
AWS Lambda

Manuel du développeur



AWS Lambda: Manuel du développeur

Copyright © 2020 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

| | |
|---|----|
| Présentation d'AWS Lambda | 1 |
| Dans quels cas est-il conseillé d'utiliser AWS Lambda ? | 1 |
| Vous utilisez AWS Lambda pour la première fois ? | 2 |
| Mise en route | 3 |
| Création d'une fonction | 3 |
| Utiliser le Designer (Concepteur) | 4 |
| Appeler la fonction Lambda | 4 |
| Nettoyage | 6 |
| Éditeur de code | 6 |
| Utilisation de fichiers et de dossiers | 7 |
| Utilisation du code | 8 |
| Utilisation du mode plein écran | 11 |
| Utilisation des préférences | 11 |
| AWS CLI | 12 |
| Prérequis | 12 |
| Créer le rôle d'exécution | 12 |
| Créer la fonction | 14 |
| Répertorier les fonctions Lambda dans votre compte | 16 |
| Récupérer une fonction Lambda | 16 |
| Nettoyage | 17 |
| Concepts | 17 |
| Fonction | 17 |
| Qualificateur | 18 |
| Runtime | 18 |
| Événement | 18 |
| Concurrency | 18 |
| Déclencheur | 19 |
| Fonctions | 19 |
| Modèle de programmation | 19 |
| Package de déploiement | 21 |
| Couches | 21 |
| Dimensionnement | 21 |
| Contrôles de simultanéité | 22 |
| Appel asynchrone | 24 |
| Mappages de source d'événement | 25 |
| Destinations | 26 |
| Plans de fonction | 27 |
| Modèles d'application | 28 |
| Outils | 28 |
| AWS Command Line Interface | 28 |
| Modèle d'application sans serveur AWS | 28 |
| INTERFACE DE LIGNE DE COMMANDE SAM | 29 |
| Outils de création de code | 29 |
| Limites | 30 |
| Autorisations | 32 |
| Rôle d'exécution | 33 |
| Création d'un rôle d'exécution dans la console IAM | 34 |
| Gestion des rôles avec l'API IAM | 34 |
| Stratégies gérées pour les fonctionnalités Lambda | 35 |
| Stratégies basées sur les ressources | 36 |
| Attribution de l'accès à la fonction aux services AWS | 38 |
| Octroi de l'accès à la fonction à d'autres comptes | 38 |
| Octroi de l'accès de la couche à d'autres comptes | 40 |
| Nettoyage des stratégies basées sur les ressources | 40 |

| | |
|---|-----|
| Stratégies utilisateur | 41 |
| Développement des fonctions | 41 |
| Développement et utilisation des couches | 44 |
| Rôles entre comptes | 45 |
| Ressources et conditions | 45 |
| Fonctions | 47 |
| Mappages de source d'événement | 48 |
| Couches | 49 |
| Limites d'autorisations | 49 |
| Gestion des fonctions | 52 |
| Console de configuration | 53 |
| Variables d'environnement | 55 |
| Variables d'environnement d'exécution | 57 |
| Sécurisation des variables d'environnement | 58 |
| Configuration de variables d'environnement avec l'API Lambda | 60 |
| Exemples de code et de modèles | 60 |
| Concurrency | 61 |
| Configuration de la simultanéité réservée | 62 |
| Configuration de la simultanéité provisionnée | 64 |
| Configuration de la simultanéité avec l'API Lambda | 67 |
| Versions | 70 |
| Gestion des versions avec l'API Lambda | 71 |
| Utilisation des versions | 71 |
| Stratégies basées sur une ressource | 72 |
| Alias | 72 |
| Gestion des alias avec l'API Lambda | 73 |
| Utilisation des alias | 73 |
| Stratégies basées sur une ressource | 74 |
| Configuration du routage d'alias | 74 |
| Couches | 76 |
| Configuration d'une fonction pour utiliser les couches | 76 |
| Gestion des couches | 77 |
| Inclusion de dépendances de bibliothèques dans une couche | 79 |
| Autorisations d'utilisation des couches | 80 |
| AWS CloudFormation and AWS SAM | 80 |
| Exemples d'applications | 81 |
| Réseau | 81 |
| Rôle d'exécution et autorisations utilisateur | 82 |
| Configuration de l'accès au Amazon VPC avec l'API Lambda | 83 |
| Accès à Internet et aux services pour des fonctions connectées à un VPC | 83 |
| Exemples de configurations de VPC | 84 |
| Base de données | 84 |
| Utilisation des autorisations de la fonction pour l'authentification | 85 |
| Exemple d'application | 86 |
| Balises | 87 |
| Utilisation des balises avec AWS CLI | 89 |
| Exigences relatives à la clé et à la valeur des balises | 90 |
| Appel de fonctions | 91 |
| Appel synchrone | 92 |
| Appel asynchrone | 93 |
| Configuration de la gestion des erreurs pour les appels asynchrones | 96 |
| Configuration des destinations pour les appels asynchrones | 96 |
| API de configuration d'appel asynchrone | 98 |
| Files d'attente de lettres mortes | 99 |
| Mappage de source d'événement | 101 |
| États de fonction | 105 |
| Dimensionnement d'une fonction | 106 |

| | |
|--|-----|
| Gestion des erreurs | 111 |
| Mobile SDK pour Android | 113 |
| Didacticiel | 114 |
| Exemple de code | 119 |
| Environnements d'exécution Lambda | 122 |
| Contexte d'exécution | 124 |
| Stratégie de prise en charge des environnements d'exécution | 124 |
| Environnements d'exécution personnalisés | 126 |
| Utilisation d'un runtime personnalisé | 126 |
| Création d'un runtime personnalisé | 126 |
| Interface de runtime | 128 |
| Appel suivant | 128 |
| Réponse d'appel | 129 |
| Erreur d'appel | 129 |
| Erreur d'initialisation | 130 |
| Didacticiel – Runtime personnalisé | 130 |
| Prérequis | 130 |
| Création d'une fonction | 131 |
| Créer une couche | 133 |
| Mettre à jour la fonction | 133 |
| Mettre à jour le runtime | 134 |
| Partager la couche | 135 |
| Nettoyage | 135 |
| Applications Lambda | 136 |
| Gérer des applications | 136 |
| Surveillance des applications | 137 |
| Tableaux de bord de surveillance personnalisés | 137 |
| Didacticiel – Créez une application | 139 |
| Prérequis | 140 |
| Création d'une application | 140 |
| Appel de la fonction | 141 |
| Ajouter une ressource AWS | 142 |
| Mettre à jour la limite des autorisations | 144 |
| Mettre à jour le code de la fonction | 144 |
| Étapes suivantes | 146 |
| Dépannage | 146 |
| Nettoyage | 147 |
| Déploiements propagés | 148 |
| Exemple de modèle AWS SAM Lambda | 148 |
| Cas d'utilisation | 149 |
| Exemple 1 : Amazon S3 transmet les événements et appelle une fonction Lambda | 150 |
| Exemple 2 : AWS Lambda extrait les événements d'un flux Kinesis et appelle une fonction Lambda | 150 |
| Bonnes pratiques | 151 |
| Code de fonction | 151 |
| Configuration de fonctions | 152 |
| Alarmes et métriques | 153 |
| Appels d'événements de flux | 153 |
| Utilisation avec d'autres services | 155 |
| Alexa | 157 |
| API Gateway | 157 |
| Autorisations | 160 |
| Gestion des erreurs avec une API API Gateway | 161 |
| Choix d'un type d'API | 162 |
| Exemples d'applications | 164 |
| Didacticiel | 164 |
| Exemple de code | 173 |

