants :

- `inputType` – Le premier paramètre du gestionnaire sont les données d'entrée, qui peuvent correspondre à des données d'événement (publiées par une source d'événement) ou à des données d'entrée personnalisées que vous spécifiez, telles qu'une chaîne ou n'importe quel objet de données personnalisé.
- `returnType` – Si vous envisagez d'appeler la fonction Lambda de manière synchrone (via le type d'appel `RequestResponse`), vous pouvez renvoyer la sortie de votre fonction en utilisant l'un quelconque des types de données pris en charge. Par exemple, si vous utilisez une fonction Lambda en tant que back-end pour application mobile, vous lappelez de façon synchrone. Le type de données de sortie sera sérialisé dans JSON.

Si vous envisagez d'appeler la fonction Lambda de manière asynchrone (en utilisant le type d'appel `Event`), le paramètre `returnType` doit être `void`. Par exemple, si vous utilisez AWS Lambda avec des sources d'événements comme Amazon S3 ou Amazon SNS, ces sources d'événements appellent la fonction Lambda en utilisant le type d'appel `Event`.

Gestion des flux

Seul le type `System.IO.Stream` est pris en charge comme paramètre d'entrée par défaut.

Prenons l'exemple de code C# suivant.

```
using System.IO;

namespace Example{

    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

Dans l'exemple de code C#, le premier paramètre du gestionnaire correspond aux données d'entrée du gestionnaire (MyHandler), qui peuvent correspondre à des données d'événement (publiées par une source d'événement comme Amazon S3), à des données d'entrée personnalisées que vous spécifiez, telles qu'un `Stream` (comme dans cet exemple), ou à n'importe quel objet de données personnalisé. La sortie est de type `Stream`.

Gestion des types de données standard

Pour tous les autres types, comme indiqué ci-dessous, vous devez spécifier un sérialiseur.

- Types primitifs .NET (par exemple, chaîne ou int).
- Ensembles et cartes – `IList`, `IEnumerable`, `IList<T>`, `Array`, `IDictionary`, `IDictionary<TKey, TValue>`
- Types POCO (Plain old CLR objects)
- Types d'événement AWS prédéfinis
- Concernant les appels asynchrones, le type de retour sera ignoré par Lambda. Le type de retour peut être défini sur « void » dans de tels cas.
- Si vous utilisez la programmation asynchrone .NET, le type de retour peut être `Task` et `Task<T>`, et utiliser les mots clés `async` et `await`. Pour plus d'informations, consultez [Utilisation des fonctions asynchrones dans C# avec AWS Lambda \(p. 370\)](#).

Vous devez sérialiser les paramètres d'entrée et de sortie de votre fonction, sauf si ces derniers sont de type `System.IO.Stream`. AWS Lambda fournit un sérialiseur par défaut qui peut être appliqué au niveau de l'assemblage ou de la méthode de votre application. Ou bien, vous pouvez définir votre propre sérialiseur en implémentant l'interface `ILambdaSerializer` fournie par la bibliothèque `Amazon.Lambda.Core`. Pour plus d'informations, consultez [Package de déploiement AWS Lambda dans C# \(p. 358\)](#).

Pour ajouter l'attribut de sérialiseur par défaut à une méthode, vous devez d'abord ajouter une dépendance sur `Amazon.Lambda.Serialization.Json` dans votre fichier `project.json`.

```
{
    "version": "1.0.0-*",
    "dependencies": {
        "Microsoft.NETCore.App": {
            "type": "platform",
            "version": "1.0.1"
        },
        "Amazon.Lambda.Serialization.Json": "1.3.0"
    },
    "frameworks": {
        "netcoreapp1.0": {
            "imports": "dnxcore50"
        }
    }
}
```

```
}
```

L'exemple suivant illustre la souplesse dont vous pouvez tirer parti en spécifiant le sérialiseur Json.NET par défaut sur une méthode, puis le sérialiseur de votre choix sur une autre méthode :

```
public class ProductService{
    [LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
    public Product DescribeProduct(DescribeProductRequest request)
    {
        return catalogService.DescribeProduct(request.Id);
    }

    [LambdaSerializer(typeof(MyJsonSerializer))]
    public Customer DescribeCustomer(DescribeCustomerRequest request)
    {
        return customerService.DescribeCustomer(request.Id);
    }
}
```

Signatures de gestionnaire

Lors de la création de fonctions Lambda, vous devez fournir une chaîne de gestionnaire, qui indiquera à AWS Lambda où le code à appeler doit être recherché. Dans C#, le format est le suivant :

ASSEMBLY::TYPE::METHOD où :

- **ASSEMBLY** est le nom du fichier d'assemblage .NET pour votre application. Si vous n'avez pas défini le nom de l'assemblage à l'aide du paramètre `buildOptions.outputName` dans `project.json` lorsque vous utilisez l'interface de ligne de commande .NET Core, le nom **ASSEMBLY** sera le nom du dossier contenant votre fichier `project.json`. Pour plus d'informations, consultez [Interface de ligne de commande .NET Core \(p. 358\)](#). Dans ce cas, supposons que le nom du dossier soit `HelloWorldApp`.
- **TYPE** est le nom complet du type de gestionnaire, composé de `Namespace` et de `ClassName`. Dans ce cas `Example.Hello`.
- **METHOD** est le nom du gestionnaire de la fonction, dans le cas présent `MyHandler`.

En fin de compte, la signature sera au format **Assembly::Namespace.ClassName::MethodName**

Prenons à nouveau l'exemple suivant :

```
using System.IO;
{
    namespace Example

    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

La chaîne de gestionnaire serait : `HelloWorldApp::Example.Hello::MyHandler`

Important

Si la méthode spécifiée dans votre chaîne de gestionnaire est surchargée, vous devez fournir la signature exacte de la méthode que Lambda doit appeler. AWS Lambda rejettéra une signature par ailleurs valide si la résolution exige une sélection parmi plusieurs signatures (surchargeées).

Restrictions liées au gestionnaire de la fonction Lambda

Notez que des restrictions s'appliquent à la signature du gestionnaire

- Il peut ne pas s'agir du mot clé `unsafe` et utiliser des types de pointeur dans la signature du gestionnaire, bien que le contexte `unsafe` puisse être utilisé dans la méthode de gestionnaire et ses dépendances. Pour en savoir plus, consultez [unsafe \(référence C#\)](#).
- Il peut ne pas transférer un certain nombre de paramètres à l'aide du mot clé `params` ou utiliser `ArgIterator` comme paramètre d'entrée ou de retour, pour prendre en charge un nombre de paramètres variable.
- Le gestionnaire peut ne pas être une méthode générique (par exemple, `IList<T> Sort<T>(IList<T> entrée)`).
- Les gestionnaires asynchrones avec une signature `async void` ne sont pas pris en charge.

Utilisation des fonctions asynchrones dans C# avec AWS Lambda

Si vous savez que votre fonction Lambda requerra un processus de longue durée, comme le chargement de fichiers volumineux dans Amazon S3 ou la lecture d'un large flux d'enregistrements depuis DynamoDB, vous pouvez utiliser le modèle `async/await`. En créant un gestionnaire avec cette signature, Lambda exécutera la fonction de façon synchrone et attendra pendant 5 minutes maximum que l'exécution se termine avant le renvoi ou l'expiration. Exemples :

```
public async Task<Response> ProcessS3ImageResizeAsync(SimpleS3Event input)
{
    var response = await client.DoAsyncWork(input);
    return response;
}
```

Si vous utilisez ce modèle, tenez compte des points suivants :

- AWS Lambda ne prendra pas en charge les méthodes `async void`.
- Si vous créez une fonction Lambda asynchrone sans implémenter l'opérateur `await`, .NET émettra un avertissement relatif au compilateur et vous constaterez alors un comportement inattendu. Par exemple, certaines actions asynchrones s'exécuteront, tandis que d'autres non. Ou, certaines actions asynchrones ne se termineront pas tant que l'exécution de la fonction ne sera pas terminée.

```
public async Task ProcessS3ImageResizeAsync(SimpleS3Event event) // Compiler warning
{
    client.DoAsyncWork(input);
}
```

- Votre fonction Lambda peut inclure plusieurs appels asynchrones, qui peuvent être appelés en parallèle. Vous pouvez utiliser les méthodes `Task.WhenAll` et `Task.WhenAny` pour travailler sur plusieurs tâches. Pour utiliser la méthode `Task.WhenAll`, vous transmettez une liste des opérations sous la forme d'une grappe associée à la méthode. Notez que dans l'exemple suivant, si vous n'incluez aucune opération dans la grappe, cet appel peut être renvoyé avant la fin de son exécution.

```
public async Task DoesNotWaitForAllTasks1()
{
    // In Lambda, Console.WriteLine goes to CloudWatch Logs.
```

```
var task1 = Task.Run(() => Console.WriteLine("Test1"));
var task2 = Task.Run(() => Console.WriteLine("Test2"));
var task3 = Task.Run(() => Console.WriteLine("Test3"));

// Lambda may return before printing "Test2" since we never wait on task2.
await Task.WhenAll(task1, task3);
}
```

Pour utiliser la méthode `Task.WhenAny`, vous transmettez à nouveau une liste des opérations sous la forme d'une grappe associée à la méthode. L'appel est renvoyé dès la fin de la première opération, même si les autres sont toujours en cours d'exécution.

```
public async Task DoesNotWaitForAllTasks2()
{
    // In Lambda, Console.WriteLine goes to CloudWatch Logs.
    var task1 = Task.Run(() => Console.WriteLine("Test1"));
    var task2 = Task.Run(() => Console.WriteLine("Test2"));
    var task3 = Task.Run(() => Console.WriteLine("Test3"));

    // Lambda may return before printing all tests since we're only waiting for one to
    // finish.
    await Task.WhenAny(task1, task2, task3);
}
```

Objet de contexte AWS Lambda dans C#

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 367\)](#). Cet objet fournit les propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `FunctionName` – The name of the Lambda function.
- `FunctionVersion` – The [version \(p. 38\)](#) of the function.
- `InvokedFunctionArn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `MemoryLimitInMB` – The amount of memory configured on the function.
- `AwsRequestId` – The identifier of the invocation request.
- `LogGroupName` – The log group for the function.
- `LogStreamName` – The log stream for the function instance.
- `RemainingTime (TimeSpan)` – The number of milliseconds left before the execution times out.
- `Identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `ClientContext` – (mobile apps) Client context provided to the Lambda invoker by the client application.
- `Logger` L'[objet Logger \(p. 372\)](#) pour la fonction.

L'extrait de code C# suivant illustre une fonction simple de gestionnaire qui affiche certaines informations de contexte.

```
public async Task Handler(ILambdaContext context)
{
    Console.WriteLine("Function name: " + context.FunctionName);
    Console.WriteLine("RemainingTime: " + context.RemainingTime);
    await Task.Delay(TimeSpan.FromSeconds(0.42));
```

```
        Console.WriteLine("RemainingTime after sleep: " + context.RemainingTime);  
    }
```

Journalisation des fonctions AWS Lambda dans C#

Votre fonction Lambda peut contenir des instructions de journalisation ; ces journaux sont ensuite écrits par AWS Lambda dans CloudWatch Logs.

Dans le modèle de programmation C#, il existe trois manières d'enregistrer des données dans votre fonction :

- Utilisez les méthodes `Write` ou `WriteLine` statiques fournies par la classe `Console` C#. Tout ce qui est écrit dans une sortie standard ou dans une erreur standard à l'aide de `Console.Write` ou d'une méthode similaire sera consigné dans CloudWatch Logs.

```
public class ProductService  
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request)  
    {  
        Console.WriteLine("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

- Utilisez la méthode `Log` sur la classe `Amazon.Lambda.Core.LambdaLogger`. Il s'agit d'une classe statique qui peut être utilisée n'importe où dans votre application. Pour l'utiliser, vous devez inclure la bibliothèque `Amazon.Lambda.Core`.

```
using Amazon.Lambda.Core;  
  
public class ProductService  
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request)  
    {  
        LambdaLogger.Log("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

Chaque appel à `LambdaLogger.Log` entraîne un événement CloudWatch Logs, dans la mesure où la taille de l'événement respecte les limites autorisées. Pour obtenir des informations sur les limites de CloudWatch Logs, consultez [Limites des journaux CloudWatch](#) dans le Guide de l'utilisateur Amazon CloudWatch.

- Utilisez l'enregistreur d'événements dans `ILambdaContext`. L'objet `ILambdaContext` (s'il est spécifié) de votre méthode contient une propriété `Logger` qui représente un `LambdaLogger`. Vous trouverez ci-dessous un exemple d'utilisation de cette méthode :

```
public class ProductService  
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request,  
        ILambdaContext context)  
    {  
        context.Logger.Log("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

Comment trouver les journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Localisez les journaux dans CloudWatch Logs. L'objet `ILambdaContext` fournit le `LogStreamName` et les propriétés `LogGroupName`. Vous pouvez utiliser ces propriétés pour localiser le flux de journal spécifique dans lequel les journaux sont écrits.
- Si vousappelez une fonction Lambda via la console, le type d'appel est toujours `RequestResponse` (exécution synchrone) et la console affiche les journaux écrits par la fonction Lambda à l'aide de l'objet `LambdaLogger`. AWS Lambda renvoie également les journaux à partir des méthodes `Console.WriteLine` et `Console.WriteLine`.
- Si vousappelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les 4 derniers Ko de données de journalisation écrites dans CloudWatch Logs. Pour plus d'informations, consultez [Invoke \(p. 452\)](#). AWS Lambda renvoie ces informations de journalisation dans l'en-tête `x-amz-log-results` de la réponse. Si vous utilisez l'AWS Command Line Interface pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type` avec la valeur `Tail`.

Erreurs de fonction AWS Lambda dans C#

Lorsqu'une exception se produit dans votre fonction Lambda, Lambda vous communique les informations relatives à cette exception. Les exceptions peuvent se produire à deux endroits différents :

- Initialisation (Lambda charge votre code, valide la chaîne de gestionnaire et crée une instance de votre classe si celle-ci est n'est pas statique).
- L'appel de la fonction Lambda.

Les informations relatives à l'exception sérialisées sont renvoyées sous la forme d'une charge utile en tant qu'objet JSON modélisé, et consignées dans les journaux CloudWatch en tant que résultats.

Dans la phase d'initialisation, les exceptions peuvent être levées concernant des chaînes de gestionnaire non valides, un type enfreignant une règle ou une méthode (voir [Restrictions liées au gestionnaire de la fonction Lambda \(p. 370\)](#)), ou toute autre méthode de validation (si l'attribut de sérialiseur a été oublié et qu'un POCO est défini comme type d'entrée ou de sortie). Ces exceptions sont de type `LambdaException`. Exemples :

```
{  
  "errorType": "LambdaException",  
  "errorMessage": "Invalid lambda function handler: 'http://this.is.not.a.valid.handler/'.  
The valid format is 'ASSEMBLY::TYPE::METHOD'."  
}
```

Si votre constructeur lève une exception, le type d'erreur est également de type `LambdaException`, mais l'exception levée lors de la construction est fournie dans la propriété `cause`, qui constitue elle-même un objet d'exception modélisé :

```
{  
  "errorType": "LambdaException",  
  "errorMessage": "An exception was thrown when the constructor for type  
'LambdaExceptionTestFunction.ThrowExceptionInConstructor'  
was invoked. Check inner exception for more details.",  
  "cause": {  
    "errorType": "TargetInvocationException",  
    "errorMessage": "Exception has been thrown by the target of an invocation.",  
    "stackTrace": [  
      "at System.Runtime.TargetInvocationException.ThrowAsOuterException()",  
      "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess()",  
      "at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification()",  
      "at System.Threading.Tasks.Task.ThrowIfExceptional()",  
      "at System.Threading.Tasks.Task`1.GetResultCore()",  
      "at System.Threading.Tasks.Task`1.get_Result()",  
      "at LambdaExceptionTestFunction.ThrowExceptionInConstructor() in C:\Users\johndoe\source\repos\Lambda  
bTest\LambdaExceptionTestFunction\LambdaExceptionTestFunction.cs:line 12",  
      "at LambdaTestFunction.  
  }  
}
```

```
        "at System.RuntimeTypeHandle.CreateInstance(RuntimeType type, Boolean publicOnly,
Boolean noCheck, Boolean&canBeCached,
RuntimeMethodHandleInternal&ctor, Boolean& bNeedSecurityCheck)",
        "at System.RuntimeType.CreateInstanceSlow(Boolean publicOnly, Boolean skipCheckThis,
Boolean fillCache, StackCrawlMark& stackMark)",
        "at System.Activator.CreateInstance(Type type, Boolean nonPublic)",
        "at System.Activator.CreateInstance(Type type)"
    ],
"cause": {
    "errorType": "ArithmetException",
    "errorMessage": "Sorry, 2 + 2 = 5",
    "stackTrace": [
        "at LambdaExceptionTestFunction.ThrowExceptionInConstructor..ctor()"
    ]
}
}
```

Comme cela est indiqué dans l'exemple, les exceptions internes sont toujours conservées (en tant que propriété cause) et peuvent être profondément imbriquées.

Les exceptions peuvent également se produire pendant un appel. Dans ce cas, le type d'exception est préservé et l'exception est renvoyée directement en tant que charge utile ainsi que dans les journaux CloudWatch. Par exemple :