| | |
|---|-----|
| Plan de microservice | 175 |
| Exemple de modèle | 177 |
| CloudTrail | 177 |
| Journaux CloudTrail | 179 |
| Didacticiel | 182 |
| Exemple de code | 186 |
| CloudWatch Events | 188 |
| Didacticiel | 189 |
| Exemple de modèle | 192 |
| Expressions de planification | 193 |
| CloudWatch Logs | 194 |
| CloudFormation | 195 |
| CloudFront (Lambda@Edge) | 197 |
| CodeCommit | 198 |
| CodePipeline | 199 |
| Autorisations | 200 |
| Didacticiel | 201 |
| Cognito | 206 |
| Config | 207 |
| DynamoDB | 208 |
| Autorisations du rôle d'exécution | 210 |
| Configuration d'un flux comme source d'événement | 210 |
| API de mappage de la source d'événement | 211 |
| Gestion des erreurs | 213 |
| Métriques Amazon CloudWatch | 214 |
| Didacticiel | 214 |
| Exemple de code | 219 |
| Exemple de modèle | 222 |
| EC2 | 222 |
| Autorisations | 223 |
| Didacticiel : Instances Spot | 224 |
| ElastiCache | 232 |
| Prérequis | 232 |
| Créer le rôle d'exécution | 233 |
| Créer un cluster ElastiCache | 233 |
| Créer un package de déploiement | 233 |
| Créer la fonction Lambda | 234 |
| Tester la fonction Lambda | 234 |
| Elastic Load Balancing | 235 |
| IoT | 236 |
| IoT Events | 237 |
| Kinesis Firehose | 239 |
| Flux Kinesis | 239 |
| Configuration de votre fonction et de votre flux de données | 241 |
| Autorisations du rôle d'exécution | 242 |
| Configuration d'un flux comme source d'événement | 242 |
| API de mappage de la source d'événement | 243 |
| Gestion des erreurs | 245 |
| Métriques Amazon CloudWatch | 246 |
| Didacticiel | 246 |
| Exemple de code | 250 |
| Exemple de modèle | 253 |
| Lex | 255 |
| Rôles et autorisations | 257 |
| RDS | 257 |
| Didacticiel : Amazon RDS | 258 |
| S3 | 262 |

| | |
|--|-----|
| Didacticiel | 263 |
| Exemple de code | 270 |
| Exemple de modèle | 275 |
| Lot S3 | 276 |
| Appel de fonctions Lambda à partir d'opérations par lot Amazon S3 | 277 |
| SES | 278 |
| SNS | 280 |
| Didacticiel | 281 |
| Exemple de code | 283 |
| SQS | 285 |
| Dimensionnement et traitement | 287 |
| Configuration d'une file d'attente à utiliser avec Lambda | 287 |
| Autorisations du rôle d'exécution | 288 |
| Configuration d'une file d'attente en tant que source d'événement | 288 |
| API de mappage de la source d'événement | 211 |
| Didacticiel | 289 |
| Exemple de code | 292 |
| Exemple de modèle | 295 |
| X-Ray | 296 |
| Autorisations du rôle d'exécution | 298 |
| Démon AWS X-Ray | 298 |
| Activation du suivi actif avec l'API Lambda | 298 |
| Activation du suivi actif avec AWS CloudFormation | 299 |
| Exemples | 300 |
| Fonction vide | 301 |
| Architecture et code de gestionnaire | 302 |
| Automatisation du déploiement avec AWS CloudFormation et l'AWS CLI | 303 |
| Instrumentation avec le AWS X-Ray | 306 |
| Gestion des dépendances avec des couches | 307 |
| Processeur d'erreurs | 308 |
| Structure d'événement et architecture | 309 |
| Instrumentation avec AWS X-Ray | 310 |
| Modèle AWS CloudFormation et ressources supplémentaires | 311 |
| Utilisation de Node.js | 312 |
| Gestionnaire | 314 |
| Gestionnaires asynchrones | 315 |
| Gestionnaires non asynchrones | 315 |
| Package de déploiement | 316 |
| Mise à jour d'une fonction sans dépendances | 316 |
| Mise à jour d'une fonction avec dépendances supplémentaires | 317 |
| Contexte | 318 |
| Journalisation | 319 |
| Affichage des journaux dans la AWS Management Console | 320 |
| Utilisation de l'AWS CLI | 321 |
| Suppression de journaux | 322 |
| Erreurs | 322 |
| Suivi | 324 |
| Activation du suivi actif avec l'API Lambda | 326 |
| Activation du suivi actif avec AWS CloudFormation | 327 |
| Stockage des dépendances d'exécution dans une couche | 327 |
| Travail avec Python | 329 |
| Gestionnaire | 331 |
| Package de déploiement | 332 |
| Prérequis | 332 |
| Mise à jour d'une fonction sans dépendances | 333 |
| Mise à jour d'une fonction avec dépendances supplémentaires | 333 |
| Avec un environnement virtuel | 334 |

| | |
|---|-----|
| Contexte | 336 |
| Journalisation | 337 |
| Affichage des journaux dans la AWS Management Console | 338 |
| Utilisation de l'AWS CLI | 338 |
| Suppression de journaux | 340 |
| Bibliothèque de journalisation | 340 |
| Erreurs | 341 |
| Suivi | 342 |
| Activation du suivi actif avec l'API Lambda | 344 |
| Activation du suivi actif avec AWS CloudFormation | 345 |
| Stockage des dépendances d'exécution dans une couche | 345 |
| Utilisation de Ruby | 347 |
| Gestionnaire | 349 |
| Package de déploiement | 350 |
| Mise à jour d'une fonction sans dépendances | 350 |
| Mise à jour d'une fonction avec dépendances supplémentaires | 351 |
| Contexte | 351 |
| Journalisation | 352 |
| Affichage des journaux dans la AWS Management Console | 338 |
| Utilisation de l'AWS CLI | 338 |
| Suppression de journaux | 340 |
| Erreurs | 356 |
| Tracing | 358 |
| Activation du suivi actif avec l'API Lambda | 360 |
| Activation du suivi actif avec AWS CloudFormation | 361 |
| Stockage des dépendances d'exécution dans une couche | 361 |
| Travail avec Java | 363 |
| Exemples d'application | 364 |
| Package de déploiement | 366 |
| Création d'un package de déploiement avec Gradle | 368 |
| Création d'un package de déploiement avec Maven | 368 |
| Chargement d'un package de déploiement avec l'API Lambda | 370 |
| Chargement d'un package de déploiement avec AWS SAM | 371 |
| Gestionnaire | 372 |
| Choix des types d'entrée et de sortie | 373 |
| Interfaces du gestionnaire | 374 |
| Exemple de code de gestionnaire | 375 |
| Contexte | 376 |
| Contexte dans des exemples d'applications | 378 |
| Journalisation | 378 |
| Affichage des journaux dans la AWS Management Console | 380 |
| Utilisation de l'AWS CLI | 380 |
| Suppression de journaux | 382 |
| Journalisation avancée avec Log4j 2 et SLF4J | 382 |
| Exemple de code de journalisation | 384 |
| Erreurs | 384 |
| Affichage de la sortie d'erreur | 386 |
| Présentation des types et sources d'erreurs | 387 |
| Gestion des erreurs dans les clients | 388 |
| Gestion des erreurs dans d'autres services AWS | 389 |
| Gestion des erreurs dans les exemples d'applications | 389 |
| Tracing | 390 |
| Activation du suivi actif avec l'API Lambda | 392 |
| Activation du suivi actif avec AWS CloudFormation | 393 |
| Stockage des dépendances d'exécution dans une couche | 393 |
| Suivi dans des exemples d'applications | 394 |
| Didacticiel - IDE Eclipse | 395 |

| | |
|---|-----|
| Prérequis | 395 |
| Créer et développer un projet | 395 |
| Utilisation de Go | 398 |
| Package de déploiement | 398 |
| Création d'un package de déploiement sous Windows | 399 |
| Gestionnaire | 399 |
| Gestionnaire de fonctions Lambda à l'aide de types structurés | 401 |
| Utilisation de l'état global | 402 |
| Contexte | 403 |
| Accès aux informations du contexte d'appel | 403 |
| Journalisation | 404 |
| Affichage des journaux dans la AWS Management Console | 406 |
| Utilisation de l'AWS CLI | 406 |
| Suppression de journaux | 408 |
| Erreurs | 408 |
| Tracing | 408 |
| Activation du suivi actif avec l'API Lambda | 411 |
| Activation du suivi actif avec AWS CloudFormation | 411 |
| Variables d'environnement | 412 |
| Utilisation de C# | 413 |
| Package de déploiement | 414 |
| Interface de ligne de commande .NET Core | 414 |
| AWS Toolkit for Visual Studio | 417 |
| Gestionnaire | 419 |
| Gestion des flux | 420 |
| Gestion des types de données standard | 420 |
| Signatures de gestionnaire | 421 |
| Sérialisation des fonctions Lambda | 422 |
| Restrictions liées au gestionnaire de fonctions Lambda | 422 |
| Utilisation des fonctions asynchrones en C# avec AWS Lambda | 423 |
| Contexte | 424 |
| Journalisation | 424 |
| Affichage des journaux dans la AWS Management Console | 426 |
| Utilisation de l'AWS CLI | 426 |
| Suppression de journaux | 428 |
| Erreurs | 428 |
| Tracing | 430 |
| Activation du suivi actif avec l'API Lambda | 433 |
| Activation du suivi actif avec AWS CloudFormation | 434 |
| Utilisation de PowerShell | 435 |
| Environnement de développement | 435 |
| Package de déploiement | 436 |
| Gestionnaire | 438 |
| Renvoi de données | 438 |
| Contexte | 439 |
| Journalisation | 439 |
| Affichage des journaux dans la AWS Management Console | 441 |
| Utilisation de l'AWS CLI | 441 |
| Suppression de journaux | 443 |
| Erreurs | 443 |
| Surveillance | 444 |
| Console de surveillance | 444 |
| Métriques de fonction | 445 |
| Utilisation des métriques d'appel | 446 |
| Utilisation des métriques de performances | 447 |
| Utilisation des métriques de simultanéité | 447 |
| Journaux CloudWatch | 447 |

| | |
|--|-----|
| Sécurité | 449 |
| Protection des données | 449 |
| Chiffrement en transit | 450 |
| Chiffrement au repos | 450 |
| Identity and Access Management | 451 |
| Public ciblé | 451 |
| Authentification avec des identités | 451 |
| Gestion de l'accès à l'aide de stratégies | 453 |
| Fonctionnement de AWS Lambda avec IAM | 455 |
| Exemples de stratégies basées sur l'identité | 455 |
| Dépannage | 457 |
| Validation de la conformité | 459 |
| Résilience | 460 |
| Sécurité de l'infrastructure | 460 |
| Configuration et analyse des vulnérabilités | 461 |
| Dépannage | 462 |
| Déploiement | 462 |
| Appel | 464 |
| Exécution | 466 |
| Mise en réseau | 467 |
| Versions | 469 |
| Mises à jour antérieures | 479 |
| Référence d'API | 485 |
| Actions | 485 |
| AddLayerVersionPermission | 487 |
| AddPermission | 490 |
| CreateAlias | 494 |
| CreateEventSourceMapping | 498 |
| CreateFunction | 504 |
| DeleteAlias | 513 |
| DeleteEventSourceMapping | 515 |
| DeleteFunction | 519 |
| DeleteFunctionConcurrency | 521 |
| DeleteFunctionEventInvokeConfig | 523 |
| DeleteLayerVersion | 525 |
| DeleteProvisionedConcurrencyConfig | 527 |
| GetAccountSettings | 529 |
| GetAlias | 531 |
| GetEventSourceMapping | 534 |
| GetFunction | 538 |
| GetFunctionConcurrency | 541 |
| GetFunctionConfiguration | 543 |
| GetFunctionEventInvokeConfig | 549 |
| GetLayerVersion | 552 |
| GetLayerVersionByArn | 555 |
| GetLayerVersionPolicy | 558 |
| GetPolicy | 560 |
| GetProvisionedConcurrencyConfig | 562 |
| Invoke | 565 |
| InvokeAsync | 570 |
| ListAliases | 572 |
| ListEventSourceMappings | 575 |
| ListFunctionEventInvokeConfigs | 578 |
| ListFunctions | 581 |
| ListLayers | 584 |
| ListLayerVersions | 586 |
| ListProvisionedConcurrencyConfigs | 589 |