```
{  
    "errorType": "AggregateException",  
    "errorMessage": "One or more errors occurred. (An unknown web exception occurred!)",  
    "stackTrace": [  
        "at System.Threading.Tasks.Task.ThrowIfExceptional(Boolean  
includeTaskCanceledExceptions)",  
        "at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)",  
        "at lambda_method(Closure , Stream , Stream , ContextInfo )"  
    ],  
    "cause": {  
        "errorType": "UnknownWebException",  
        "errorMessage": "An unknown web exception occurred!",  
        "stackTrace": [  
            "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()",  
            "--- End of stack trace from previous location where exception was thrown ---",  
            "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",  
            "at  
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task  
task)",  
            "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",  
            "at LambdaDemo107.LambdaEntryPoint.<CheckWebsiteStatus>d__0.MoveNext()"  
        ],  
        "cause": {  
            "errorType": "WebException",  
            "errorMessage": "An error occurred while sending the request. SSL peer certificate or  
SSH remote key was not OK",  
            "stackTrace": [  
                "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)",  
                "at System.Threading.Tasks.TaskFactory`1.FromAsyncCoreLogic(IAsyncResult iar,  
Func`2 endFunction, Action`1 endAction, Task`1 promise, Boolean requiresSynchronization)",  
                "--- End of stack trace from previous location where exception was thrown ---",  
                "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",  
                "at  
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task  
task)",  
                "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",  
                "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()"  
            ],  
            "cause": {  
            }  
        }  
    }  
}
```

```
"errorType": "HttpRequestException",
"errorMessage": "An error occurred while sending the request.",
"stackTrace": [
    "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
    "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
    "at System.Net.Http.HttpClient.<FinishSendAsync>d__58.MoveNext()",
    "--- End of stack trace from previous location where exception was thrown ---",
    "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
    "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
    "at System.Net.HttpWebRequest.<SendRequest>d__63.MoveNext()",
    "--- End of stack trace from previous location where exception was thrown ---",
    "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
    "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
    "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)"
],
"cause": {
    "errorType": "CurlException",
    "errorMessage": "SSL peer certificate or SSH remote key was not OK",
    "stackTrace": [
        "at System.Net.Http.CurlHandler.ThrowIfCURLError(CURLcode error)",
        "at
System.Net.Http.CurlHandler.MultiAgent.FinishRequest(StrongToWeakReference`1 easyWrapper,
CURLcode messageResult)"
    ]
}
}
}
}
```

La méthode dans laquelle les informations relatives aux erreurs sont transmises varie selon le type d'appel :

- Type d'appel `RequestResponse` (exécution synchrone) : dans ce cas, vous recevez le message d'erreur.

Par exemple, si vous appelez une fonction Lambda à l'aide de la console Lambda, le type d'appel correspond toujours à `RequestResponse`, et la console affiche les informations d'erreur renvoyées par AWS Lambda dans la section Résultat de l'exécution de la console.

- Type d'appel `Event` (exécution asynchrone) : dans ce cas, AWS Lambda ne renvoie rien. Au lieu de cela, il consigne les informations d'erreur dans CloudWatch Logs et dans les métriques CloudWatch.

En fonction de la source d'événement, AWS Lambda réessaie parfois d'exécuter la fonction Lambda qui a échoué. Pour plus d'informations, consultez [Comportement de nouvelle tentative d'AWS Lambda \(p. 78\)](#).

Gestion des erreurs de fonction

Vous pouvez créer la gestion des erreurs personnalisées pour déclencher une exception directement depuis la fonction Lambda et la gérer directement (Réessayer ou CATCH) dans le cadre d'une machine d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

Prenons l'exemple d'un état `CreateAccount` qui est une tâche écrivant les détails d'un client dans une base de données à l'aide d'une fonction Lambda.

- Si la tâche réussit, un compte est créé et un e-mail de bienvenue est envoyé.
- Si un utilisateur essaie de créer un compte pour un nom d'utilisateur qui existe déjà, la fonction Lambda génère une erreur, ce qui pousse la machine d'état à suggérer un autre nom d'utilisateur et à recommencer le processus de création de compte.

L'extrait de code suivant montre comment procéder. Notez que les erreurs personnalisées dans C# doivent étendre la classe `Exception`.

```
namespace Example {
    public class AccountAlreadyExistsException : Exception {
        public AccountAlreadyExistsException(String message) :
            base(message)
    }
}

namespace Example {
    public class Handler {
        public static void CreateAccount() {
            throw new AccountAlreadyExistsException("Account is in use!");
        }
    }
}
```

Vous pouvez configurer Step Functions de façon à intercepter l'erreur à l'aide d'une règle `Catch`. Lambda définit automatiquement le nom de l'erreur comme étant le nom de classe simple de l'exception lors de l'exécution :

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": [ "AccountAlreadyExistsException" ],
                    "Next": "SuggestAccountName"
                }
            ]
        },
        ...
    }
}
```

Lors de l'exécution, AWS Step Functions identifie l'erreur et [transitionne](#) jusqu'à l'état `SuggestAccountName` comme spécifié dans la transition `Next`.

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda, ce qui vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Création de fonctions Lambda avec PowerShell

Les sections suivantes expliquent comment [les modèles de programmation courants et les concepts de base](#) s'appliquent lorsque vous créez le code d'une fonction Lambda dans PowerShell.

Runtimes .NET

Nom	Identificateur	Langages
.NET Core 2.1	dotnetcore2.1	C# PowerShell Core 6.0
.NET Core 2.0	dotnetcore2.0	C#
.NET Core 1.0	dotnetcore1.0	C#

Notez que les fonctions Lambda dans PowerShell requièrent PowerShell Core 6.0. Windows PowerShell n'est pas pris en charge.

Avant de commencer, vous devez configurer un environnement de développement PowerShell. Pour obtenir des instructions sur la façon de procéder, consultez [Configuration d'un environnement de développement PowerShell \(p. 378\)](#).

Pour découvrir comment utiliser le module AWSLambdaPSCore pour télécharger des exemples de projets PowerShell à partir de modèles, créer des packages de déploiement PowerShell et déployer des fonctions PowerShell dans le cloud AWS, consultez [Utilisation du module AWSLambdaPSCore \(p. 378\)](#).

Rubriques

- [Package de déploiement AWS Lambda dans PowerShell \(p. 377\)](#)
- [Gestionnaire de fonctions AWS Lambda dans PowerShell \(p. 380\)](#)
- [Objet de contexte AWS Lambda dans PowerShell \(p. 382\)](#)
- [Journalisation des fonctions AWS Lambda dans PowerShell \(p. 382\)](#)
- [Erreurs de fonction AWS Lambda dans PowerShell \(p. 383\)](#)

Package de déploiement AWS Lambda dans PowerShell

Un package de déploiement Lambda PowerShell est un fichier ZIP qui contient votre script PowerShell, les modules PowerShell requis pour votre script PowerShell, et les assemblies nécessaires pour héberger PowerShell Core.

AWSLambdaPSCore est un module PowerShell que vous pouvez installer à partir de [PowerShell Gallery](#). Vous devez utiliser ce module pour créer votre package de déploiement Lambda PowerShell.

Vous êtes tenu d'utiliser l'instruction `#Requires` dans vos scripts PowerShell pour indiquer les modules dont vos scripts dépendent. Cette instruction effectue deux tâches importantes. 1) Elle communique aux

autres développeurs les modules que le script utilise, et 2) elle identifie les modules dépendants que les outils AWS PowerShell doivent mettre en package avec le script, dans le cadre du déploiement. Pour plus d'informations sur l'instruction `#Requires` dans PowerShell, consultez [About Requires](#). Pour plus d'informations sur les packages de déploiement PowerShell, consultez [Package de déploiement AWS Lambda dans PowerShell \(p. 377\)](#).

Lorsque votre fonction Lambda PowerShell utilise les applets de commande AWS PowerShell, veillez à définir une instruction `#Requires` qui référence le module `AWS PowerShell .NetCore`, qui prend en charge PowerShell Core (et non pas le module `AWS PowerShell`, qui prend en charge uniquement Windows PowerShell). De plus, veillez à utiliser la version 3.3 270.0 ou plus récente de `AWS PowerShell .NetCore`, qui optimise le processus d'importation d'applet de commande. Si vous utilisez une version plus ancienne, vous connaîtrez des démarrages à froid plus longs. Pour plus d'informations, consultez [Outils AWS pour PowerShell](#).

Avant de commencer, vous devez configurer un environnement de développement PowerShell. Pour obtenir des instructions sur la façon de procéder, consultez [Configuration d'un environnement de développement PowerShell \(p. 378\)](#).

Configuration d'un environnement de développement PowerShell

Pour configurer votre environnement de développement afin d'écrire des scripts PowerShell, procédez comme suit :

1. Installez la bonne version de PowerShell. La prise en charge Lambda de PowerShell repose sur la version inter-plateforme PowerShell Core 6.0. Cela signifie que vous pouvez développer vos fonctions Lambda PowerShell sur Windows, Linux ou Mac. Si vous n'avez pas cette version de PowerShell, vous trouverez des instructions dans [Installation de PowerShell Core](#).
2. Installez le kit SDK .NET Core 2.1. Étant donné que PowerShell Core repose sur .NET Core, la prise en charge de PowerShell dans Lambda utilise le même runtime Lambda .NET Core 2.1 pour les fonctions Lambda .NET Core et PowerShell. Le kit SDK .NET Core 2.1 est utilisé par les nouvelles applets de commande de publication Lambda PowerShell pour créer le package de déploiement Lambda. Le kit SDK .NET Core 2.1 est disponible sur [.NET downloads](#), sur le site web de Microsoft. Veillez à installer le kit SDK et non le runtime.
3. Installez le module AWSLambdaPSCore. Vous pouvez l'installer à partir de [PowerShell Gallery](#) ou en utilisant la commande d'interpréteur PowerShell Core suivante :

```
Install-Module AWSLambdaPSCore -Scope CurrentUser
```

Étapes suivantes

- Pour en savoir plus sur l'écriture des fonctions Lambda dans PowerShell, consultez [Création de fonctions Lambda avec PowerShell \(p. 377\)](#).
- Pour en savoir plus sur l'utilisation du module AWSLambdaPSCore pour télécharger des exemples de projets PowerShell à partir de modèles, sur la création de packages de déploiement PowerShell et sur le déploiement des fonctions PowerShell dans le cloud AWS, consultez [Utilisation du module AWSLambdaPSCore \(p. 378\)](#).

Utilisation du module AWSLambdaPSCore

Le module AWSLambdaPSCore possède les nouvelles applets de commande suivantes pour vous aider à créer et publier des fonctions Lambda PowerShell.

Nom d'applet de commande	Description
Get#AWSPowerShellLambdaTemplate	Renvoie la liste des modèles de mise en route.
New#AWSPowerShellLambda	Crée un script PowerShell initial basé sur un modèle.
Publish#AWSPowerShellLambda	Publie un script PowerShell donné dans Lambda.
New#AWSPowerShellLambdaPackage	Crée un package de déploiement Lambda qui peut être utilisé dans un système d'intégration continue / de livraison continue pour le déploiement.

Pour commencer à écrire et appeler un script PowerShell avec Lambda, vous pouvez utiliser l'applet de commande `New-AWSPowerShellLambda` pour créer un script de démarrage basé sur un modèle. Vous pouvez utiliser l'applet de commande `Publish-AWSPowerShellLambda` pour déployer votre script dans AWS Lambda. Ensuite, vous pouvez tester votre script via la ligne de commande ou la console.

Pour créer un nouveau script PowerShell, le charger et le tester, suivez cette procédure :

1. Consultez les modèles disponibles.

Pour afficher la liste des modèles disponibles, exécutez la commande suivante :

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template          Description
-----
Basic            Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
DetectLabels     Use Amazon Rekognition service to tag image files in Amazon S3
                 with detected labels.
KinesisStreamProcessor Script to process an Amazon Kinesis stream
S3Event          Script to process S3 events
SNSSubscription  Script to be subscribed to an Amazon SNS topic
SQSQueueProcessor Script to be subscribed to an Amazon SQS queue
```

Notez que les nouveaux modèles sont répertoriés en permanence et que ce résultat n'est qu'un exemple.

2. Créez un script basique.

Exécutez la commande suivante pour créer un exemple de script basé sur le modèle `Basic` :

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

Un nouveau fichier nommé `MyFirstPSScript.ps1` est créé dans un nouveau sous-répertoire du répertoire actuel. Le nom de ce répertoire est basé sur le paramètre `-ScriptName`. Vous pouvez utiliser le paramètre `-Directory` pour choisir un autre répertoire.

Vous pouvez voir que le nouveau fichier a le contenu suivant :

```
# PowerShell script file to be executed as a AWS Lambda function.
#
# When executing in Lambda the following variables will be predefined.
#   $LambdaInput - A PSObject that contains the Lambda function input data.
#   $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
#                   information about the currently running Lambda environment.
#
```

```
# The last item in the PowerShell pipeline will be returned as the result of the Lambda
# function.
#
# To include PowerShell modules with your Lambda function, like the
# AWSPowerShell.NetCore module, add a "#Requires" statement
# indicating the module and version.

#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}

# Uncomment to send the input to CloudWatch Logs
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

3. Modifiez l'exemple de script.

Pour voir comment les messages consignés de votre script PowerShell sont envoyés à CloudWatch Logs, supprimez la mise en commentaire de la ligne `Write-Host` dans l'exemple de script.

Pour illustrer comment renvoyer des données à partir de vos fonctions Lambda, ajoutez une nouvelle ligne à la fin du script avec `$PSVersionTable`. Cela ajoute `$PSVersionTable` dans le pipeline PowerShell. Une fois que le script PowerShell est terminé, le dernier objet figurant dans le pipeline PowerShell correspond aux données de retour de la fonction Lambda. `$PSVersionTable` est une variable globale PowerShell qui fournit également des informations sur l'environnement d'exécution.

Après avoir effectué ces modifications, les deux dernières lignes de l'exemple de script ressemblent à ce qui suit :

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
$PSVersionTable
```

4. Publiez dans AWS Lambda.

Après avoir modifié le fichier `MyFirstPSScript.ps1`, modifiez le répertoire en spécifiant l'emplacement du script. Ensuite, exécutez la commande suivante pour publier le script dans AWS Lambda :

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name MyFirstPSScript -
Region us-east-1
```

Notez que le paramètre `-Name` spécifie le nom de la fonction Lambda, qui s'affiche dans la console Lambda. Vous pouvez utiliser cette fonction pour appeler manuellement votre script.

5. Testez la fonction Lambda.

Vous pouvez tester la fonction Lambda PowerShell que vous venez de publier en utilisant l'extension d'interface de ligne de commande dotnet ou la console. La commande d'interface de ligne de commande dotnet suivante est un exemple de la façon de tester votre fonction :

```
dotnet lambda invoke-function MyFirstPSScript --region us-east-1
```

Pour plus d'information sur l'extension dotnet de l'interface de ligne de commande, consultez [Interface de ligne de commande .NET Core \(p. 358\)](#).

Gestionnaire de fonctions AWS Lambda dans PowerShell

Lorsqu'une fonction Lambda est appelée, le gestionnaire Lambda appelle le script PowerShell.

Lorsque le script PowerShell est appelé, les variables suivantes sont prédéfinies :

- **\$LambdaInput** – Un objet PSObject qui contient les données d'entrée pour le gestionnaire. Ces données d'entrée peuvent être des données d'événement (publiées par une source d'événement) ou des données d'entrée personnalisées que vous fournissez, telles qu'une chaîne ou n'importe quel objet de données personnalisé.
- **\$LambdaContext** – Objet Amazon.Lambda.Core.ILambdaContext qui vous permet d'accéder à des informations sur l'exécution en cours — comme le nom de la fonction en cours d'exécution, la limite de mémoire, le temps d'exécution restant et la journalisation.

Prenons l'exemple de code PowerShell suivant.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Ce script renvoie la propriété `FunctionName` qui est obtenue à partir de la variable `$LambdaContext`.