| | |
|--|------------|
| ListTags | 592 |
| ListVersionsByFunction | 594 |
| PublishLayerVersion | 597 |
| PublishVersion | 601 |
| PutFunctionConcurrency | 608 |
| PutFunctionEventInvokeConfig | 611 |
| PutProvisionedConcurrencyConfig | 615 |
| RemoveLayerVersionPermission | 618 |
| RemovePermission | 620 |
| TagResource | 622 |
| UntagResource | 624 |
| UpdateAlias | 626 |
| UpdateEventSourceMapping | 630 |
| UpdateFunctionCode | 636 |
| UpdateFunctionConfiguration | 643 |
| UpdateFunctionEventInvokeConfig | 652 |
| Data Types | 655 |
| AccountLimit | 657 |
| AccountUsage | 658 |
| AliasConfiguration | 659 |
| AliasRoutingConfiguration | 661 |
| Concurrency | 662 |
| DeadLetterConfig | 663 |
| DestinationConfig | 664 |
| Environment | 665 |
| EnvironmentError | 666 |
| EnvironmentResponse | 667 |
| EventSourceMappingConfiguration | 668 |
| FunctionCode | 671 |
| FunctionCodeLocation | 672 |
| FunctionConfiguration | 673 |
| FunctionEventInvokeConfig | 678 |
| Layer | 680 |
| LayersListItem | 681 |
| LayerVersionContentInput | 682 |
| LayerVersionContentOutput | 683 |
| LayerVersionsListItem | 684 |
| OnFailure | 686 |
| OnSuccess | 687 |
| ProvisionedConcurrencyConfigListItem | 688 |
| TracingConfig | 690 |
| TracingConfigResponse | 691 |
| VpcConfig | 692 |
| VpcConfigResponse | 693 |
| Erreurs de certificat lors de l'utilisation d'un kit SDK | 693 |
| Glossaire AWS | 695 |

Présentation d'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 votre code dans l'un des [langages pris en charge par AWS Lambda \(p. 122\)](#).

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 générer des applications sans serveur composées de fonctions déclenchées par des événements, et les déployer automatiquement à l'aide d'CodePipeline et d'AWS CodeBuild. Pour de plus amples informations, veuillez consulter [Applications AWS Lambda \(p. 136\)](#).

Dans quels cas est-il conseillé d'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 et que vous utilisez 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 dans les environnements d'exécution fournis. 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. 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 découvrirez également le modèle de programmation et d'autres concepts Lambda. Pour plus d'informations, consultez [Mise en route avec AWS Lambda \(p. 3\)](#).
3. Lisez la section [Déploiement d'applications à l'aide d'AWS Lambda \(p. 136\)](#) 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 en savoir plus, consultez [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Mise en route avec AWS Lambda

Pour démarrer avec AWS Lambda, utilisez la console Lambda pour créer une fonction. En quelques minutes, vous pouvez créer et appeler une fonction, afficher des journaux, des métriques et des données de suivi.

Note

Pour utiliser Lambda et d'autres services AWS, vous avez besoin d'un compte AWS. Si vous n'avez pas de compte, visitez le site aws.amazon.com et choisissez Créer un compte AWS. Pour obtenir des instructions détaillées, consultez la page [Créer et activer un compte AWS](#).

Une bonne pratique consiste à créer un utilisateur AWS Identity and Access Management (IAM) avec des autorisations d'administrateur et de l'utiliser pour toutes les tâches qui ne nécessitent pas d'informations d'identification racine. Créez un mot de passe pour l'accès à la console, et des clés d'accès pour utiliser les outils de ligne de commande. Pour obtenir des instructions, consultez la page [Création de votre premier utilisateur et groupe administrateur IAM](#) dans le IAM Guide de l'utilisateur.

Vous pouvez créer des fonctions dans la console Lambda ou avec une boîte à outils pour environnement de développement intégré (IDE), des outils de ligne de commande ou des kits de développement logiciel (SDK). La console Lambda fournit un [éditeur de code \(p. 6\)](#) pour les langages non compilés qui vous permet de modifier et de tester le code rapidement. L'[AWS CLI \(p. 12\)](#) vous donne un accès direct à l'API Lambda pour la configuration avancée et les cas d'utilisation d'automatisation.

Rubriques

- [Créer une fonction Lambda avec la console \(p. 3\)](#)
- [Création de fonctions à l'aide de l'éditeur de la console AWS Lambda \(p. 6\)](#)
- [Utilisation d'AWS Lambda avec l'AWS Command Line Interface \(p. 12\)](#)
- [Concepts AWS Lambda \(p. 17\)](#)
- [Fonctions AWS Lambda \(p. 19\)](#)
- [Conseils d'utilisation de AWS Lambda \(p. 28\)](#)
- [Limites AWS Lambda \(p. 30\)](#)

Créer une fonction Lambda avec la console

Dans cet exercice de mise en route, vous créez une fonction Lambda à l'aide de la console AWS Lambda. Ensuite, vousappelez 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.

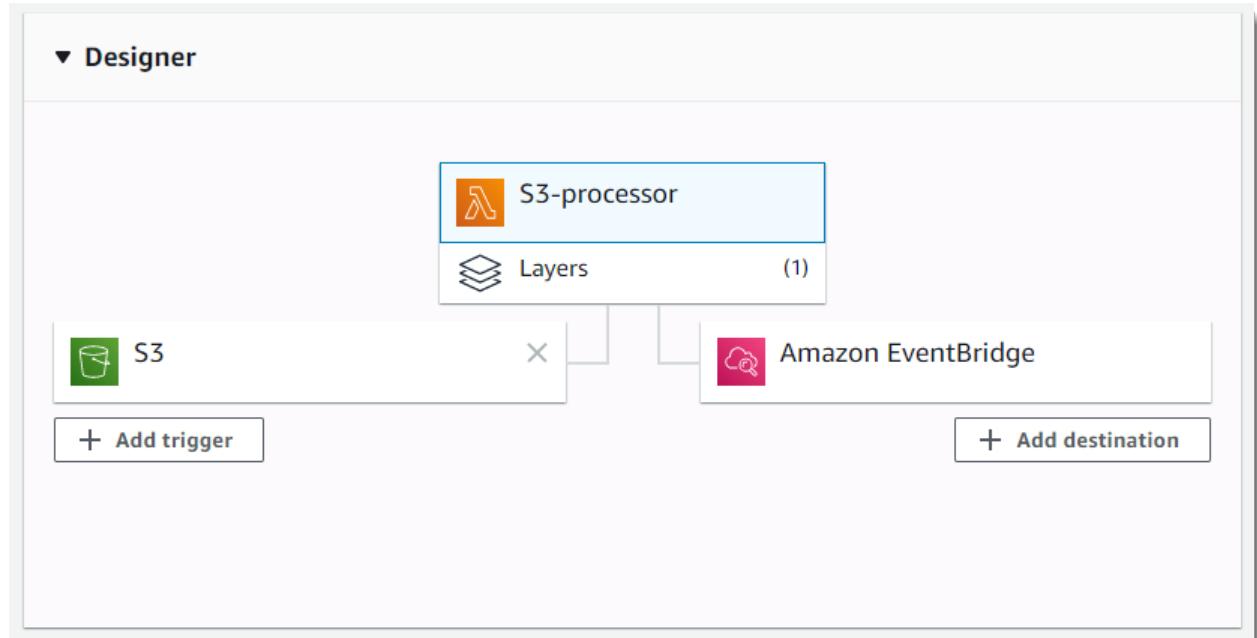
Pour créer une fonction Lambda

1. Ouvrez la [console AWS Lambda](#).
2. Choisissez Créer une fonction.
3. Pour Nom de la fonction, entrez **my-function**.
4. Sélectionnez Create function.