Note

Vous êtes tenu d'utiliser l'instruction `#Requires` dans vos scripts PowerShell pour indiquer les modules dont vos scripts dépendent. Cette instruction effectue deux tâches importantes. 1) Elle communique aux autres développeurs les modules que le script utilise, et 2) elle identifie les modules dépendants que les outils AWS PowerShell doivent mettre en package avec le script, dans le cadre du déploiement. Pour plus d'informations sur l'instruction `#Requires` dans PowerShell, consultez [About Requires](#). Pour plus d'informations sur les packages de déploiement PowerShell, consultez [Package de déploiement AWS Lambda dans PowerShell \(p. 377\)](#). Lorsque votre fonction Lambda PowerShell utilise les applets de commande AWS PowerShell, veillez à définir une instruction `#Requires` qui référence le module `AWSPowerShell.NetCore`, qui prend en charge PowerShell Core (et non pas le module `AWSPowerShell`, qui prend en charge uniquement Windows PowerShell). De plus, veillez à utiliser la version 3.3 270.0 ou plus récente de `AWSPowerShell.NetCore`, qui optimise le processus d'importation d'applet de commande. Si vous utilisez une version plus ancienne, vous connaîtrez des démarrages à froid plus longs. Pour plus d'informations, consultez [Outils AWS pour PowerShell](#).

Renvoi de données

Certains appels Lambda sont destinés à renvoyer des données à leur mandataire. Par exemple, si un appel répond à une demande web provenant d'API Gateway, notre fonction Lambda doit renvoyer la réponse. Pour PowerShell Lambda, le dernier objet qui est ajouté dans le pipeline PowerShell correspond aux données de retour de l'appel Lambda. Si l'objet est une chaîne, les données sont renvoyées en l'état. Dans le cas contraire, l'objet est converti en données JSON à l'aide de l'applet de commande `ConvertTo-Json`.

Par exemple, prenez en considération l'instruction PowerShell suivante, qui ajoute `$PSVersionTable` dans le pipeline PowerShell :

```
$PSVersionTable
```

Une fois que le script PowerShell est terminé, le dernier objet figurant dans le pipeline PowerShell correspond aux données de retour de la fonction Lambda. `$PSVersionTable` est une variable globale PowerShell qui fournit également des informations sur l'environnement d'exécution.

Objet de contexte AWS Lambda dans PowerShell

Lorsque Lambda exécute votre fonction, il transmet les informations de contexte en rendant une variable `$LambdaContext` disponible pour le [gestionnaire \(p. 380\)](#). Cette variable fournit des méthodes et des propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `FunctionName` – The name of the Lambda function.
- `FunctionVersion` – The [version \(p. 38\)](#) of the function.
- `InvokedFunctionArn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.
- `MemoryLimitInMB` – The amount of memory configured on the function.
- `AwsRequestId` – The identifier of the invocation request.
- `LogGroupName` – The log group for the function.
- `LogStreamName` – The log stream for the function instance.
- `RemainingTime` – The number of milliseconds left before the execution times out.
- `Identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `ClientContext` – (mobile apps) Client context provided to the Lambda invoker by the client application.
- `Logger` – L'[objet Logger \(p. 382\)](#) pour la fonction.

L'extrait de code PowerShell suivant illustre une fonction simple de gestionnaire qui affiche certaines informations de contexte.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

Journalisation des fonctions AWS Lambda dans PowerShell

Votre fonction Lambda peut contenir des instructions de journalisation ; ces journaux sont ensuite écrits par AWS Lambda dans CloudWatch Logs.

Dans le modèle de programmation Lambda dans PowerShell, les applets de commande `Write` telles que `Write-Host`, `Write-Output` et `Write-Information` sont toutes écrites dans CloudWatch Logs.

Par exemple, l'exemple de code suivant écrit un message dans CloudWatch Logs :

```
Write-Host 'Hello World. This string is written to CloudWatch logs.'
```

Comment trouver les journaux

Pour trouver les journaux écrits par la fonction Lambda, procédez comme suit :

- Recherchez les journaux dans CloudWatch Logs. La variable `$LambdaContext` fournit les propriétés `LogGroupName` et `LogStreamName`. Vous pouvez utiliser ces propriétés pour localiser le flux de journal spécifique dans lequel les journaux sont écrits.
- Si vous appelez une fonction Lambda par programmation, vous pouvez ajouter le paramètre `LogType` pour récupérer les 4 derniers Ko de données de journalisation écrites dans CloudWatch Logs. Pour plus d'informations, consultez [Invoke \(p. 452\)](#). AWS Lambda renvoie ces informations de journalisation dans l'en-tête `x-amz-log-results` de la réponse. Si vous utilisez l'AWS Command Line Interface (AWS CLI) pour appeler la fonction, vous pouvez spécifier le paramètre `--log-type` avec la valeur `Tail`.

Erreurs de fonction AWS Lambda dans PowerShell

Si la fonction Lambda présente une erreur de mise hors service, AWS Lambda reconnaît l'échec, sérialise les informations correspondantes dans JSON, puis les renvoie.

Prenons l'exemple d'instruction de script PowerShell suivant :

```
throw 'The Account is not found'
```

Lorsque vousappelez cette fonction Lambda, elle lève une erreur de mise hors service, et AWS Lambda renvoie le message d'erreur suivant :

```
{
  "errorMessage": "The Account is not found",
  "errorType": "RuntimeException"
}
```

Notez qu'`errorType` a pour valeur `RuntimeException`, qui correspond à l'exception par défaut levée par PowerShell. Vous pouvez utiliser les types d'erreur personnalisés en levant l'erreur comme suit :

```
throw @{ 'Exception'='AccountNotFound'; 'Message'='The Account is not found' }
```

Le message d'erreur est sérialisé avec `errorType` défini sur `AccountNotFound` :

```
{
  "errorMessage": "The Account is not found",
  "errorType": "AccountNotFound"
}
```

Si vous n'avez pas besoin d'un message d'erreur, vous pouvez lever une chaîne dans le format d'un code d'erreur. Le format de code d'erreur exige que la chaîne commence par un caractère et qu'elle contienne ensuite uniquement des lettres et des chiffres, sans aucun espace ni symbole.

Par exemple, si votre fonction Lambda contient :

```
throw 'AccountNotFound'
```

L'erreur est sérialisée comme suit :

```
{
  "errorMessage": "AccountNotFound",
  "errorType": "AccountNotFound"
}
```

Gestion des erreurs de fonction

Vous pouvez utiliser un attribut `errorType` personnalisé dans votre fonction Lambda et gérer les erreurs de fonction directement (avec `Retry` ou `Catch`) au sein d'une machine d'état AWS Step Functions. Pour plus d'informations, consultez [Gestion des conditions d'erreur à l'aide d'une machine d'état](#).

La gestion des erreurs personnalisée facilite la création des applications [sans serveur](#). Cette fonctionnalité s'intègre à tous les langages pris en charge par le [Modèle de programmation \(p. 24\)](#) Lambda. Ceci vous permet de concevoir votre application dans les langages de programmation de votre choix et de les associer à votre guise.

Pour en savoir plus sur la création de vos propres applications sans serveur à l'aide d'AWS Step Functions et d'AWS Lambda, consultez [AWS Step Functions](#).

Création de fonctions Lambda avec Ruby

Vous pouvez exécuter le code Ruby dans AWS Lambda. Lambda fournit un [runtime \(p. 103\)](#) pour Ruby qui exécute votre code pour traiter les événements. Votre code s'exécute dans un environnement comprenant le Kit SDK AWS pour Ruby, avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez.

Lambda prend en charge les runtimes Ruby suivants :

Runtimes Ruby

Nom	Identificateur
Ruby 2.5	<code>ruby2.5</code>

Si vous ne possédez pas encore de rôle d'exécution pour le développement de fonctions, créez-en un.

Pour créer un rôle d'exécution

1. Ouvrez la [page des rôles](#) dans la console IAM.
2. Choisissez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – Lambda.
 - Autorisations – `AWSLambdaBasicExecutionRole`.
 - Nom de rôle – **lambda-role**.

La stratégie `AWSLambdaBasicExecutionRole` possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Vous pouvez ajouter des autorisations pour ce rôle plus tard, ou le remplacer par un autre rôle spécifique à une fonction particulière.

Pour créer une fonction Ruby

1. Ouvrez la [console Lambda](#).
2. Sélectionnez Create function.
3. Configurez les paramètres suivants :
 - Nom – **ruby-function**.
 - Runtime – Ruby 2.5.
 - Rôle – Sélectionner un rôle existant.
 - Rôle existant – **lambda-role**.
4. Sélectionnez Create function.
5. Pour configurer un événement de test, choisissez Test.

6. Dans Nom d'événement, saisissez **test**.
7. Sélectionnez Créer.
8. Pour exécuter la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `lambda_function.rb`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'[éditeur de code \(p. 17\)](#) intégré. Choisissez Save (Enregistrer) pour sauvegarder vos modifications, puis choisissez Test pour exécuter votre code.

Note

La console Lambda utilise AWS Cloud9 pour fournir un environnement de développement intégré dans le navigateur. Vous pouvez également utiliser AWS Cloud9 pour développer des fonctions Lambda dans votre propre environnement. Pour plus d'informations, consultez [Utilisation des fonctions AWS Lambda](#) dans le guide de l'utilisateur de AWS Cloud9.

`lambda_function.rb` définit une fonction nommée `lambda_handler` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la [fonction de gestionnaire \(p. 386\)](#) que Lambda appelle lorsque la fonction est invoquée. Le runtime de la fonction Ruby obtient les événements d'appels de Lambda et les transmet au gestionnaire.

Chaque fois que vous enregistrez le code de votre fonction, la console Lambda crée un package de déploiement, qui est une archive ZIP contenant le code de votre fonction. Au fur et à mesure du développement de votre fonction, nous vous conseillons de stocker le code de votre fonction dans le contrôle de code source, d'ajouter des bibliothèques et d'automatiser les déploiements. Commencez par [créer un package de déploiement \(p. 387\)](#) et mettre à jour votre code dans la ligne de commande.

Le runtime de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'appel. L'[objet de contexte \(p. 389\)](#) contient des informations supplémentaires sur l'appel, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles avec les variables d'environnement.

Votre fonction Lambda s'accompagne d'un groupe de journaux CloudWatch Logs. Le runtime de la fonction envoie des détails sur chaque appel à CloudWatch Logs, et relaie les [journaux que votre fonction génère \(p. 390\)](#) pendant l'appel. Si votre fonction [renvoie une erreur \(p. 390\)](#), Lambda met en forme l'erreur et la renvoie au mécanisme d'appel.

Rubriques

- [Gestionnaire de fonctions AWS Lambda en Ruby \(p. 386\)](#)
- [Package de déploiement AWS Lambda en Ruby \(p. 387\)](#)
- [Objet de contexte AWS Lambda en Ruby \(p. 389\)](#)
- [Journalisation des fonctions AWS Lambda en Ruby \(p. 390\)](#)
- [Erreurs de fonctions AWS Lambda en Ruby \(p. 390\)](#)

Gestionnaire de fonctions AWS Lambda en Ruby

Le gestionnaire de votre fonction Lambda est la méthode que Lambda appelle lorsque votre fonction est invoquée. Dans l'exemple suivant, le fichier `function.rb` définit une méthode de gestionnaire nommée `handler`. La fonction de gestionnaire prend deux objets en tant qu'entrées et renvoie un document JSON.

Example function.rb

```
require 'json'
```

```
def handler(event:, context:)
  { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Dans la configuration de votre fonction, le paramètre `handler` indique à Lambda où trouver le gestionnaire. Pour l'exemple précédent, la valeur correcte pour ce paramètre est `function.handler`. Elle inclut deux noms séparées par un point : le nom du fichier et le nom de la méthode de gestionnaire.

Vous pouvez également définir votre méthode de gestionnaire dans une classe. L'exemple suivant définit une méthode de gestionnaire nommée `process` sur une classe nommée `Handler` dans un module appelé `LambdaFunctions`.

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
    end
  end
end
```

Dans ce cas, le paramètre de gestionnaire est `source.LambdaFunctions::Handler.process`.

Les deux objets qui sont acceptés par le gestionnaire sont l'événement d'appel et le contexte. L'événement est un objet Ruby qui contient la charge utile qui est fournie par le mécanisme d'appel. Si la charge utile est un document JSON, l'objet d'événement est un hachage Ruby. Sinon, il s'agit d'une chaîne. L'[objet de contexte \(p. 389\)](#) contient des méthodes et des propriétés qui fournissent des informations sur l'appel, la fonction et l'environnement d'exécution.

Le gestionnaire de fonction est exécuté chaque fois que votre fonction Lambda est appelée. Le code statique à l'extérieur du gestionnaire est exécuté une fois par instance de la fonction. Si votre gestionnaire utilise des ressources telles que les clients de kits SDK et les connexions de base de données, vous pouvez les créer à l'extérieur de la méthode de gestionnaire afin de les réutiliser pour plusieurs appels.

Chaque instance de votre fonction peut traiter plusieurs événements d'appels, mais elle ne traite qu'un événement à la fois. Le nombre d'instances traitant un événement à un moment donné est la simultanéité de votre fonction. Pour plus d'informations sur le contexte d'exécution de Lambda, consultez [Contexte d'exécution d'AWS Lambda \(p. 105\)](#).

Package de déploiement AWS Lambda en Ruby

A deployment package is a ZIP archive that contains your function code and dependencies. You need to create a deployment package if you use the Lambda API to manage functions, or if your code uses libraries other than the AWS SDK. Other libraries and dependencies need to be included in the deployment package. You can upload the package directly to Lambda, or you can use an Amazon S3 bucket, and then upload it to Lambda.

If you use the Lambda [console editor \(p. 17\)](#) to author your function, the console manages the deployment package. You can use this method as long as you don't need to add any libraries. You can also use it to update a function that already has libraries in the deployment package, as long as the total size doesn't exceed 3 MB.

Sections

- [Mise à jour d'une fonction sans dépendances \(p. 388\)](#)

- Mise à jour d'une fonction avec dépendances supplémentaires (p. 388)

Mise à jour d'une fonction sans dépendances

Pour créer ou mettre à jour une fonction à l'aide de l'API Lambda, créez une archive contenant le code de votre fonction et chargez-la avec l'AWS CLI.

Pour mettre à jour une fonction Ruby sans dépendances

1. Créez une archive ZIP.

```
~/my-function$ zip function.zip function.rb
```

2. Utilisez la commande update-function-code pour charger le package.

```
~/my-function$ aws lambda update-function-code --function-name ruby25 --zip-file fileb://function.zip
{
    "FunctionName": "ruby25",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:ruby25",
    "Runtime": "ruby2.5",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 300,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-23T21:00:10.248+0000",
    "CodeSha256": "Qf0haXm3JtGTmcDbjMc1I2di6YFMi9niEuiYonYptAk=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "d1e983e3-ca8e-434b-8dc1-7add83d72ebd"
}
```

Mise à jour d'une fonction avec dépendances supplémentaires

Si votre fonction dépend de bibliothèques autres que le Kit SDK AWS pour Ruby, installez-les dans un répertoire local avec [Bundler](#), et incluez-les dans votre package de déploiement.

Pour mettre à jour une fonction Ruby avec dépendances

1. Installez les bibliothèques dans le répertoire des fournisseurs avec la commande bundle.

```
~/my-function$ bundle install --path vendor/bundle
Fetching gem metadata from https://rubygems.org/.....
Resolving dependencies...
Fetching aws-eventstream 1.0.1
Installing aws-eventstream 1.0.1
...
```

Le `--path` installe les gems dans le répertoire de projet au lieu de l'emplacement du système, et définit celui-ci comme chemin d'accès par défaut pour les futures installations. Pour installer des gems de façon globale ultérieurement, utilisez l'option `--system`.

2. Créez une archive ZIP.

```
package$ zip -r function.zip function.rb vendor
  adding: function.rb (deflated 37%)
  adding: vendor/ (stored 0%)
  adding: vendor/bundle/ (stored 0%)
  adding: vendor/bundle/ruby/ (stored 0%)
  adding: vendor/bundle/ruby/2.5.0/ (stored 0%)
  adding: vendor/bundle/ruby/2.5.0/build_info/ (stored 0%)
  adding: vendor/bundle/ruby/2.5.0/cache/ (stored 0%)
  adding: vendor/bundle/ruby/2.5.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

3. Mettez à jour le code de la fonction.

```
~/my-function$ aws lambda update-function-code --function-name ruby25 --zip-file
fileb://function.zip
{
    "FunctionName": "ruby25",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:ruby25",
    "Runtime": "ruby2.5",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 998918,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-20T20:51:35.871+0000",
    "CodeSha256": "fJ3TxYnFosnnpN483dz9/rTzcXrbOiuu4iOZx34nXZI=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "9ca7c45b-bcda-4e51-ab5f-7c42fa916e39"
}
```

Objet de contexte AWS Lambda en Ruby

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 386\)](#). Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'appel, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `get_remaining_time_in_millis` – Returns the number of milliseconds left before the execution times out.

Propriétés du contexte

- `function_name` – The name of the Lambda function.
- `function_version` – The [version \(p. 38\)](#) of the function.
- `invoked_function_arn` – The Amazon Resource Name (ARN) used to invoke the function. Indicates if the invoker specified a version number or alias.

- `memory_limit_in_mb` – The amount of memory configured on the function.
- `aws_request_id` – The identifier of the invocation request.
- `log_group_name` – The log group for the function.
- `log_stream_name` – The log stream for the function instance.
- `deadline_ms` – The date that the execution times out, in Unix time milliseconds.
- `identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `client_context` – (mobile apps) Client context provided to the Lambda invoker by the client application.

Journalisation des fonctions AWS Lambda en Ruby

Votre fonction Lambda est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journaux pour chaque instance de votre fonction. Le runtime envoie des détails sur chaque appel au flux de journaux et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour afficher les journaux de votre fonction Lambda

1. Ouvrez la [page Journaux de la console CloudWatch](#).
2. Choisissez le groupe de journaux pour votre fonction (`/aws/lambda/nom-fonction`).
3. Choisissez le premier flux dans la liste.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des instructions `puts` ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`.

Pour obtenir les journaux pour un appel à partir de la ligne de commande, utilisez l'option `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'appel.