Lambda crée une fonction Node.js et un rôle d'exécution qui accorde à la fonction l'autorisation de charger des journaux. Lambda endosse le rôle d'exécution lorsque vousappelez votre fonction et l'utilise pour créer des informations d'identification pour le kit de développement logiciel (SDK) AWS et lire les données à partir des sources d'événements.

Utiliser le Designer (Concepteur)

Le Designer (Concepteur) présente une vue d'ensemble de votre fonction et de ses ressources en amont et en aval. Vous pouvez l'utiliser pour configurer des déclencheurs, des couches et des destinations.



Choisissez my-function dans le Designer pour revenir au code et à la configuration de la fonction. Pour les langages de script, Lambda inclut un exemple de code qui renvoie une réponse de réussite. Vous pouvez modifier le code de votre fonction avec l'éditeur [AWS Cloud9](#) intégré aussi longtemps que votre code source ne dépasse pas la limite 3 MB.

Appeler la fonction Lambda

Appelez votre fonction Lambda à l'aide de l'échantillon de données d'événements fourni dans la console.

Pour appeler une fonction

1. Dans le coin supérieur droit, choisissez Tester.
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"  
}
```

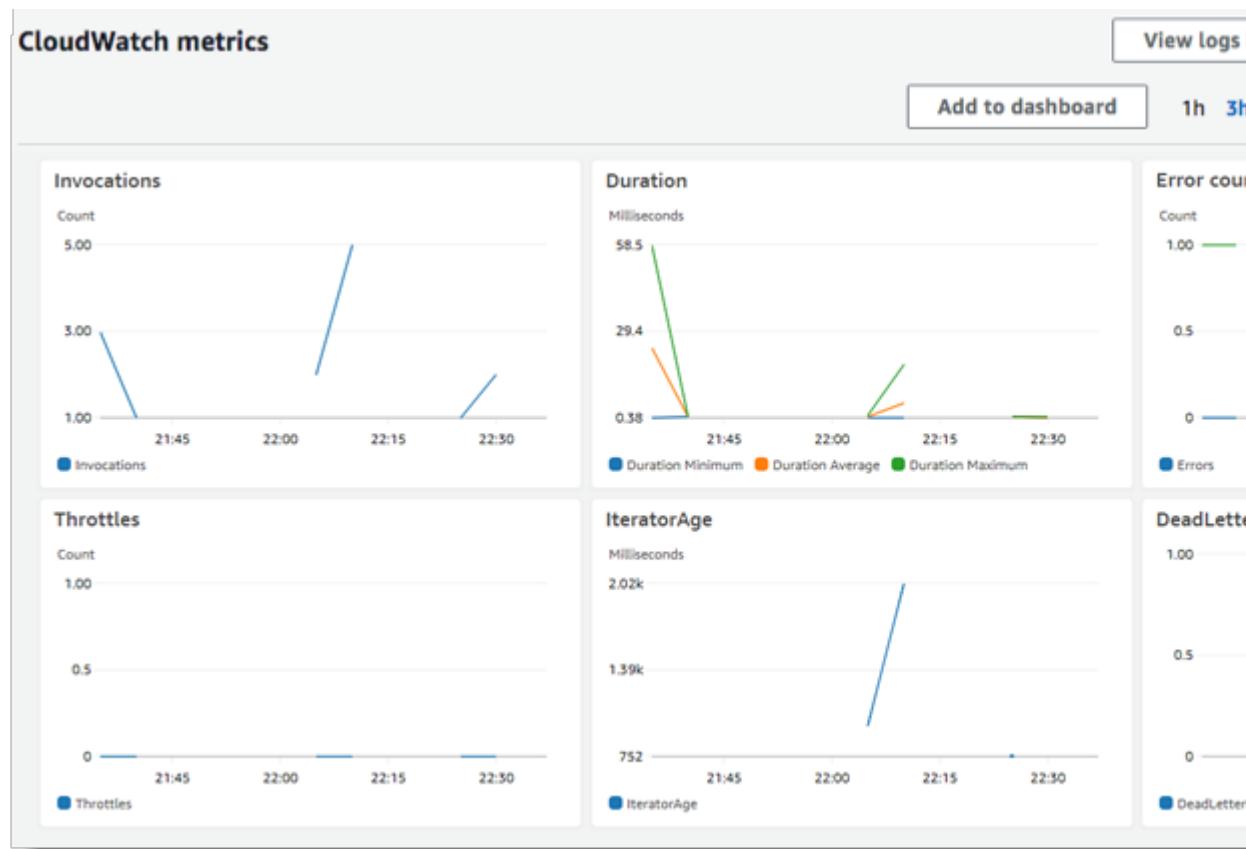
Vous pouvez modifier les clés et les valeurs de l'exemple de code JSON. Toutefois, ne modifiez pas la structure de l'événement. Si vous modifiez les clés et les valeurs, vous devez mettre à jour l'exemple de code en conséquence.

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.
 - 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`.
 - 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.

6. Exécutez la fonction Lambda plusieurs fois pour collecter des métriques que vous examinerez à l'étape suivante.
7. Choisissez Surveillance. Cette page affiche des graphiques de métriques que Lambda envoie à CloudWatch.



Pour en savoir plus sur ces graphiques, consultez la page [Surveillance des fonctions dans la console AWS Lambda \(p. 444\)](#).

Nettoyage

Si vous avez terminé d'utiliser l'exemple de fonction, supprimez-le. Vous pouvez également supprimer le rôle d'exécution créé par la console et le groupe de journaux qui stocke les journaux de la fonction.

Pour supprimer une fonction Lambda

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Actions, puis la fonction Supprimer.
4. Sélectionnez Supprimer.

Pour supprimer le groupe de journaux

1. Ouvrez la [page Groupes de journaux](#) de la console Amazon CloudWatch.
2. Choisissez le groupe de journaux de la fonction (/aws/lambda/my-function).
3. Choisissez Actions, puis Supprimer le groupe de journaux.
4. Sélectionnez Oui, supprimer.