```
$ aws lambda invoke --function-name ruby-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzV1NGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name ruby-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 19 MB
```

`base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Erreurs de fonctions AWS Lambda en Ruby

Lorsque votre code soulève une erreur, Lambda génère une représentation JSON de l'erreur. Ce document d'erreur s'affiche dans le journal d'appels et, pour un appel synchrone, dans la sortie.

Example function.rb

```
def handler(event:, context:)
    puts "Processing event..."
    [1, 2, 3].first("two")
    "Success"
end
```

Ce code débouche sur une erreur de type. Lambda attrape l'erreur et génère un document JSON avec des champs pour le message d'erreur, le type et la trace de la pile.

```
{
  "errorMessage": "no implicit conversion of String into Integer",
  "errorType": "Function<TypeError>",
  "stackTrace": [
    "/var/task/function.rb:3:in `first'",
    "/var/task/function.rb:3:in `handler'"
  ]
}
```

Lorsque vous appelez la fonction à partir de la ligne de commande, le code de statut ne change pas, mais la réponse inclut un champ **FunctionError**, et la sortie inclut le document d'erreur.

```
$ aws lambda invoke --function-name ruby-function out
{
  "StatusCode": 200,
  "FunctionError": "UncaughtException",
  "ExecutedVersion": "$LATEST"
}
$ cat out
{"errorMessage": "no implicit conversion of String into Integer", "errorType": "Function<TypeError>", "stackTrace": ["/var/task/function.rb:3:in `first'", "/var/task/function.rb:3:in `handler'"]}
```

Pour afficher l'erreur dans le journal des erreurs, utilisez l'option **--log-type** et décodez la chaîne en base64 dans la réponse.

```
$ aws lambda invoke --function-name ruby-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Version: $LATEST
Processing event...
Error raised from handler method
{
  "errorMessage": "no implicit conversion of String into Integer",
  "errorType": "Function<TypeError>",
  "stackTrace": [
    "/var/task/function.rb:3:in `first'",
    "/var/task/function.rb:3:in `handler'"
  ]
}
END RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3
REPORT RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Duration: 22.74 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 18 MB
```

Pour plus d'informations, consultez [Journalisation des fonctions AWS Lambda en Ruby \(p. 390\)](#).

Référence API

Cette section contient la documentation de référence de l'API AWS Lambda. Lorsque vous effectuez des appels d'API, vous devez fournir une signature pour authentifier votre demande. AWS Lambda prend en charge Signature version 4. Pour plus d'informations, consultez [Processus de signature Signature Version 4](#) dans la Référence générale d'Amazon Web Services.

Pour obtenir une présentation du service, consultez la section [Qu'est-ce qu'AWS Lambda ? \(p. 1\)](#).

Vous pouvez utiliser l'AWS CLI pour explorer l'API AWS Lambda. Ce guide offre plusieurs didacticiels qui utilisent l'AWS CLI.

Rubriques

- [Actions \(p. 392\)](#)
- [Data Types \(p. 521\)](#)

Actions

The following actions are supported:

- [AddLayerVersionPermission \(p. 394\)](#)
- [AddPermission \(p. 398\)](#)
- [CreateAlias \(p. 402\)](#)
- [CreateEventSourceMapping \(p. 406\)](#)
- [CreateFunction \(p. 410\)](#)
- [DeleteAlias \(p. 418\)](#)
- [DeleteEventSourceMapping \(p. 420\)](#)
- [DeleteFunction \(p. 423\)](#)
- [DeleteFunctionConcurrency \(p. 425\)](#)
- [DeleteLayerVersion \(p. 427\)](#)
- [GetAccountSettings \(p. 429\)](#)
- [GetAlias \(p. 431\)](#)
- [GetEventSourceMapping \(p. 434\)](#)
- [GetFunction \(p. 437\)](#)
- [GetFunctionConfiguration \(p. 440\)](#)
- [GetLayerVersion \(p. 445\)](#)
- [GetLayerVersionPolicy \(p. 448\)](#)
- [GetPolicy \(p. 450\)](#)
- [Invoke \(p. 452\)](#)
- [InvokeAsync \(p. 457\)](#)
- [ListAliases \(p. 459\)](#)
- [ListEventSourceMappings \(p. 462\)](#)
- [ListFunctions \(p. 465\)](#)
- [ListLayers \(p. 468\)](#)
- [ListLayerVersions \(p. 470\)](#)

- [ListTags \(p. 473\)](#)
- [ListVersionsByFunction \(p. 475\)](#)
- [PublishLayerVersion \(p. 478\)](#)
- [PublishVersion \(p. 482\)](#)
- [PutFunctionConcurrency \(p. 488\)](#)
- [RemoveLayerVersionPermission \(p. 491\)](#)
- [RemovePermission \(p. 493\)](#)
- [TagResource \(p. 496\)](#)
- [UntagResource \(p. 498\)](#)
- [UpdateAlias \(p. 500\)](#)
- [UpdateEventSourceMapping \(p. 504\)](#)
- [UpdateFunctionCode \(p. 508\)](#)
- [UpdateFunctionConfiguration \(p. 514\)](#)

AddLayerVersionPermission

Adds permissions to the resource-based policy of a version of an [AWS Lambda layer](#). Use this action to grant layer usage permission to other accounts. You can grant permission to a single account, all AWS accounts, or all accounts in an organization.

To revoke permission, call [RemoveLayerVersionPermission \(p. 491\)](#) with the statement ID that you specified when you added it.

Request Syntax

```
POST /2018-10-31/layers/LayerName/versions/VersionNumber/policy?RevisionId=RevisionId
HTTP/1.1
Content-type: application/json

{
    "Action": "string",
    "OrganizationId": "string",
    "Principal": "string",
    "StatementId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 394\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|[a-zA-Z0-9-_+]

[RevisionId \(p. 394\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

[VersionNumber \(p. 394\)](#)

The version number.

Request Body

The request accepts the following data in JSON format.

[Action \(p. 394\)](#)

The API action that grants access to the layer. For example, lambda:GetLayerVersion.

Type: String

Pattern: lambda:GetLayerVersion

Required: Yes

[OrganizationId \(p. 394\)](#)

With the principal set to *, grant permission to all accounts in the specified organization.

Type: String

Pattern: o-[a-zA-Z0-9]{10,32}

Required: No

[Principal \(p. 394\)](#)

An account ID, or * to grant permission to all AWS accounts.

Type: String

Pattern: \d{12}|*|arn:(aws[a-zA-Z-]*):iam::\d{12}:root

Required: Yes

[StatementId \(p. 394\)](#)

An identifier that distinguishes the policy from others on the same layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

Required: Yes

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "RevisionId": "string",
    "Statement": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[RevisionId \(p. 395\)](#)

A unique identifier for the current revision of the policy.

Type: String

[Statement \(p. 395\)](#)

The permission statement.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PolicyLengthExceededException

Lambda function access policy is limited to 20 KB.

HTTP Status Code: 400

PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

AddPermission

Grants an AWS service or another account permission to use a function. You can apply the policy at the function level, or specify a qualifier to restrict access to a single version or alias. If you use a qualifier, the invoker must use the full Amazon Resource Name (ARN) of that version or alias to invoke the function.

To grant permission to another account, specify the account ID as the `Principal`. For AWS services, the principal is a domain-style identifier defined by the service, like `s3.amazonaws.com` or `sns.amazonaws.com`. For AWS services, you can also specify the ARN or owning account of the associated resource as the `SourceArn` or `SourceAccount`. If you grant permission to a service principal without specifying the source, other accounts could potentially configure resources in their account to invoke your Lambda function.

This action adds a statement to a resource-based permission policy for the function. For more information about function policies, see [Lambda Function Policies](#).

Request Syntax

```
POST /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "Action": "string",
  "EventSourceToken": "string",
  "Principal": "string",
  "RevisionId": "string",
  "SourceAccount": "string",
  "SourceArn": "string",
  "StatementId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 398)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Qualifier](#) (p. 398)

Specify a version or alias to add permissions to a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (`|[a-zA-Z0-9$-_]+`)

Request Body

The request accepts the following data in JSON format.

[Action \(p. 398\)](#)

The action that the principal can use on the function. For example, `lambda:InvokeFunction` or `lambda:GetFunction`.

Type: String

Pattern: (`lambda:[*]` | `lambda:[a-zA-Z]+[*]`)

Required: Yes

[EventSourceToken \(p. 398\)](#)

For Alexa Smart Home functions, a token that must be supplied by the invoker.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Pattern: [a-zA-Z0-9._\-\+]+

Required: No

[Principal \(p. 398\)](#)

The AWS service or account that invokes the function. If you specify a service, use `SourceArn` or `SourceAccount` to limit who can invoke the function through that service.

Type: String

Pattern: .*

Required: Yes

[RevisionId \(p. 398\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

Type: String

Required: No

[SourceAccount \(p. 398\)](#)

For AWS services, the ID of the account that owns the resource. Use instead of `SourceArn` to grant permission to resources owned by another account (e.g. all of an account's Amazon S3 buckets).

Or use together with `SourceArn` to ensure that the resource is owned by the specified account. For example, an Amazon S3 bucket could be deleted by its owner and recreated by another account.

Type: String

Pattern: \d{12}

Required: No

[SourceArn \(p. 398\)](#)

For AWS services, the ARN of the AWS resource that invokes the function. For example, an Amazon S3 bucket or Amazon SNS topic.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z{2}(-gov)?-[a-zA-Z]+\d{1})?:(\d{12})?:(.*?)

Required: No

[StatementId \(p. 398\)](#)

A statement identifier that differentiates the statement from others in the same policy.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9_-]+)

Required: Yes

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "Statement": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[Statement \(p. 400\)](#)

The permission statement added to the function policy.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[PolicyLengthExceededException](#)

Lambda function access policy is limited to 20 KB.

HTTP Status Code: 400

[PreconditionFailedException](#)

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

- HTTP Status Code: 412
ResourceConflictException

The resource already exists.
- HTTP Status Code: 409
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.
- HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.
- HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

CreateAlias

Creates an [alias](#) for a Lambda function version. Use aliases to provide clients with a function identifier that you can update to invoke a different version.

You can also map an alias to split invocation requests between two versions. Use the `RoutingConfig` parameter to specify a second version and the percentage of invocation requests that it receives.

Request Syntax

```
POST /2015-03-31/functions/FunctionName/aliases HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 402)

The name of the lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Request Body

The request accepts the following data in JSON format.

[Description](#) (p. 402)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 402\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

Required: Yes

[Name \(p. 402\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Required: Yes

[RoutingConfig \(p. 402\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 403\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Description \(p. 403\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 403\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[Name \(p. 403\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\0-9]+$)([a-zA-Z0-9-_]+)`

[RevisionId \(p. 403\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 403\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

CreateEventSourceMapping

Creates a mapping between an event source and an AWS Lambda function. Lambda reads items from the event source and triggers the function.

For details about each event source type, see the following topics.

- [Using AWS Lambda with Amazon Kinesis](#)
- [Using AWS Lambda with Amazon SQS](#)
- [Using AWS Lambda with Amazon DynamoDB](#)

Request Syntax

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "Enabled": boolean,
    "EventSourceArn": "string",
    "FunctionName": "string",
    "StartingPosition": "string",
    "StartingPositionTimestamp": number
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

BatchSize ([p. 406](#))

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

Enabled ([p. 406](#))

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

EventSourceArn ([p. 406](#))

The Amazon Resource Name (ARN) of the event source.

- Amazon Kinesis - The ARN of the data stream or a stream consumer.
- Amazon DynamoDB Streams - The ARN of the stream.
- Amazon Simple Queue Service - The ARN of the queue.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)

Required: Yes

[FunctionName \(p. 406\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Version or Alias ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Required: Yes

[StartingPosition \(p. 406\)](#)

The position in a stream from which to start reading. Required for Amazon Kinesis and Amazon DynamoDB Streams sources. AT_TIMESTAMP is only supported for Amazon Kinesis streams.

Type: String

Valid Values: TRIM_HORIZON | LATEST | AT_TIMESTAMP

Required: No

[StartingPositionTimestamp \(p. 406\)](#)

With StartingPosition set to AT_TIMESTAMP, the time from which to start reading, in Unix time seconds.

Type: Timestamp

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json
```

```
{  
    "BatchSize": number,  
    "EventSourceArn": "string",  
    "FunctionArn": "string",  
    "LastModified": number,  
    "LastProcessingResult": "string",  
    "State": "string",  
    "StateTransitionReason": "string",  
    "UUID": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 407\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 407\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*?)

[FunctionArn \(p. 407\)](#)

The ARN of the Lambda function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 407\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 407\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 407\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 407\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String
[UUID \(p. 407\)](#)

The identifier of the event source mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

CreateFunction

Creates a Lambda function. To create a function, you need a [deployment package](#) and an [execution role](#). The deployment package contains your function code. The execution role grants the function permission to use AWS services such as Amazon CloudWatch Logs for log streaming and AWS X-Ray for request tracing.

A function has an unpublished version, and can have published versions and aliases. A published version is a snapshot of your function code and configuration that can not be changed. An alias is a named resource that maps to a version, and can be changed to map to a different version. Use the `Publish` parameter to create version 1 of your function from its initial configuration.

The other parameters let you configure version-specific and function-level settings. You can modify version-specific settings later with [UpdateFunctionConfiguration \(p. 514\)](#). Function-level settings apply to both the unpublished and published versions of the function and include tags ([TagResource \(p. 496\)](#)) and per-function concurrency limits ([PutFunctionConcurrency \(p. 488\)](#)).

If another account or a AWS service invokes your function, use [AddPermission \(p. 398\)](#) to grant permission by creating a resource-based IAM policy. You can grant permissions at the function level, on a version, or on an alias.

To invoke your function directly, use [Invoke \(p. 452\)](#). To invoke your function in response to events in other AWS services, create an event source mapping ([CreateEventSourceMapping \(p. 406\)](#)), or configure a function trigger in the other service. For more information, see [Invoking Functions](#).

Request Syntax

```
POST /2015-03-31/functions HTTP/1.1
Content-type: application/json

{
  "Code": {
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string",
    "ZipFile": blob
  },
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string" : "string"
    }
  },
  "FunctionName": "string",
  "Handler": "string",
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "Publish": boolean,
  "Role": "string",
  "Runtime": "string",
  "Tags": {
    "string" : "string"
  },
  "Timeout": number,
  "TracingConfig": {
    "Mode": "string"
  }
},
```

```
    "VpcConfig": {  
        "SecurityGroupIds": [ "string" ],  
        "SubnetIds": [ "string" ]  
    }  
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

[Code \(p. 410\)](#)

The code for the function.

Type: [FunctionCode \(p. 534\)](#) object

Required: Yes

[DeadLetterConfig \(p. 410\)](#)

A dead letter queue configuration that specifies the queue or topic where Lambda sends asynchronous events when they fail processing. For more information, see [Dead Letter Queues](#).

Type: [DeadLetterConfig \(p. 528\)](#) object

Required: No

[Description \(p. 410\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 410\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 529\)](#) object

Required: No

[FunctionName \(p. 410\)](#)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: Yes

[Handler \(p. 410\)](#)

The name of the method within your code that Lambda calls to execute your function. The format includes the filename and can also include namespaces and other qualifiers, depending on the runtime. For more information, see [Programming Model](#).

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: Yes

[KMSKeyArn \(p. 410\)](#)

The ARN of the AWS Key Management Service key used to encrypt your function's environment variables. If not provided, AWS Lambda uses a default service key.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.]+|(.))`

Required: No

[Layers \(p. 410\)](#)

A list of [function layers](#) to add to the function's execution environment. Specify each layer by ARN, including the version.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+`

Required: No

[MemorySize \(p. 410\)](#)

The amount of memory that your function has access to. Increasing the function's memory also increases its CPU allocation. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

[Publish \(p. 410\)](#)

Set to true to publish the first version of the function during creation.

Type: Boolean

Required: No

[Role \(p. 410\)](#)

The Amazon Resource Name (ARN) of the function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+

Required: Yes

[Runtime \(p. 410\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

Required: Yes

[Tags \(p. 410\)](#)

A list of [tags](#) to apply to the function.

Type: String to string map

Required: No

[Timeout \(p. 410\)](#)

The amount of time that Lambda allows a function to run before terminating it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 410\)](#)

Set `Mode` to `Active` to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 546\)](#) object

Required: No

[VpcConfig \(p. 410\)](#)

For network connectivity to AWS resources in a VPC, specify a list of security groups and subnets in the VPC. When you connect a function to a VPC, it can only access resources and the internet through that VPC. For more information, see [VPC Settings](#).

Type: [VpcConfig \(p. 548\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```

```
{  
    "CodeSha256": "string",  
    "CodeSize": number,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    },  
    "Description": "string",  
    "Environment": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "Variables": {  
            "string" : "string"  
        }  
    },  
    "FunctionArn": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "KMSKeyArn": "string",  
    "LastModified": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "Timeout": number,  
    "TracingConfig": {  
        "Mode": "string"  
    },  
    "Version": "string",  
    "VpcConfig": {  
        "SecurityGroupIds": [ "string" ],  
        "SubnetIds": [ "string" ],  
        "VpcId": "string"  
    }  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CodeSha256](#) (p. 413)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize](#) (p. 413)

The size of the function's deployment package in bytes.

Type: Long

[DeadLetterConfig](#) (p. 413)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

[Description \(p. 413\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 413\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

[FunctionArn \(p. 413\)](#)

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 413\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 413\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 413\)](#)

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.:*)|()`

[LastModified \(p. 413\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 413\)](#)

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

[MasterArn \(p. 413\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 413\)](#)

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 413\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 413\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.+]`

[Runtime \(p. 413\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Timeout \(p. 413\)](#)

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 413\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

[Version \(p. 413\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 413\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

DeleteAlias

Deletes a Lambda function [alias](#).

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 418)

The name of the lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Name](#) (p. 418)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-]+$)([a-zA-Z0-9-_]+)`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

DeleteEventSourceMapping

Deletes an [event source mapping](#). You can get the identifier of a mapping from the output of [ListEventSourceMappings](#) (p. 462).

Request Syntax

```
DELETE /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID](#) (p. 420)

The identifier of the event source mapping.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize](#) (p. 420)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn](#) (p. 420)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9-]+)([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12}):(.*)

[FunctionArn \(p. 420\)](#)

The ARN of the Lambda function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 420\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 420\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 420\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 420\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 420\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceInUseException](#)

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

DeleteFunction

Deletes a Lambda function. To delete a specific function version, use the [Qualifier](#) parameter. Otherwise, all versions and aliases are deleted.

To delete Lambda event source mappings that invoke a function, use [DeleteEventSourceMapping \(p. 420\)](#). For AWS services and resources that invoke your function directly, delete the trigger in the service where you originally configured it.

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 423\)](#)

The name of the Lambda function or version.

Name formats

- Function name - my-function (name-only), my-function:1 (with version).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Qualifier \(p. 423\)](#)

Specify a version to delete. You cannot delete a version that is referenced by an alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

DeleteFunctionConcurrency

Removes a concurrent execution limit from a function.

Request Syntax

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 425)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_+]):(:(\\$LATEST|[a-zA-Z0-9-_+]))?

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

- HTTP Status Code: 404
ServiceException
 - The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
TooManyRequestsException
 - Request throughput limit exceeded.
- HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

DeleteLayerVersion

Deletes a version of an [AWS Lambda layer](#). Deleted versions can no longer be viewed or added to functions. To avoid breaking functions, a copy of the version remains in Lambda until no functions refer to it.

Request Syntax

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName](#) (p. 427)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+]| [a-zA-Z0-9-_+]

[VersionNumber](#) (p. 427)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetAccountSettings

Retrieves details about your account's [limits](#) and usage in a region.

Request Syntax

```
GET /2016-08-19/account-settings/ HTTP/1.1
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AccountLimit": {
        "CodeSizeUnzipped": number,
        "CodeSizeZipped": number,
        "ConcurrentExecutions": number,
        "TotalCodeSize": number,
        "UnreservedConcurrentExecutions": number
    },
    "AccountUsage": {
        "FunctionCount": number,
        "TotalCodeSize": number
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AccountLimit](#) (p. 429)

Limits related to concurrency and code storage.

Type: [AccountLimit](#) (p. 522) object

[AccountUsage](#) (p. 429)

The number of functions and amount of storage in use.

Type: [AccountUsage](#) (p. 523) object

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException
Request throughput limit exceeded.
HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetAlias

Returns details about a Lambda function [alias](#).

Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 431)

The name of the lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Name](#) (p. 431)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 431\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 431\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 431\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 431\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 431\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 431\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Errors

InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetEventSourceMapping

Returns details about an event source mapping. You can get the identifier of a mapping from the output of [ListEventSourceMappings \(p. 462\)](#).

Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 434\)](#)

The identifier of the event source mapping.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 434\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 434\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 434\)](#)

The ARN of the Lambda function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 434\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 434\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 434\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 434\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 434\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetFunction

Returns information about function or function version, with a link to download the deployment package that's valid for 10 minutes. If you specify a function version, only details specific to that version are returned.

Request Syntax

```
GET /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 437)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$\#LATEST|[a-zA-Z0-9-_]+))?

[Qualifier](#) (p. 437)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Code": {
    "Location": "string",
    "RepositoryType": "string"
  },
  "Concurrency": {
    "ReservedConcurrentExecutions": number
  },
  "Configuration": {
```

```
"CodeSha256": "string",
"CodeSize": number,
"DeadLetterConfig": {
    "TargetArn": "string"
},
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
},
"Tags": {
    "string" : "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Code \(p. 437\)](#)

The deployment package of the function or version.

Type: [FunctionCodeLocation \(p. 535\)](#) object

[Concurrency \(p. 437\)](#)

The function's reserved concurrency.

Type: [Concurrency \(p. 527\)](#) object

[Configuration \(p. 437\)](#)

The configuration of the function or version.

Type: [FunctionConfiguration \(p. 536\)](#) object

[Tags \(p. 437\)](#)

The function's [tags](#).

Type: String to string map

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetFunctionConfiguration

Returns a the version-specific settings of a Lambda function or version. The output includes only options that can vary between versions of a function. To modify these settings, use [UpdateFunctionConfiguration \(p. 514\)](#).

To get all of a function's details, including function-level settings, use [GetFunction \(p. 437\)](#).

Request Syntax

```
GET /2015-03-31/functions/FunctionName/configuration?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 440\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Qualifier \(p. 440\)](#)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
}
```

```
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 440\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 440\)](#)

The size of the function's deployment package in bytes.

Type: Long

[DeadLetterConfig \(p. 440\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

[Description \(p. 440\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 440\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

[FunctionArn \(p. 440\)](#)

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*):lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 440\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 440\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 440\)](#)

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*):[a-zA-Z0-9-_\.]+\.*|()`

[LastModified \(p. 440\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 440\)](#)

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

[MasterArn \(p. 440\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[MemorySize \(p. 440\)](#)

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 440\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 440\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+

[Runtime \(p. 440\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Timeout \(p. 440\)](#)

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 440\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

[Version \(p. 440\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 440\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetLayerVersion

Returns information about a version of an [AWS Lambda layer](#), with a link to download the layer archive that's valid for 10 minutes.

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 445\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|([a-zA-Z0-9-_]+)

[VersionNumber \(p. 445\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CompatibleRuntimes \(p. 445\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Content \(p. 445\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 543\)](#) object

[CreatedDate \(p. 445\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 445\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 445\)](#)

The ARN of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 445\)](#)

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 445\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 445\)](#)

The version number.

Type: Long

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetLayerVersionPolicy

Returns the permission policy for a version of an [AWS Lambda layer](#). For more information, see [AddLayerVersionPermission \(p. 394\)](#).

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber/policy HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 448\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_+]`)

[VersionNumber \(p. 448\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Policy": "string",
    "RevisionId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Policy \(p. 448\)](#)

The policy document.

Type: String

[RevisionId \(p. 448\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

GetPolicy

Returns the resource-based IAM policy for a function, version, or alias.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 450)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Qualifier (p. 450)

Specify a version or alias to get the policy for that resource.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$_.-]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Policy": "string",
    "RevisionId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Policy \(p. 450\)](#)

The resource-based policy.

Type: String

[RevisionId \(p. 450\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

Invoke

Invokes a Lambda function. You can invoke a function synchronously and wait for the response, or asynchronously. To invoke a function asynchronously, set `InvocationType` to `Event`.

For synchronous invocation, details about the function response, including errors, are included in the response body and headers. For either invocation type, you can find more information in the [execution log](#) and [trace](#). To record function errors for asynchronous invocations, configure your function with a [dead letter queue](#).

The status code in the API response does not reflect function errors. Error codes are reserved for errors that prevent your function from executing, such as permissions errors, [limit errors](#), or issues with your function's code and configuration. For example, Lambda returns `TooManyRequestsException` if executing the function would cause you to exceed a concurrency limit at either the account level (`ConcurrentInvocationLimitExceeded`) or function level (`ReservedFunctionConcurrentInvocationLimitExceeded`).

For functions with a long timeout, your client may be disconnected during synchronous invocation while it waits for a response. Configure your HTTP client, SDK, firewall, proxy, or operating system to allow for long connections with timeout or keep-alive settings.

This operation requires permission for the `lambda:InvokeFunction` action.

Request Syntax

```
POST /2015-03-31/functions/FunctionName/invocations?Qualifier=Qualifier HTTP/1.1
X-Amz-Invocation-Type: InvocationType
X-Amz-Log-Type: LogType
X-Amz-Client-Context: ClientContext

Payload
```

URI Request Parameters

The request requires the following URI parameters.

[ClientContext](#) (p. 452)

Up to 3583 bytes of base64-encoded data about the invoking client to pass to the function in the context object.

[FunctionName](#) (p. 452)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[InvocationType \(p. 452\)](#)

Choose from the following options.

- `RequestResponse` (default) - Invoke the function synchronously. Keep the connection open until the function returns a response or times out. The API response includes the function response and additional data.
- `Event` - Invoke the function asynchronously. Send events that fail multiple times to the function's dead-letter queue (if configured). The API response only includes a status code.
- `DryRun` - Validate parameter values and verify that the user or role has permission to invoke the function.

Valid Values: `Event` | `RequestResponse` | `DryRun`

[LogType \(p. 452\)](#)

Set to `Tail` to include the execution log in the response.

Valid Values: `None` | `Tail`

[Qualifier \(p. 452\)](#)

Specify a version or alias to invoke a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(|[a-zA-Z0-9$_-]+)`

Request Body

The request accepts the following binary data.

[Payload \(p. 452\)](#)

JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 StatusCode
X-Amz-Function-Error: FunctionError
X-Amz-Log-Result: LogResult
X-Amz-Executed-Version: ExecutedVersion

Payload
```

Response Elements

If the action is successful, the service sends back the following HTTP response.

[StatusCode \(p. 453\)](#)

The HTTP status code will be in the 200 range for successful request. For the `RequestResponse` invocation type this status code will be 200. For the `Event` invocation type this status code will be 202. For the `DryRun` invocation type the status code will be 204.

The response returns the following HTTP headers.

[ExecutedVersion \(p. 453\)](#)

The version of the function that executed. When you invoke a function with an alias, indicates which version the alias resolved to.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[FunctionError \(p. 453\)](#)

If present, indicates that an error occurred during function execution. Details about the error are included in the response payload.

- **Handled** - The runtime caught an error thrown by the function and formatted it into a JSON document.
- **Unhandled** - The runtime did not handle the error. For example, the function ran out of memory or timed out.

[LogResult \(p. 453\)](#)

The last 4 KB of the execution log, base64 encoded.

The response returns the following as the HTTP body.

[Payload \(p. 453\)](#)

The response from the function, or an error object.

Errors

[EC2AccessDeniedException](#)

Need additional permissions to configure VPC settings.

HTTP Status Code: 502

[EC2ThrottledException](#)

AWS Lambda was throttled by Amazon EC2 during Lambda function initialization using the execution role provided for the Lambda function.

HTTP Status Code: 502

[EC2UnexpectedException](#)

AWS Lambda received an unexpected EC2 client exception while setting up for the Lambda function.

HTTP Status Code: 502

[ENILimitReachedException](#)

AWS Lambda was not able to create an Elastic Network Interface (ENI) in the VPC, specified as part of Lambda function configuration, because the limit for network interfaces has been reached.

HTTP Status Code: 502

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

InvalidRuntimeException

The runtime or runtime version specified is not supported.

HTTP Status Code: 502

InvalidSecurityGroupIDException

The Security Group ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidSubnetIDException

The Subnet ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidZipFileException

AWS Lambda could not unzip the deployment package.

HTTP Status Code: 502

KMSAccessDeniedException

Lambda was unable to decrypt the environment variables because KMS access was denied. Check the Lambda function's KMS permissions.

HTTP Status Code: 502

KMSDisabledException

Lambda was unable to decrypt the environment variables because the KMS key used is disabled. Check the Lambda function's KMS key settings.

HTTP Status Code: 502

KMSInvalidStateException

Lambda was unable to decrypt the environment variables because the KMS key used is in an invalid state for Decrypt. Check the function's KMS key settings.

HTTP Status Code: 502

KMSNotFoundException

Lambda was unable to decrypt the environment variables because the KMS key was not found. Check the function's KMS key settings.

HTTP Status Code: 502

RequestTooLargeException

The request payload exceeded the `Invoke` request body JSON input limit. For more information, see [Limits](#).

HTTP Status Code: 413

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
SubnetIPAddressLimitReachedException

AWS Lambda was not able to set up VPC access for the Lambda function because one or more configured subnets has no available IP addresses.

HTTP Status Code: 502
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429
UnsupportedMediaTypeException

The content type of the `Invoke` request body is not JSON.

HTTP Status Code: 415

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

InvokeAsync

This action has been deprecated.

Important

For asynchronous function invocation, use [Invoke \(p. 452\)](#).

Invokes a function asynchronously.

Request Syntax

```
POST /2014-11-13/functions/FunctionName/invoke-async/ HTTP/1.1  
InvokeArgs
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 457)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Request Body

The request accepts the following binary data.

InvokeArgs (p. 457)

JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 Status
```

Response Elements

If the action is successful, the service sends back the following HTTP response.

[Status \(p. 457\)](#)

It will be 202 upon success.

Errors

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

InvalidRuntimeException

The runtime or runtime version specified is not supported.

HTTP Status Code: 502

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListAliases

Returns a list of [aliases](#) for a Lambda function.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases?  
FunctionVersion=FunctionVersion&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 459)

The name of the lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionVersion](#) (p. 459)

Specify a function version to only list aliases that invoke that version.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[Marker](#) (p. 459)

Specify the pagination token returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 459)

Limit the number of aliases returned.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json
```

```
{  
    "Aliases": [  
        {  
            "AliasArn": "string",  
            "Description": "string",  
            "FunctionVersion": "string",  
            "Name": "string",  
            "RevisionId": "string",  
            "RoutingConfig": {  
                "AdditionalVersionWeights": {  
                    "string" : number  
                }  
            }  
        }  
    ],  
    "NextMarker": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Aliases \(p. 459\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 524\)](#) objects

[NextMarker \(p. 459\)](#)

Pagination token included if more results are available.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListEventSourceMappings

Lists event source mappings. Specify an `EventSourceArn` to only show event source mappings for a single event source.

Request Syntax

```
GET /2015-03-31/event-source-mappings/?  
EventSourceArn=EventSourceArn&FunctionName=FunctionName&Marker=Marker&MaxItems=MaxItems  
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[EventSourceArn \(p. 462\)](#)

The Amazon Resource Name (ARN) of the event source.

- Amazon Kinesis - The ARN of the data stream or a stream consumer.
- Amazon DynamoDB Streams - The ARN of the stream.
- Amazon Simple Queue Service - The ARN of the queue.

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9-]+)([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*?)`

[FunctionName \(p. 462\)](#)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Version or Alias ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Marker \(p. 462\)](#)

A pagination token returned by a previous call.

[MaxItems \(p. 462\)](#)

The maximum number of event source mappings to return.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "EventSourceMappings": [
    {
      "BatchSize": number,
      "EventSourceArn": "string",
      "FunctionArn": "string",
      "LastModified": number,
      "LastProcessingResult": "string",
      "State": "string",
      "StateTransitionReason": "string",
      "UUID": "string"
    }
  ],
  "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[EventSourceMappings \(p. 463\)](#)

A list of event source mappings.

Type: Array of [EventSourceMappingConfiguration \(p. 532\)](#) objects

[NextMarker \(p. 463\)](#)

A pagination token that's returned when the response doesn't contain all event source mappings.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListFunctions

Returns a list of Lambda functions, with the version-specific configuration of each.

Set `FunctionVersion` to `ALL` to include all published versions of each function in addition to the unpublished version. To get more information about a function or version, use [GetFunction \(p. 437\)](#).