Pour supprimer le rôle d'exécution

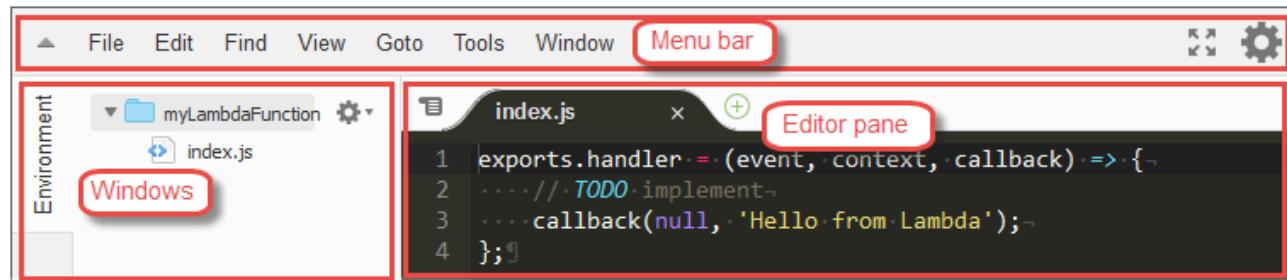
1. Accédez à la page [Rôles](#) de la console AWS Identity and Access Management.
2. Choisissez le rôle de la fonction (my-function-role-~~31example~~).
3. Choisissez Supprimer le rôle.
4. Choisissez Oui, supprimer.

Vous pouvez automatiser la création et le nettoyage des fonctions, des rôles et des groupes de journaux avec AWS CloudFormation et le AWS CLI. Pour obtenir des exemples d'applications entièrement fonctionnelles, veuillez consulter [Exemples d'applications Lambda \(p. 300\)](#).

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.



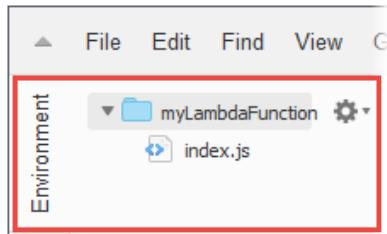
Pour obtenir la liste des rôles de chaque commande, consultez la [Référence des commandes de menu](#) dans le Guide de l'utilisateur AWS Cloud9. Remarque : Certaines des commandes répertoriées dans cette référence ne sont pas disponibles dans l'éditeur de code.

Rubriques

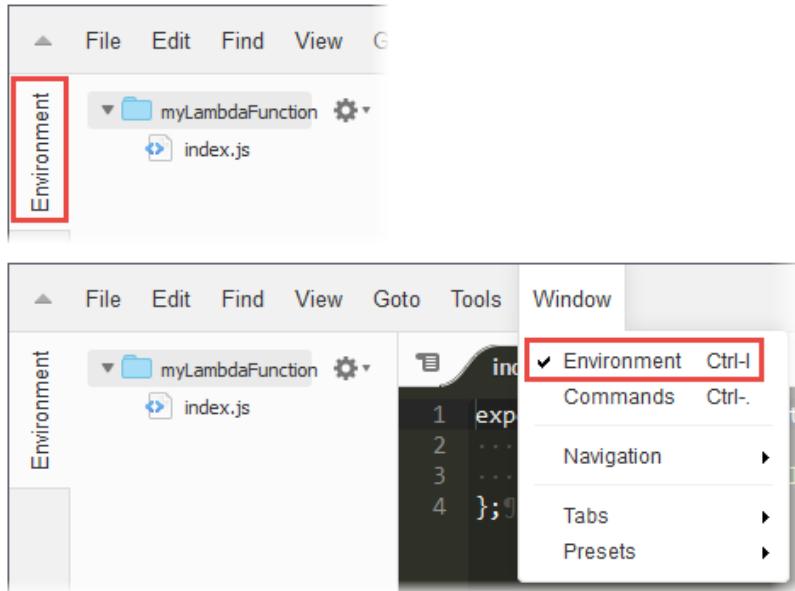
- [Utilisation de fichiers et de dossiers \(p. 7\)](#)
- [Utilisation du code \(p. 8\)](#)
- [Utilisation du mode plein écran \(p. 11\)](#)
- [Utilisation des préférences \(p. 11\)](#)

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.

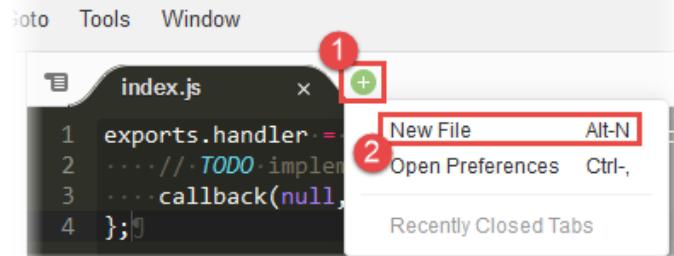


Pour ouvrir un seul fichier et afficher son contenu dans le volet de l'éditeur, double-cliquez sur le fichier dans la fenêtre Environment.

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.

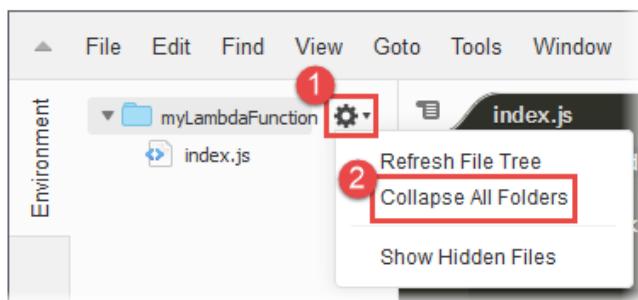
Pour enregistrer un fichier, avec le fichier ouvert et son contenu visible dans le volet de l'éditeur, sélectionnez File, Save dans la barre de menus.

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.

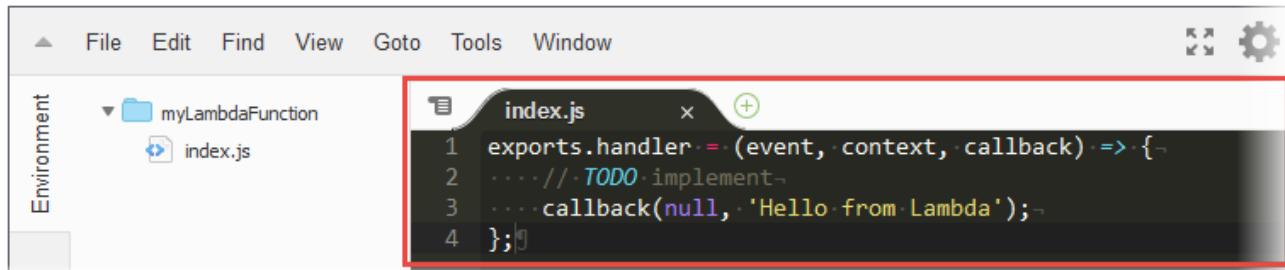
Pour réduire les dossiers, sélectionnez l'icône représentant un engrenage dans la fenêtre Environment, puis Collapse All Folders.



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.

Utilisation du code

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

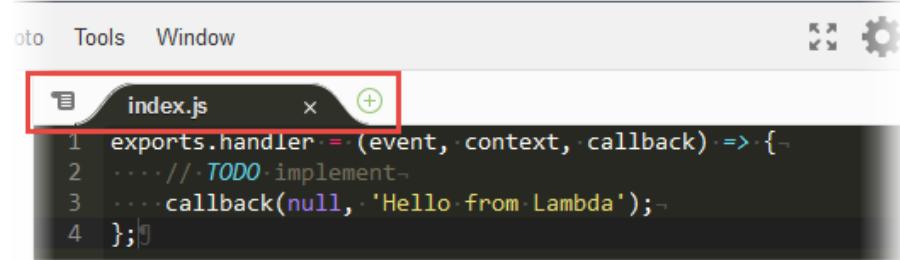


The screenshot shows the AWS Lambda developer interface. On the left, there's a sidebar labeled "Environment" with a folder icon "myLambdaFunction" and a file icon "index.js". The main area is a code editor with a dark theme. A red box highlights the tab bar where the "index.js" tab is selected. The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

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

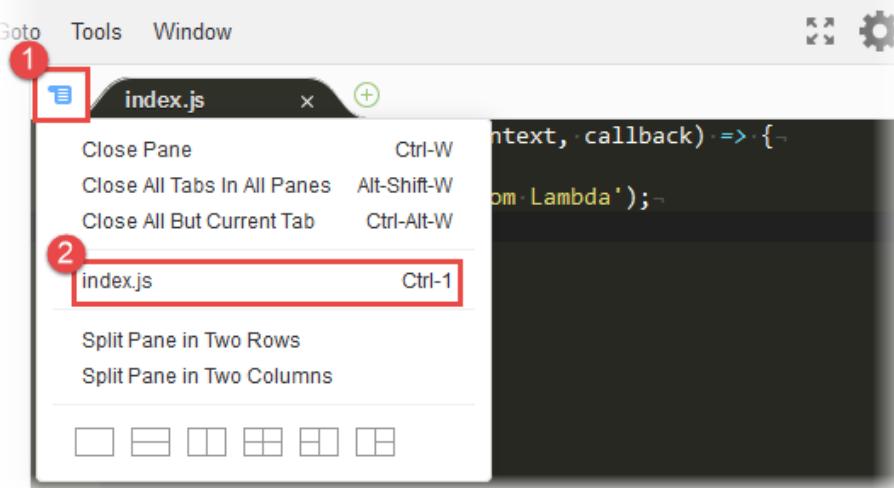
Utilisation des boutons de l'onglet

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.

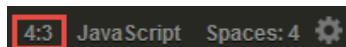