Request Syntax

```
GET /2015-03-31/functions/?  
FunctionVersion=FunctionVersion&Marker=Marker&MasterRegion=MasterRegion&MaxItems=MaxItems  
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionVersion \(p. 465\)](#)

Set to `ALL` to include entries for all published versions of each function.

Valid Values: `ALL`

[Marker \(p. 465\)](#)

Specify the pagination token returned by a previous request to retrieve the next page of results.

[MasterRegion \(p. 465\)](#)

For Lambda@Edge functions, the region of the master function. For example, `us-east-2` or `ALL`. If specified, you must set `FunctionVersion` to `ALL`.

Pattern: `ALL | [a-z]{2}(-gov)?-[a-z]+-\d{1}`

[MaxItems \(p. 465\)](#)

Specify a value between 1 and 50 to limit the number of functions in the response.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "Functions": [  
        {  
            "CodeSha256": "string",  
            "CodeSize": number,  
            "DeadLetterConfig": {  
                "TargetArn": "string"  
            },  
            "Description": "string",  
            "Environment": {  
                "Error": {  
                    "Code": "string",  
                    "Message": "string"  
                }  
            },  
            "FunctionArn": "string",  
            "Handler": "string",  
            "LastModified": "string",  
            "MemorySize": number,  
            "Runtime": "string",  
            "State": "string",  
            "Timeout": number  
        }  
    ]  
}
```

```
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
],
"NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Functions \(p. 465\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 536\)](#) objects

[NextMarker \(p. 465\)](#)

Pagination token included if more results are available.

Type: String

Errors

InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListLayers

Lists [AWS Lambda layers](#) and shows information about the latest version of each. Specify a [runtime identifier](#) to list only layers that indicate that they're compatible with that runtime.

Request Syntax

```
GET /2018-10-31/layers?CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

CompatibleRuntime ([p. 468](#))

A runtime identifier. For example, go1.x.

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

Marker ([p. 468](#))

A pagination token returned by a previous call.

MaxItems ([p. 468](#))

The maximum number of layers to return.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Layers": [
    {
      "LatestMatchingVersion": {
        "CompatibleRuntimes": [ "string" ],
        "CreatedDate": "string",
        "Description": "string",
        "LayerVersionArn": "string",
        "LicenseInfo": "string",
        "Version": number
      },
      "LayerArn": "string",
      "LayerName": "string"
    }
  ],
  "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Layers \(p. 468\)](#)

A list of function layers.

Type: Array of [LayersListItem \(p. 541\)](#) objects

[NextMarker \(p. 468\)](#)

A pagination token returned when the response doesn't contain all layers.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListLayerVersions

Lists the versions of an [AWS Lambda layer](#). Versions that have been deleted aren't listed. Specify a [runtime identifier](#) to list only versions that indicate that they're compatible with that runtime.

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions?  
CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[CompatibleRuntime](#) (p. 470)

A runtime identifier. For example, go1.x.

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[LayerName](#) (p. 470)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\d{12}:layer:[a-zA-Z0-9-_+])|[a-zA-Z0-9-_+]

[Marker](#) (p. 470)

A pagination token returned by a previous call.

[MaxItems](#) (p. 470)

The maximum number of versions to return.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "LayerVersions": [  
        {  
            "CompatibleRuntimes": [ "string" ],  
            "CreatedDate": "string",  
            "Description": "string",  
            "LayerVersionArn": "string",  
            "LicenseInfo": "string",  
            "LastModified": "string",  
            "LayerVersionId": "string",  
            "Sha256": "string",  
            "Size": 123,  
            "Status": "string",  
            "StatusReason": "string",  
            "Version": 123  
        }  
    ]  
}
```

```
        "Version": number
    }
],
"NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[LayerVersions \(p. 470\)](#)

A list of versions.

Type: Array of [LayerVersionsListItem \(p. 544\)](#) objects

[NextMarker \(p. 470\)](#)

A pagination token returned when the response doesn't contain all versions.

Type: String

Errors

[InvalidOperationException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListTags

Returns a function's [tags](#). You can also view tags with [GetFunction](#) (p. 437).

Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Resource](#) (p. 473)

The function's Amazon Resource Name (ARN).

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\${LATEST|[a-zA-Z0-9-_]+))?

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Tags": [
    {
      "string" : "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Tags](#) (p. 473)

The function's tags.

Type: String to string map

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the [CreateFunction](#) or the [UpdateFunctionConfiguration](#) API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

ListVersionsByFunction

Returns a list of [versions](#), with the version-specific configuration of each.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 475)

The name of the lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Marker](#) (p. 475)

Specify the pagination token returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 475)

Limit the number of versions returned.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "NextMarker": "string",
    "Versions": [
        {
            "CodeSha256": "string",
            "CodeSize": number,
            "DeadLetterConfig": {
                "TargetArn": "string"
            },
            "Description": "string",
            "Environment": {
```

```
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
]
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[NextMarker \(p. 475\)](#)

Pagination token included if more results are available.

Type: String

[Versions \(p. 475\)](#)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration \(p. 536\)](#) objects

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

PublishLayerVersion

Creates an [AWS Lambda layer](#) from a ZIP archive. Each time you call `PublishLayerVersion` with the same version name, a new version is created.

Add layers to your function with [CreateFunction \(p. 410\)](#) or [UpdateFunctionConfiguration \(p. 514\)](#).

Request Syntax

```
POST /2018-10-31/layers/LayerName/versions HTTP/1.1
Content-type: application/json

{
  "CompatibleRuntimes": [ "string" ],
  "Content": {
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string",
    "ZipFile": blob
  },
  "Description": "string",
  "LicenseInfo": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 478\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_+]`)

Request Body

The request accepts the following data in JSON format.

[CompatibleRuntimes \(p. 478\)](#)

A list of compatible [function runtimes](#). Used for filtering with [ListLayers \(p. 468\)](#) and [ListLayerVersions \(p. 470\)](#).

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: `nodejs` | `nodejs4.3` | `nodejs6.10` | `nodejs8.10` | `java8` | `python2.7` | `python3.6` | `python3.7` | `dotnetcore1.0` | `dotnetcore2.0` | `dotnetcore2.1` | `nodejs4.3-edge` | `go1.x` | `ruby2.5` | `provided`

Required: No

[Content \(p. 478\)](#)

The function layer archive.

Type: [LayerVersionContentInput \(p. 542\)](#) object

Required: Yes

[Description \(p. 478\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[LicenseInfo \(p. 478\)](#)

The layer's software license. It can be any of the following:

- An [SPDX license identifier](#). For example, `MIT`.
- The URL of a license hosted on the internet. For example, <https://opensource.org/licenses/MIT>.
- The full text of the license.

Type: String

Length Constraints: Maximum length of 512.

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CompatibleRuntimes \(p. 479\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Content \(p. 479\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 543\)](#) object

[CreatedDate \(p. 479\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 479\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 479\)](#)

The ARN of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 479\)](#)

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 479\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 479\)](#)

The version number.

Type: Long

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400
`InvalidParameterValueException`

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400
`ResourceNotFoundException`

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
`TooManyRequestsException`

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

PublishVersion

Creates a [version](#) from the current code and configuration of a function. Use versions to create a snapshot of your function code and configuration that doesn't change.

AWS Lambda does not publish a version if the function's configuration and code hasn't changed since the last version. Use [UpdateFunctionCode \(p. 508\)](#) or [UpdateFunctionConfiguration \(p. 514\)](#) to update the function prior to publishing a version.

Clients can invoke versions directly or with an alias. To create an alias, use [CreateAlias \(p. 402\)](#).

Request Syntax

```
POST /2015-03-31/functions/FunctionName/versions HTTP/1.1
Content-type: application/json

{
  "CodeSha256": "string",
  "Description": "string",
  "RevisionId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 482\)](#)

The name of the lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Request Body

The request accepts the following data in JSON format.

[CodeSha256 \(p. 482\)](#)

Only publish a version if the hash matches the value specified. Use this option to avoid publishing a version if the function code has changed since you last updated it. You can get the hash for the version you uploaded from the output of [UpdateFunctionCode \(p. 508\)](#).

Type: String

Required: No

Description (p. 482)

Specify a description for the version to override the description in the function configuration.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

RevisionId (p. 482)

Only update the function if the revision ID matches the ID specified. Use this option to avoid publishing a version if the function configuration has changed since you last updated it.

Type: String

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ]
    }
}
```

```
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 483\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 483\)](#)

The size of the function's deployment package in bytes.

Type: Long

[DeadLetterConfig \(p. 483\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

[Description \(p. 483\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 483\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

[FunctionArn \(p. 483\)](#)

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 483\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?\d{12}:?(function:)?([a-zA-Z0-9-_\.]+)(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 483\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

[KMSKeyArn \(p. 483\)](#)

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-z0-9-.]+:[.*])|()

[LastModified \(p. 483\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 483\)](#)

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

[MasterArn \(p. 483\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[MemorySize \(p. 483\)](#)

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 483\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 483\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.]+

[Runtime \(p. 483\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Timeout \(p. 483\)](#)

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 483\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

[Version \(p. 483\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

[VpcConfig \(p. 483\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Errors

[CodeStorageExceeded](#)[Exception](#)

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

[InvalidParameterValue](#)[Exception](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[PreconditionFailed](#)[Exception](#)

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

[ResourceNotFound](#)[Exception](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[Service](#)[Exception](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

PutFunctionConcurrency

Sets the maximum number of simultaneous executions for a function, and reserves capacity for that concurrency level.

Concurrency settings apply to the function as a whole, including all published versions and the unpublished version. Reserving concurrency both guarantees that your function has capacity to process the specified number of events simultaneously, and prevents it from scaling beyond that level. Use [GetFunction \(p. 437\)](#) to see the current setting for a function.

Use [GetAccountSettings \(p. 429\)](#) to see your regional concurrency limit. You can reserve concurrency for as many functions as you like, as long as you leave at least 100 simultaneous executions unreserved for functions that aren't configured with a per-function limit. For more information, see [Managing Concurrency](#).

Request Syntax

```
PUT /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 488\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Request Body

The request accepts the following data in JSON format.

[ReservedConcurrentExecutions \(p. 488\)](#)

The number of simultaneous executions to reserve for the function.

Type: Integer

Valid Range: Minimum value of 0.

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[ReservedConcurrentExecutions \(p. 489\)](#)

The number of concurrent executions reserved for this function. For more information, see [Managing Concurrency](#).

Type: Integer

Valid Range: Minimum value of 0.

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

`ResourceNotFoundException`

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

`TooManyRequestsException`

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

RemoveLayerVersionPermission

Removes a statement from the permissions policy for a version of an [AWS Lambda layer](#). For more information, see [AddLayerVersionPermission \(p. 394\)](#).

Request Syntax

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber/policy/StatementId?  
RevisionId=RevisionId HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 491\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+])|([a-zA-Z0-9-_]+)

[RevisionId \(p. 491\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

[StatementId \(p. 491\)](#)

The identifier that was specified when the statement was added.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

[VersionNumber \(p. 491\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400
PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

RemovePermission

Revokes function use permission from an AWS service or another account. You can get the ID of the statement from the output of [GetPolicy \(p. 450\)](#).

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/policy/StatementId?  
Qualifier=Qualifier&RevisionId=RevisionId HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 493\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:(aws[a-zA-Z-]*):lambda:`)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Qualifier \(p. 493\)](#)

Specify a version or alias to remove permissions from a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

[RevisionId \(p. 493\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

[StatementId \(p. 493\)](#)

Statement ID of the permission to remove.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

TagResource

Adds [tags](#) to a function.

Request Syntax

```
POST /2017-03-31/tags/ARN HTTP/1.1
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[Resource \(p. 496\)](#)

The function's Amazon Resource Name (ARN).

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\#LATEST|[a-zA-Z0-9-_]+))?

Request Body

The request accepts the following data in JSON format.

[Tags \(p. 496\)](#)

A list of tags to apply to the function.

Type: String to string map

Required: Yes

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UntagResource

Removes [tags](#) from a function.

Request Syntax

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Resource \(p. 498\)](#)

The function's Amazon Resource Name (ARN).

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_+]))?

[TagKeys \(p. 498\)](#)

A list of tag keys to remove from the function.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the [CreateFunction](#) or the [UpdateFunctionConfiguration](#) API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException
Request throughput limit exceeded.
HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UpdateAlias

Updates the configuration of a Lambda function [alias](#).

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "RevisionId": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 500)

The name of the lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Name](#) (p. 500)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-]+$)([a-zA-Z0-9-_]+)`

Request Body

The request accepts the following data in JSON format.

[Description](#) (p. 500)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 500\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

Required: No

[RevisionId \(p. 500\)](#)

Only update the alias if the revision ID matches the ID specified. Use this option to avoid modifying an alias that has changed since you last read it.

Type: String

Required: No

[RoutingConfig \(p. 500\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 501\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 501\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 501\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 501\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 501\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 501\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UpdateEventSourceMapping

Updates an event source mapping. You can change the function that AWS Lambda invokes, or pause invocation and resume later from the same location.

Request Syntax

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "Enabled": boolean,
    "FunctionName": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 504\)](#)

The identifier of the event source mapping.

Request Body

The request accepts the following data in JSON format.

[BatchSize \(p. 504\)](#)

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

[Enabled \(p. 504\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

[FunctionName \(p. 504\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.

- Version or Alias ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 505\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 505\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z){2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 505\)](#)

The ARN of the Lambda function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 505\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 505\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 505\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 505\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 505\)](#)

The identifier of the event source mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the CreateFunction or the UpdateFunctionConfiguration API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceInUseException

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UpdateFunctionCode

Updates a Lambda function's code.

The function's code is locked when you publish a version. You cannot modify the code of a published version, only the unpublished version.

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/code HTTP/1.1
Content-type: application/json

{
  "DryRun": boolean,
  "Publish": boolean,
  "RevisionId": "string",
  "S3Bucket": "string",
  "S3Key": "string",
  "S3ObjectVersion": "string",
  "ZipFile": blob
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 508\)](#)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*):lambda:)([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Request Body

The request accepts the following data in JSON format.

[DryRun \(p. 508\)](#)

Set to true to validate the request parameters and access permissions without modifying the function code.

Type: Boolean

Required: No

[Publish \(p. 508\)](#)

Set to true to publish a new version of the function after updating the code. This has the same effect as calling [PublishVersion \(p. 482\)](#) separately.

Type: Boolean

Required: No

[RevisionId \(p. 508\)](#)

Only update the function if the revision ID matches the ID specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[S3Bucket \(p. 508\)](#)

An Amazon S3 bucket in the same region as your function. The bucket can be in a different AWS account.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.).\$

Required: No

[S3Key \(p. 508\)](#)

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[S3ObjectVersion \(p. 508\)](#)

For versioned objects, the version of the deployment package object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[ZipFile \(p. 508\)](#)

The base64-encoded contents of the deployment package. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```

```
{  
    "CodeSha256": "string",  
    "CodeSize": number,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    },  
    "Description": "string",  
    "Environment": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "Variables": {  
            "string" : "string"  
        }  
    },  
    "FunctionArn": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "KMSKeyArn": "string",  
    "LastModified": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "Timeout": number,  
    "TracingConfig": {  
        "Mode": "string"  
    },  
    "Version": "string",  
    "VpcConfig": {  
        "SecurityGroupIds": [ "string" ],  
        "SubnetIds": [ "string" ],  
        "VpcId": "string"  
    }  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256](#) (p. 509)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize](#) (p. 509)

The size of the function's deployment package in bytes.

Type: Long

[DeadLetterConfig](#) (p. 509)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

[Description \(p. 509\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 509\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

[FunctionArn \(p. 509\)](#)

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 509\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 509\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 509\)](#)

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.:*)|()`

[LastModified \(p. 509\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 509\)](#)

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

[MasterArn \(p. 509\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 509\)](#)

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 509\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 509\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.]+`

[Runtime \(p. 509\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Timeout \(p. 509\)](#)

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 509\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

[Version \(p. 509\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 509\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

UpdateFunctionConfiguration

Modify the version-specific settings of a Lambda function.

These settings can vary between versions of a function and are locked when you publish a version. You cannot modify the configuration of a published version, only the unpublished version.

To configure function concurrency, use [PutFunctionConcurrency \(p. 488\)](#). To grant invoke permissions to an account or AWS service, use [AddPermission \(p. 398\)](#).

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/configuration HTTP/1.1
Content-type: application/json

{
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string": "string"
    }
  },
  "Handler": "string",
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "RevisionId": "string",
  "Role": "string",
  "Runtime": "string",
  "Timeout": number,
  "TracingConfig": {
    "Mode": "string"
  },
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ]
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 514\)](#)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Request Body

The request accepts the following data in JSON format.

[DeadLetterConfig \(p. 514\)](#)

A dead letter queue configuration that specifies the queue or topic where Lambda sends asynchronous events when they fail processing. For more information, see [Dead Letter Queues](#).

Type: [DeadLetterConfig \(p. 528\)](#) object

Required: No

[Description \(p. 514\)](#)

A short user-defined function description. AWS Lambda does not use this value. Assign a meaningful description as you see fit.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 514\)](#)

The parent object that contains your environment's configuration settings.

Type: [Environment \(p. 529\)](#) object

Required: No

[Handler \(p. 514\)](#)

The function that Lambda calls to begin executing your function. For Node.js, it is the `module-name.export` value in your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: No

[KMSKeyArn \(p. 514\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If you elect to use the AWS Lambda default service key, pass in an empty string ("") for this parameter.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.]+|()`

Required: No

[Layers \(p. 514\)](#)

A list of [function layers](#) to add to the function's execution environment.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Required: No

[MemorySize \(p. 514\)](#)

The amount of memory, in MB, your Lambda function is given. AWS Lambda uses this memory size to infer the amount of CPU allocated to your function. Your function use-case determines your CPU and memory requirements. For example, a database operation might need less memory compared to an image processing function. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

[RevisionId \(p. 514\)](#)

Only update the function if the revision ID matches the ID specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[Role \(p. 514\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda will assume when it executes your function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+

Required: No

[Runtime \(p. 514\)](#)

The runtime version for the function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

Required: No

[Timeout \(p. 514\)](#)

The amount of time that Lambda allows a function to run before terminating it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 514\)](#)

Set Mode to Active to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 546\)](#) object

Required: No

[VpcConfig \(p. 514\)](#)

Specify security groups and subnets in a VPC to which your Lambda function needs access.

Type: [VpcConfig \(p. 548\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
}
```

}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 517\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 517\)](#)

The size of the function's deployment package in bytes.