Pour créer un fichier, dans la barre de boutons de l'onglet, cliquez sur le bouton +, puis sélectionnez New File. Quand vous êtes prêt à enregistrer le fichier, sélectionnez File, Save ou File, Save As dans la barre de menus. Ensuite, dans la boîte de dialogue Save As qui s'affiche, nommez le fichier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

Utilisation de la barre d'état

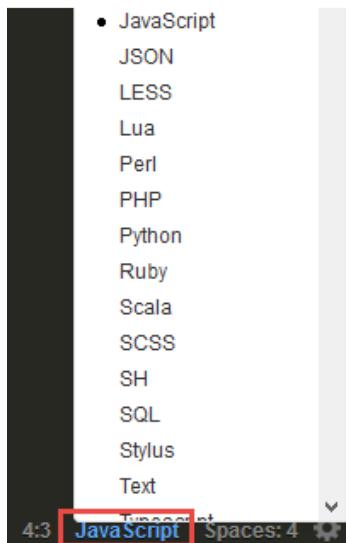
Utilisez la barre d'état pour accéder rapidement à une ligne du fichier actif et changer la façon dont s'affiche le code.



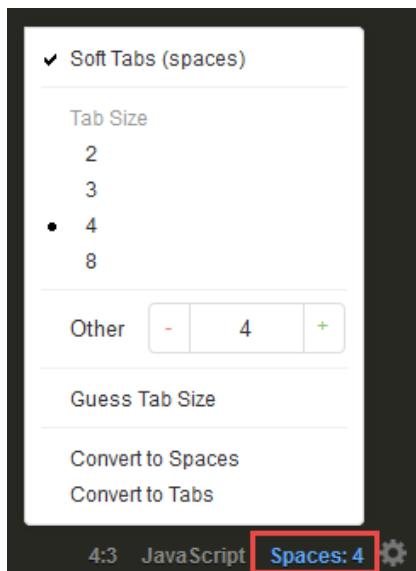
Pour passer rapidement à une ligne du fichier actif, choisissez le sélecteur de ligne, tapez le numéro de la ligne à atteindre, puis appuyez sur Enter.



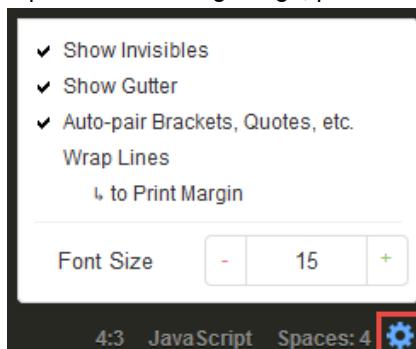
Pour modifier le modèle de couleurs du code dans le fichier actif, sélectionnez le sélecteur de modèle de couleurs du code, puis sélectionnez le nouveau modèle de couleurs du code.



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 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 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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Ce didacticiel utilise l'AWS Command Line Interface (AWS CLI) pour appeler les opérations d'API de service. Pour installer l'AWS CLI, consultez [Installation de l'interface de ligne de commande AWS](#) dans le AWS Command Line Interface Guide de l'utilisateur.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS. Pour créer un rôle d'exécution avec l'interface de ligne de commande AWS, utilisez la commande `create-role`.

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document file://trust-policy.json
{
    "Role": {
        "Path": "/",
        "RoleName": "lambda-ex",
        "RoleId": "AROAQFOXAMPL6TZ6ITKWND",
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
        "CreateDate": "2020-01-17T23:19:12Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Le fichier `trust-policy.json` est un fichier JSON dans le répertoire actuel qui définit la [stratégie d'approbation](#) du rôle. Cette stratégie d'approbation permet à Lambda d'utiliser les autorisations du rôle en attribuant au principal du service l'autorisation `lambda.amazonaws.com` pour appeler l'action `AssumeRole` d'AWS Security Token Service.

Example trust-policy.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Vous pouvez également spécifier la stratégie d'approbation en ligne. Les conditions d'utilisation de guillemets d'échappement dans la chaîne JSON varient en fonction de votre shell.

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document
'{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}}]'
```