Type: Long

[DeadLetterConfig \(p. 517\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

[Description \(p. 517\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 517\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

[FunctionArn \(p. 517\)](#)

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 517\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?\d{12}:(function:)?([a-zA-Z0-9-_\.]+)(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 517\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

[KMSKeyArn \(p. 517\)](#)

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.]+|()

[LastModified \(p. 517\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 517\)](#)

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

[MasterArn \(p. 517\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-zA-Z]{2}(-gov)?-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[MemorySize \(p. 517\)](#)

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 517\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 517\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/_-]+

[Runtime \(p. 517\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

[Timeout \(p. 517\)](#)

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 517\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

[Version \(p. 517\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 517\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Errors

InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

Data Types

The following data types are supported:

- [AccountLimit \(p. 522\)](#)
- [AccountUsage \(p. 523\)](#)
- [AliasConfiguration \(p. 524\)](#)
- [AliasRoutingConfiguration \(p. 526\)](#)
- [Concurrency \(p. 527\)](#)
- [DeadLetterConfig \(p. 528\)](#)
- [Environment \(p. 529\)](#)
- [EnvironmentError \(p. 530\)](#)
- [EnvironmentResponse \(p. 531\)](#)
- [EventSourceMappingConfiguration \(p. 532\)](#)
- [FunctionCode \(p. 534\)](#)
- [FunctionCodeLocation \(p. 535\)](#)
- [FunctionConfiguration \(p. 536\)](#)
- [Layer \(p. 540\)](#)
- [LayersListItem \(p. 541\)](#)
- [LayerVersionContentInput \(p. 542\)](#)
- [LayerVersionContentOutput \(p. 543\)](#)
- [LayerVersionsListItem \(p. 544\)](#)
- [TracingConfig \(p. 546\)](#)
- [TracingConfigResponse \(p. 547\)](#)
- [VpcConfig \(p. 548\)](#)
- [VpcConfigResponse \(p. 549\)](#)

AccountLimit

Limits related to concurrency and code storage. All file and storage sizes are in bytes.

Contents

CodeSizeUnzipped

The maximum size of your function's code and layers when extracted.

Type: Long

Required: No

CodeSizeZipped

The maximum size of a deployment package when uploaded directly to AWS Lambda. Use Amazon S3 for larger files.

Type: Long

Required: No

ConcurrentExecutions

The maximum number of simultaneous function executions.

Type: Integer

Required: No

TotalCodeSize

The amount of storage space that you can use for all deployment packages and layer archives.

Type: Long

Required: No

UnreservedConcurrentExecutions

The maximum number of simultaneous function executions, less the concurrency reserved for individual functions with [PutFunctionConcurrency \(p. 488\)](#).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

AccountUsage

The number of functions and amount of storage in use.

Contents

FunctionCount

The number of Lambda functions.

Type: Long

Required: No

TotalCodeSize

The amount of storage space, in bytes, in use by deployment packages and layer archives.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

AliasConfiguration

Provides configuration information about a Lambda function [alias](#).

Contents

AliasArn

The Amazon Resource Name (ARN) of the alias.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

Required: No

Description

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

FunctionVersion

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$\\$LATEST|[0-9]+)

Required: No

Name

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Required: No

RevisionId

A unique identifier that changes when you update the alias.

Type: String

Required: No

RoutingConfig

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 526\)](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

AliasRoutingConfiguration

A Lambda function alias's [traffic shifting](#) configuration.

Contents

AdditionalVersionWeights

The name of the second alias, and the percentage of traffic that is routed to it.

Type: String to double map

Key Length Constraints: Minimum length of 1. Maximum length of 1024.

Key Pattern: [0–9]⁺

Valid Range: Minimum value of 0.0. Maximum value of 1.0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

Concurrency

Contents

ReservedConcurrentExecutions

The number of concurrent executions reserved for this function. For more information, see [Managing Concurrency](#).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

DeadLetterConfig

The [dead letter queue](#) for failed asynchronous invocations.

Contents

TargetArn

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.*])|()`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

Environment

A function's environment variable settings.

Contents

Variables

Environment variable key-value pairs.

Type: String to string map

Key Pattern: [a-zA-Z]([a-zA-Z0-9_])+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

EnvironmentError

Error messages for environment variables that could not be applied.

Contents

ErrorCode

The error code.

Type: String

Required: No

Message

The error message.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

EnvironmentResponse

The results of a configuration update that applied environment variables.

Contents

Error

Error messages for environment variables that could not be applied.

Type: [EnvironmentError \(p. 530\)](#) object

Required: No

Variables

Environment variable key-value pairs.

Type: String to string map

Key Pattern: [a-zA-Z]([a-zA-Z0-9_])⁺

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

EventSourceMappingConfiguration

A mapping between an AWS resource and an AWS Lambda function. See [CreateEventSourceMapping \(p. 406\)](#) for details.

Contents

BatchSize

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

EventSourceArn

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*?)`

Required: No

FunctionArn

The ARN of the Lambda function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

LastModified

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

Required: No

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

Required: No

State

The state of the event source mapping. It can be one of the following: [Creating](#), [Enabling](#), [Enabled](#), [Disabling](#), [Disabled](#), [Updating](#), or [Deleting](#).

Type: String

Required: No

StateTransitionReason

The cause of the last state change, either `User initiated` or `Lambda initiated`.

Type: String

Required: No

UUID

The identifier of the event source mapping.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

FunctionCode

The code for the Lambda function. You can specify either an object in Amazon S3, or upload a deployment package directly.

Contents

S3Bucket

An Amazon S3 bucket in the same region as your function. The bucket can be in a different AWS account.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.)\$

Required: No

S3Key

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

S3ObjectVersion

For versioned objects, the version of the deployment package object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

ZipFile

The base64-encoded contents of the deployment package. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

FunctionCodeLocation

Details about a function's deployment package.

Contents

Location

A pre-signed URL that you can use to download the deployment package.

Type: String

Required: No

RepositoryType

The service hosting the file.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

FunctionConfiguration

Details about a function's configuration.

Contents

CodeSha256

The SHA256 hash of the function's deployment package.

Type: String

Required: No

CodeSize

The size of the function's deployment package in bytes.

Type: Long

Required: No

DeadLetterConfig

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 528\)](#) object

Required: No

Description

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Environment

The function's environment variables.

Type: [EnvironmentResponse \(p. 531\)](#) object

Required: No

FunctionArn

The function's Amazon Resource Name (ARN).

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

FunctionName

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

Handler

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: No

KMSKeyArn

The KMS key used to encrypt the function's environment variables. Only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+\.:*)|()`

Required: No

LastModified

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

Required: No

Layers

The function's [layers](#)

Type: Array of [Layer \(p. 540\)](#) objects

Required: No

MasterArn

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-zA-Z]{2}(-gov)?-[a-zA-Z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

MemorySize

The memory allocated to the function

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

RevisionId

Represents the latest updated revision of the function or alias.

Type: String

Required: No

Role

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+`

Required: No

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided`

Required: No

Timeout

The amount of time that Lambda allows a function to run before terminating it.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

TracingConfig

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 547\)](#) object

Required: No

Version

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

Required: No

VpcConfig

The function's networking configuration.

Type: [VpcConfigResponse \(p. 549\)](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

Layer

An [AWS Lambda layer](#).

Contents

Arn

The Amazon Resource Name (ARN) of the function layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\d{12}:layer:[a-zA-Z0-9-_+][0-9]+`

Required: No

CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

LayersListItem

Details about an [AWS Lambda layer](#).

Contents

LatestMatchingVersion

The newest version of the layer.

Type: [LayerVersionsListItem \(p. 544\)](#) object

Required: No

LayerArn

The Amazon Resource Name (ARN) of the function layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

Required: No

LayerName

The name of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+)|[a-zA-Z0-9-_]+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

LayerVersionContentInput

A ZIP archive that contains the contents of an [AWS Lambda layer](#). You can specify either an Amazon S3 location, or upload a layer archive directly.

Contents

S3Bucket

The Amazon S3 bucket of the layer archive.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.\.)\$

Required: No

S3Key

The Amazon S3 key of the layer archive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

S3ObjectVersion

For versioned objects, the version of the layer archive object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

ZipFile

The base64-encoded contents of the layer archive. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

LayerVersionContentOutput

Details about a version of an [AWS Lambda layer](#).

Contents

CodeSha256

The SHA-256 hash of the layer archive.

Type: String

Required: No

CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

Location

A link to the layer archive in Amazon S3 that is valid for 10 minutes.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

LayerVersionsListItem

Details about a version of an [AWS Lambda layer](#).

Contents

CompatibleRuntimes

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | nodejs4.3-edge | go1.x | ruby2.5 | provided

Required: No

CreatedDate

The date that the version was created, in ISO 8601 format. For example, 2018-11-27T15:10:45.123+0000.

Type: String

Required: No

Description

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

LayerVersionArn

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Required: No

LicensesInfo

The layer's open-source license.

Type: String

Length Constraints: Maximum length of 512.

Required: No

Version

The version number.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

TracingConfig

The function's AWS X-Ray tracing configuration.

Contents

Mode

The tracing mode.

Type: String

Valid Values: Active | PassThrough

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

TracingConfigResponse

The function's AWS X-Ray tracing configuration.

Contents

Mode

The tracing mode.

Type: String

Valid Values: Active | PassThrough

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

VpcConfig

The VPC security groups and subnets attached to a Lambda function.

Contents

SecurityGroupIds

A list of VPC security groups IDs.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of VPC subnet IDs.

Type: Array of strings

Array Members: Maximum number of 16 items.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

VpcConfigResponse

The VPC security groups and subnets attached to a Lambda function.

Contents

SecurityGroupIds

A list of VPC security groups IDs.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of VPC subnet IDs.

Type: Array of strings

Array Members: Maximum number of 16 items.

Required: No

VpcId

The ID of the VPC.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

Erreurs de certificat lors de l'utilisation d'un kit SDK

Dans la mesure où les kits AWS SDK utilisent les certificats de CA de votre ordinateur, les modifications apportées aux certificats sur les serveurs AWS peuvent entraîner des problèmes de connexion lorsque vous tentez d'utiliser un kit de développement logiciel (SDK). Vous pouvez empêcher ces défaillances en conservant les certificats de CA de votre ordinateur et le système d'exploitation à jour. Si vous rencontrez ce problème dans un environnement d'entreprise et que vous ne gérez pas votre propre ordinateur, vous pourrez être amené à demander à un administrateur de vous aider pour effectuer la mise à jour. La liste suivante présente les versions minimales requises pour le système d'exploitation et Java :

- Les versions de Microsoft Windows qui incluent des mises à jour datant de janvier 2005 et après contiennent au moins l'une des autorités de certification requises dans leur liste d'approbation.

- Mac OS X 10.4 avec Java pour Mac OS X 10.4 version 5 (février 2007), Mac OS X 10.5 (octobre 2007) et les versions ultérieures contiennent au moins l'une des CA requises dans leur liste d'approbation.
- Red Hat Enterprise Linux 5 (mars 2007), 6 et 7 et CentOS 5, 6 et 7 contiennent tous au moins l'une des autorités de CA requises dans leur liste de CA approuvées par défaut.
- Java 1.4.2_12 (mai 2006), 5 Update 2 (mars 2005) et toutes les versions ultérieures, y compris Java 6 (décembre 2006), 7 et 8, contiennent au moins l'une des CA requises dans leur liste par défaut de CA approuvées.

Lorsque vous accédez à la console de gestion AWS Lambda ou aux points de terminaison de l'API AWS Lambda via des navigateurs ou par programmation, vous devez vous assurer que vos machines clientes prennent en charge les autorités de certification suivantes :

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

Les certificats racine des deux premières autorités sont disponibles auprès des services [Amazon Trust Services](#), mais il est encore plus simple de maintenir votre ordinateur à jour. Pour en savoir plus sur les certificats fournis par ACM, consultez les [FAQ sur AWS Certificate Manager](#).

Versions d'AWS Lambda

Le tableau suivant décrit les modifications importantes apportées au Manuel du développeur d'AWS Lambda après mai 2018. Pour recevoir les notifications des mises à jour de cette documentation, abonnez-vous au [flux RSS](#).

update-history-change	update-history-description	update-history-date
Runtimes personnalisés (p. 551)	Créez un runtime personnalisé pour exécuter des fonctions Lambda dans votre langage de programmation favori. Consultez Runtimes AWS Lambda personnalisés pour plus d'informations.	November 29, 2018
Ruby (p. 551)	AWS Lambda prend désormais en charge Ruby 2.5 avec un nouveau runtime. Consultez Création de fonctions Lambda avec Ruby pour plus d'informations.	November 29, 2018
Déclencheurs des équilibreurs de charge d'applications (p. 551)	Elastic Load Balancing prend désormais en charge les fonctions Lambda en tant que cibles pour les Application Load Balancers. Consultez Utilisation de Lambda avec des équilibreurs de charge d'application pour plus d'informations.	November 29, 2018
Couches (p. 551)	Avec les couches Lambda, vous pouvez regrouper et déployer des bibliothèques, des runtimes personnalisés, ainsi que d'autres dépendances séparément depuis le code de votre fonction. Partagez vos couches avec vos autres comptes ou avec le monde entier. Consultez Couches AWS Lambda pour plus d'informations.	November 29, 2018
Utilisation des consommateurs de flux Kinesis HTTP/2 comme déclencheur (p. 551)	Vous pouvez utiliser les consommateurs de flux de données Kinesis HTTP/2 pour envoyer des événements à AWS Lambda. Les consommateurs de flux dédiés ont un débit de lecture dédié à partir de chaque partition dans votre flux de données et utilisent HTTP/2 pour réduire la latence. Consultez Utilisation d'AWS Lambda avec Kinesis pour plus de détails.	November 19, 2018

Python 3.7 (p. 551)	AWS Lambda prend désormais en charge Python 3.7 avec un nouveau runtime. Pour plus d'informations, consultez Création de fonctions Lambda en Python .	November 19, 2018
Augmentation de la limite de la charge utile d'appel de fonction asynchrone (p. 551)	La taille maximale de la charge utile pour les appels asynchrones a augmenté de 128 Ko à 256 Ko, et correspond à la taille de message maximale à partir d'un déclencheur Amazon SNS. Consultez Limites AWS Lambda pour plus de détails.	November 16, 2018
Région AWS GovCloud (USA Est) (p. 551)	AWS Lambda est désormais disponible dans la région AWS GovCloud (USA Est). Pour en savoir plus, consultez AWS GovCloud (US-East) Now Open sur le blog AWS.	November 12, 2018
Les rubriques relatives à AWS SAM ont été placées dans un manuel du développeur distinct (p. 551)	Un certain nombre de rubriques étaient axées sur la création d'applications sans serveur à l'aide du modèle Modèle d'application sans serveur AWS (AWS SAM). Ces rubriques ont été déplacées vers le Manuel du développeur Modèle d'application sans serveur AWS .	October 25, 2018
Affichage des applications Lambda dans la console (p. 551)	Vous pouvez afficher le statut de vos applications Lambda à la page Applications de la console Lambda. Cette page affiche le statut de la pile AWS CloudFormation. Elle inclut des liens vers les pages où vous pouvez consulter plus d'informations sur les ressources figurant dans la pile. Vous pouvez également consulter les métriques agrégées pour l'application et créer des tableaux de bord de surveillance personnalisés.	October 11, 2018
Délai d'attente d'exécution de la fonction (p. 551)	Pour permettre d'utiliser des fonctions de longue durée, le délai d'exécution maximal configurable est passé de 5 minutes à 15 minutes. Consultez Limites AWS Lambda pour plus de détails.	October 10, 2018

Prise en charge du langage PowerShell Core dans AWS Lambda (p. 551)	AWS Lambda prend désormais en charge le langage PowerShell Core. Pour plus d'informations, consultez Modèle de programmation pour la création de fonctions Lambda dans PowerShell .	September 11, 2018
Prise en charge de l'exécution .NET Core 2.1.0 dans AWS Lambda (p. 551)	AWS Lambda prend maintenant en charge l'exécution de .NET Core 2.1.0. Pour plus d'informations, consultez Interface de ligne de commande .NET Core .	July 9, 2018
Mises à jour disponibles sur RSS (p. 551)	Vous pouvez à présent vous abonner à un flux RSS pour recevoir les notifications du Manuel du développeur AWS Lambda.	July 5, 2018
Région Chine (Ningxia) (p. 551)	AWS Lambda est désormais disponible dans la région Région Chine (Ningxia). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	June 28, 2018
Prise en charge pour Amazon SQS en tant que source d'événement (p. 551)	AWS Lambda prend désormais en charge Amazon Simple Queue Service (Amazon SQS) en tant que source d'événement. Pour plus d'informations, consultez Appel de fonctions Lambda .	June 28, 2018

Mises à jour antérieures

Le tableau ci-après décrit les modifications importantes apportées dans chaque version du Manuel du développeur AWS Lambda avant juin 2018.

Modification	Description	Date
Prise en charge de l'environnement d'exécution Node.js 8.10	AWS Lambda prend désormais en charge le runtime Node.js version 8.10. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js (p. 289) .	2 avril 2018
ID de révision des fonctions et alias	AWS Lambda prend désormais en charge les ID de révision sur les versions et alias de vos fonctions. Vous pouvez utiliser ces identifiants pour suivre et appliquer des mises à jour conditionnelles lorsque vous mettez à jour votre version de fonction ou vos ressources d'alias.	25 janvier 2018

Modification	Description	Date
Prise en charge du runtime pour Go et .NET 2.0	La prise en charge du runtime pour Go et .NET 2.0 a été ajoutée pour AWS Lambda. Pour plus d'informations, consultez Création de fonctions Lambda avec Go (p. 344) et Création de fonctions Lambda avec C# (p. 357) .	15 janvier 2018
Nouvelle conception de la console	AWS Lambda propose une nouvelle console Lambda afin de simplifier votre expérience et a ajouté un éditeur de code Cloud9 pour que vous puissiez plus facilement déboguer et modifier le code de fonction. Pour plus d'informations, consultez Création de fonctions à l'aide de l'éditeur de la console AWS Lambda (p. 17) .	novembre 30, 2017
Définition de limites de simultanéité pour des fonctions individuelles	AWS Lambda prend désormais en charge la définition de limites de simultanéité pour des fonctions individuelles. Pour plus d'informations, consultez Gestion de la simultanéité (p. 279) .	novembre 30, 2017
Déplacement du trafic avec des alias	AWS Lambda prend désormais en charge le déplacement du trafic avec des alias. Pour plus d'informations, consultez Déplacement du trafic à l'aide des alias (p. 54) .	28 novembre 2017
Déploiement de code graduel	AWS Lambda prend désormais en charge le déploiement en toute sécurité de nouvelles versions de la fonction Lambda en tirant parti du déploiement de code. Pour plus d'informations, consultez Déploiement de code graduel .	28 novembre 2017
Région Chine (Pékin)	AWS Lambda est désormais disponible dans la région Région Chine (Pékin). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	9 novembre 2017
Présentation de SAM Local	AWS Lambda propose SAM Local (désormais appelé interface de ligne de commande SAM), un outil de l'AWS CLI qui vous fournit un environnement pour développer, tester et analyser localement vos applications sans serveur avant de les charger dans le runtime Lambda. Pour plus d'informations, consultez Test et débogage d'applications sans serveur .	11 août 2017
Région Canada (Centre)	AWS Lambda est désormais disponible dans la région Région Canada (Centre). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	22 juin 2017
Région Amérique du Sud (São Paulo)	AWS Lambda est désormais disponible dans la région Région Amérique du Sud (São Paulo). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	6 juin 2017
Prise en charge AWS Lambda d'AWS X-Ray.	Lambda prend désormais en charge X-Ray, ce qui vous permet de détecter, d'analyser et d'optimiser les problèmes de performance avec les applications Lambda. Pour plus d'informations, consultez Utilisation d'AWS X-Ray (p. 231) .	19 avril 2017
Région Asie-Pacifique (Mumbai)	AWS Lambda est désormais disponible dans la région Région Asie-Pacifique (Mumbai). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	28 mars 2017

Modification	Description	Date
AWS Lambda prend désormais en charge Node.js Runtime v6.10	AWS Lambda prend désormais en charge le runtime Node.js v6.10. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js (p. 289) .	22 mars 2017
Région UE (Londres)	AWS Lambda est désormais disponible dans la région Région UE (Londres). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	1 février 2017
Prise en charge d'AWS Lambda pour l'environnement d'exécution .NET, Lambda@Edge (Aperçu), les files d'attente de lettres mortes et le déploiement automatisé des applications sans serveur.	<p>AWS Lambda propose les fonctionnalités suivantes :</p> <ul style="list-style-type: none"> • AWS Lambda a ajouté la prise en charge de C#. Pour plus d'informations, consultez Création de fonctions Lambda avec C# (p. 357). • Lambda@Edge (Aperçu) vous permet d'exécuter des fonctions Lambda au niveau des emplacements périphériques AWS en réponse à des événements CloudFront. Pour plus d'informations, consultez Lambda@Edge (p. 221). • Un didacticiel a été ajouté pour l'automatisation du déploiement des applications sans serveur à l'aide d'AWS CodePipeline, d'AWS CodeBuild et d'AWS CloudFormation. Pour plus d'informations, consultez Déploiement d'applications sans serveur. • La rubrique Utiliser Amazon CloudWatch (p. 223) a été mise à jour pour inclure une section sur les Files d'attente Lettre Morte (p. 81), que vous pouvez configurer afin de récupérer des informations sur les appels asynchrones des fonctions Lambda ayant échoué. 	3 décembre 2016
AWS Lambda ajoute Amazon Lex comme source d'événement prise en charge.	Avec Lambda et Amazon Lex, vous pouvez créer rapidement des chatbots pour divers services, comme Slack et Facebook. Pour plus d'informations, consultez Amazon Lex (p. 89) .	30 novembre 2016
Région USA Ouest (Californie du Nord)	AWS Lambda est désormais disponible dans la région Région USA Ouest (Californie du Nord). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	21 novembre 2016

Modification	Description	Date
Introduction du modèle d'application sans serveur AWS pour la création et le déploiement d'applications basées sur Lambda et pour l'utilisation de variables d'environnement pour les paramètres de configuration de la fonction Lambda.	<p>AWS Lambda propose les nouvelles fonctions suivantes dans cette version.</p> <ul style="list-style-type: none"> Modèle d'application sans serveur AWS : vous pouvez désormais utiliser AWS SAM pour définir la syntaxe d'expression des ressources au sein d'une application sans serveur. Pour déployer votre application, il vous suffit de spécifier les ressources dont vous avez besoin dans le cadre de cette dernière, ainsi que les stratégies d'autorisations qui leur sont associées, dans un fichier de modèle AWS CloudFormation (au format JSON ou YAML), d'emballer vos artefacts de déploiement et de déployer le modèle. Pour plus d'informations, consultez Applications AWS Lambda (p. 117). Variables d'environnement : vous pouvez utiliser des variables d'environnement pour spécifier des paramètres de configuration pour votre fonction Lambda en dehors du code de votre fonction. Pour plus d'informations, consultez Variables d'environnement AWS Lambda (p. 30). 	18 novembre 2016
Ajout d'un didacticiel sous Mise en route (p. 3) pour la création d'un point de terminaison Amazon API Gateway à l'aide de la console Lambda	Ce didacticiel explique comment intégrer en toute transparence une fonction Lambda à une API via de nouvelles fonctions, présentées dans Configuration de l'intégration de proxy avec une ressource de proxy . Pour plus d'informations, consultez Création d'un microservice simple avec Lambda et API Gateway (p. 198) .	29 août 2016
Région Asie-Pacifique (Séoul)	AWS Lambda est désormais disponible dans la région Région Asie-Pacifique (Séoul). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	29 août 2016
Région Asie-Pacifique (Sydney)	Lambda est désormais disponible dans la région Région Asie-Pacifique (Sydney). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	23 juin 2016
Mises à jour apportées à la console Lambda	La console Lambda a été mise à jour pour simplifier le processus de création de rôles. Pour plus d'informations, consultez Créer une fonction Lambda avec la console (p. 6) .	23 juin 2016
AWS Lambda prend désormais en charge Node.js Runtime v4.3	AWS Lambda prend désormais en charge le runtime Node.js v4.3. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js (p. 289) .	07 avril 2016
UE (Francfort) région	Lambda est désormais disponible dans la région UE (Francfort). Pour plus d'information sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans la documentation AWS General Reference.	14 mars 2016
Prise en charge de VPC	Vous pouvez maintenant configurer une fonction Lambda pour accéder aux ressources de votre VPC. Pour plus d'informations, consultez Configuration d'une fonction Lambda pour accéder aux ressources d'un Amazon VPC (p. 61) .	11 février 2016

Modification	Description	Date
Réorganisation du contenu	<p>Le contenu a été réorganisé et fournit maintenant les éléments suivants :</p> <ul style="list-style-type: none"> • Mise en route (p. 3) – Contient un exercice basé sur la console, dans lequel vous créez une fonction Lambda « Hello World ». Vous explorez les fonctionnalités de la console AWS Lambda, y compris les plans qui vous permettent de créer des fonctions Lambda en seulement quelques clics. • Cas d'utilisation (p. 126) – Fournit des exemples d'utilisation d'AWS Lambda avec d'autres services AWS ou avec vos applications personnalisées en tant que sources d'événements, et décrit comment appeler des fonctions via HTTPS et comment configurer AWS Lambda pour appeler la fonction Lambda à intervalles réguliers. • Modèle de programmation (p. 24) – Explique des concepts clés du modèle de programmation et décrit les détails spécifiques au langage. Quel que soit le langage que vous choisissez, un modèle commun régit la création du code pour une fonction Lambda. • Création d'un package de déploiement (p. 25) – Explique comment créer des packages de déploiement pour le code de la fonction Lambda créé dans les langages pris en charge par AWS Lambda (Python, Java et Node.js). 	9 décembre 2015
AWS Lambda Runtime a été mis à jour.	<p>AWS Lambda Runtime a été mis à jour avec les versions de noyau Linux et de kit SDK suivantes :</p> <ul style="list-style-type: none"> • Kit SDK AWS pour JavaScript : 2.2.12 • Kit SDK Boto : 1.2.1 • Version du noyau Linux : 4.9.62-21.56.amzn1.x86_64 <p>Pour plus d'informations, consultez Environnement d'exécution Lambda et bibliothèques disponibles (p. 104).</p>	4 novembre 2015

Modification	Description	Date
Prise en charge de la gestion des versions prennent en charge, Python pour le développement de code pour des fonctions Lambda, événements planifiés et rallongement du temps d'exécution	<p>AWS Lambda présente les fonctions suivantes dans cette version.</p> <ul style="list-style-type: none"> Python : vous pouvez désormais développer le code de la fonction Lambda à l'aide du langage Python. Pour de plus amples informations, veuillez consulter Modèle de programmation (p. 24). Gestion des versions : vous pouvez conserver une ou plusieurs versions de la fonction Lambda. La gestion des versions vous permet de contrôler quelle version de fonction Lambda est exécutée dans des environnements différents (par exemple, développement, test ou production). Pour plus d'informations, consultez Versions et alias des fonctions AWS Lambda (p. 38). Événements planifiés : vous pouvez également configurer AWS Lambda pour appeler régulièrement le code à l'aide de la console AWS Lambda. Vous pouvez spécifier un taux fixe (nombre d'heures, de jours ou de semaines) ou vous pouvez spécifier une expression cron. Pour obtenir un exemple, consultez Utilisation de AWS Lambda avec Amazon CloudWatch Events (p. 209). Rallongement du temps d'exécution : vous pouvez maintenant définir un temps d'exécution pouvant aller jusqu'à cinq minutes pour les fonctions Lambda afin de rendre possibles les fonctions de plus longue durée, telles que l'intégration de données volumineuses et le traitement des tâches. 	08 octobre 2015
Deux nouvelles procédures	<p>Les procédures suivantes ont été ajoutées. Elles utilisent toutes les deux la fonction Lambda Java.</p> <p>Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB (p. 164)</p> <p>Utilisation d'AWS Lambda en tant que backend pour applications mobiles Android (p. 200)</p>	27 août 2015
Prise en charge de DynamoDB Streams	DynamoDB Streams est désormais disponible dans toutes les régions qui acceptent DynamoDB. Vous pouvez activer DynamoDB Streams pour votre table et utiliser une fonction Lambda comme déclencheur de cette table. Les déclencheurs sont des actions personnalisées que vous effectuez en réponse aux mises à jour apportées à la table DynamoDB. Pour afficher un exemple, consultez Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB (p. 164) .	14 juillet 2015

Modification	Description	Date
AWS Lambda prend désormais en charge l'appel des fonctions Lambda avec les clients compatibles REST.	<p>Jusqu'à présent, pour appeler votre fonction Lambda à partir de votre application web, mobile ou IoT, les kits SDK AWS (par exemple, kit AWS SDK pour Java, kit AWS SDK pour Android ou kit AWS SDK pour iOS) étaient nécessaires. Désormais, AWS Lambda prend en charge l'appel d'une fonction Lambda avec les clients compatibles REST via une API personnalisée que vous pouvez créer à l'aide d'Amazon API Gateway. Vous pouvez envoyer les requêtes à l'URL du point de terminaison de la fonction Lambda. Vous pouvez configurer la sécurité au niveau du point de terminaison pour autoriser l'accès ouvert, pour tirer parti d'AWS Identity and Access Management (IAM) afin d'autoriser l'accès ou pour utiliser des clés d'API afin de contrôler l'accès à vos fonctions Lambda par des tiers.</p> <p>Pour obtenir un exemple d'exercice de mise en route, consultez Utilisation de AWS Lambda avec Amazon API Gateway (p. 186).</p> <p>Pour plus d'informations sur Amazon API Gateway, consultez https://aws.amazon.com/api-gateway/.</p>	09 juillet 2015
La console AWS Lambda fournit maintenant des plans pour créer facilement des fonctions Lambda et les tester.	La console AWS Lambda fournit un ensemble de plans. Chaque plan fournit un exemple de configuration de source d'événement et un exemple de code que vous pouvez utiliser pour créer facilement des applications basées sur Lambda. Tous les exercices de mise en route AWS Lambda utilisent désormais les plans. Pour plus d'informations, consultez Mise en route (p. 3) .	Dans cette version
AWS Lambda prend désormais en charge Java pour créer vos fonctions Lambda.	Vous pouvez désormais créer le code Lambda en Java. Pour de plus amples informations, veuillez consulter Modèle de programmation (p. 24) .	15 juin 2015
AWS Lambda prend désormais en charge la spécification d'un objet Amazon S3 en tant que fonction .zip lorsque vous créez ou mettez à jour une fonction Lambda.	Vous pouvez importer un package de déploiement de fonctions Lambda (fichier .zip) dans un compartiment Amazon S3 de la région dans laquelle vous souhaitez créer une fonction Lambda. Vous pouvez ensuite spécifier le nom du compartiment et le nom de la clé objet lorsque vous créez ou mettez à jour une fonction Lambda.	28 mai 2015

Modification	Description	Date
AWS Lambda est désormais disponible de manière globale et prend en charge les backends mobiles	<p>AWS Lambda est maintenant disponible de manière globale en production. Cette version présente également de nouvelles fonctionnalités qui facilitent la création de backends pour les mobiles, les tablettes et l'Internet des objets (IoT) via AWS Lambda, lesquels s'adaptent automatiquement sans avoir à mettre en service ou à gérer l'infrastructure. AWS Lambda prend désormais en charge les événements en temps réel (synchrone) et les événements asynchrones. Les fonctionnalités supplémentaires comprennent la rationalisation de la configuration et de la gestion des sources d'événement. Le modèle d'autorisation et le modèle de programmation ont été simplifiés par la mise en place de stratégies de ressources pour les fonctions Lambda.</p> <p>La documentation a été mise à jour en conséquence. Pour plus d'informations, consultez les rubriques suivantes :</p> <p>Mise en route (p. 3)</p> <p>AWS Lambda</p>	9 avril 2015
Version préliminaire	Version préliminaire du Manuel du développeur AWS Lambda.	13 novembre 2014

Glossaire AWS

Pour la terminologie AWS la plus récente, consultez le [Glossaire AWS](#) dans le document AWS General Reference.