Pour ajouter des autorisations au rôle, utilisez la commande `attach-policy-to-role`. Commencez par ajouter la stratégie gérée `AWSLambdaBasicExecutionRole`.

```
$ aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

La stratégie `AWSLambdaBasicExecutionRole` possède les autorisations dont la fonction a besoin pour écrire des journaux dans CloudWatch Logs.

Créer la fonction

L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

Example index.js

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.log("EVENT\n" + JSON.stringify(event, null, 2))
  return context.logStreamName
}
```

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`. Remplacez le texte en surbrillance dans l'ARN du rôle par votre ID de compte.

```
$ aws lambda create-function --function-name my-function \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-ex
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs12.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-ex",
  "Handler": "index.handler",
  "CodeSha256": "FpFMVUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
  ...
}
```

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 my-function out --log-type Tail
{
  "StatusCode": 200,
  "LogResult": "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
```

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
  "AWS_SESSION_TOKEN": "AgoJb3JpZ2lux2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    }
  ]
}
```

```
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration:
26.73 ms\tBilled Duration: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

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": "cli",
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "Runtime": "nodejs12.x",
            "Role": "arn:aws:iam::123456789012:role/lambda-ex",
            "Handler": "index.handler",
            ...
        },
        {
            "FunctionName": "random-error",
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:random-error",
            "Runtime": "nodejs12.x",
            "Role": "arn:aws:iam::123456789012:role/lambda-role",
            "Handler": "index.handler",
            ...
        },
        ...
    ],
    "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0="
}
```

En réponse, Lambda renvoie une liste de 10 fonctions maximum. S'il existe plus de fonctions à récupérer, `NextToken` fournit un marqueur que vous pourrez utiliser dans la demande `list-functions` suivante. La commande `list-functions` AWS CLI suivante est un exemple qui illustre le paramètre `--starting-token`.

```
$ aws lambda list-functions --max-items 10 --starting-
token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0=
```

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 my-function
{
    "Configuration": {
        "FunctionName": "my-function",
        "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
        ...
    }
}
```

```
"Runtime": "nodejs12.x",
"Role": "arn:aws:iam::123456789012:role/lambda-ex",
"CodeSha256": "FpFMvUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
"Version": "$LATEST",
"TracingConfig": {
    "Mode": "PassThrough"
},
"RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
...
},
"Code": {
    "RepositoryType": "S3",
    "Location": "https://awslambda-us-east-2-tasks.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-function-4203078a-b7c9-4f35..."
}
}
```

Pour plus d'informations, consultez [GetFunction \(p. 538\)](#).

Nettoyage

Exécutez la commande `delete-function` suivante pour supprimer la fonction `my-function`.

```
$ aws lambda delete-function --function-name my-function
```

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.

Concepts AWS Lambda

Avec AWS Lambda, vous exécutez des fonctions pour traiter des événements. Vous pouvez envoyer des événements à votre fonction en l'appelant avec l'API Lambda ou en configurant un service ou une ressource AWS pour le faire.

Concepts

- [Fonction \(p. 17\)](#)
- [Qualificateur \(p. 18\)](#)
- [Runtime \(p. 18\)](#)
- [Événement \(p. 18\)](#)
- [Concurrency \(p. 18\)](#)
- [Déclencheur \(p. 19\)](#)

Fonction

Une fonction est une ressource que vous pouvez appeler pour exécuter votre code dans AWS Lambda. Une fonction contient un code qui traite les événements et un environnement d'exécution qui transmet les demandes et les réponses entre Lambda et le code de la fonction. Vous fournissez le code et vous pouvez utiliser les environnements d'exécution ou créer les vôtres.

Pour de plus amples informations, veuillez consulter [Gestion des fonctions AWS Lambda \(p. 52\)](#).

Qualificateur

Lorsque vousappelez ou affichez une fonction, vous pouvez inclure un qualificateur pour spécifier une version ou un alias. Une version est un instantané immuable du code et de la configuration d'une fonction qui a un qualificateur numérique. Par exemple, `my-function:1`. Un alias est un pointeur vers une version qui peut être mise à jour pour un mappage avec une version différente ou pour le fractionnement du trafic entre deux versions. Par exemple, `my-function:BLUE`. Vous pouvez utiliser des versions et des alias ensemble pour fournir une interface stable permettant aux clients d'appeler votre fonction.

Pour de plus amples informations, veuillez consulter [Versions de fonction AWS Lambda \(p. 70\)](#).

Runtime

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 de plus amples informations, veuillez consulter [Environnements d'exécution AWS Lambda \(p. 122\)](#).

Événement

Un événement est un document au format JSON qui contient des données pour une fonction à traiter. Le runtime Lambda convertit l'événement en objet et la transmet au code de votre fonction. Lorsque vous appelez une fonction, vous déterminez la structure et le contenu de l'événement.

Example événement personnalisé – données météorologiques

```
{  
    "TemperatureK": 281,  
    "WindKmh": -3,  
    "HumidityPct": 0.55,  
    "PressureHPa": 1020  
}
```

Lorsqu'un service AWS appelle votre fonction, il définit la forme de l'événement.

Example événement de service – notification Amazon SNS

```
{  
    "Records": [  
        {  
            "Sns": {  
                "Timestamp": "2019-01-02T12:45:07.000Z",  
                "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",  
                "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
                "Message": "Hello from SNS!",  
                ...  
            }  
        }  
    ]  
}
```

Pour de plus amples informations sur les événements des services AWS, veuillez consulter [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Concurrency

La simultanéité est le nombre de demandes auxquelles votre fonction répond à un moment donné. Lorsque votre fonction est appelée, Lambda en fournit une instance pour traiter l'événement. Lorsque le code de la

fonction a fini de s'exécuter, il peut gérer une autre demande. Si la fonction est appelée à nouveau alors qu'une demande est toujours en cours de traitement, une autre instance est fournie, ce qui augmente la simultanéité de la fonction.

La simultanéité est soumise à des limites au niveau de la région. Vous pouvez également configurer des fonctions individuelles pour limiter leur simultanéité ou pour vous assurer qu'elles peuvent atteindre un niveau spécifique de simultanéité. Pour plus d'informations, consultez [Gestion de la simultanéité pour une fonction Lambda \(p. 61\)](#).

Déclencheur

Un déclencheur est une ressource ou une configuration qui appelle une fonction Lambda. Cela inclut les services AWS qui peuvent être configurés pour appeler une fonction, les applications que vous développez et les mappages de source d'événement. Un mappage de source d'événement est une ressource dans Lambda qui lit les éléments d'un flux ou d'une file d'attente et appelle une fonction.

Pour de plus amples informations, veuillez consulter [Appel de fonctions AWS Lambda \(p. 91\)](#) et [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Fonctions AWS Lambda

AWS Lambda fournit une console de gestion et une API pour la gestion et l'appel des fonctions. Il fournit des environnements d'exécution qui prennent en charge un ensemble standard de fonctionnalités, lesquelles vous permettent de passer facilement d'un langage à l'autre et d'une infrastructure à une autre en fonction de vos besoins. Outre les fonctions, vous pouvez également créer des versions, des alias, des couches et des environnements d'exécution personnalisés.

Fonctions

- [Modèle de programmation \(p. 19\)](#)
- [Package de déploiement \(p. 21\)](#)
- [Couches \(p. 21\)](#)
- [Dimensionnement \(p. 21\)](#)
- [Contrôles de simultanéité \(p. 22\)](#)
- [Appel asynchrone \(p. 24\)](#)
- [Mappages de source d'événement \(p. 25\)](#)
- [Destinations \(p. 26\)](#)
- [Plans de fonction \(p. 27\)](#)
- [Modèles d'application \(p. 28\)](#)

Modèle de programmation

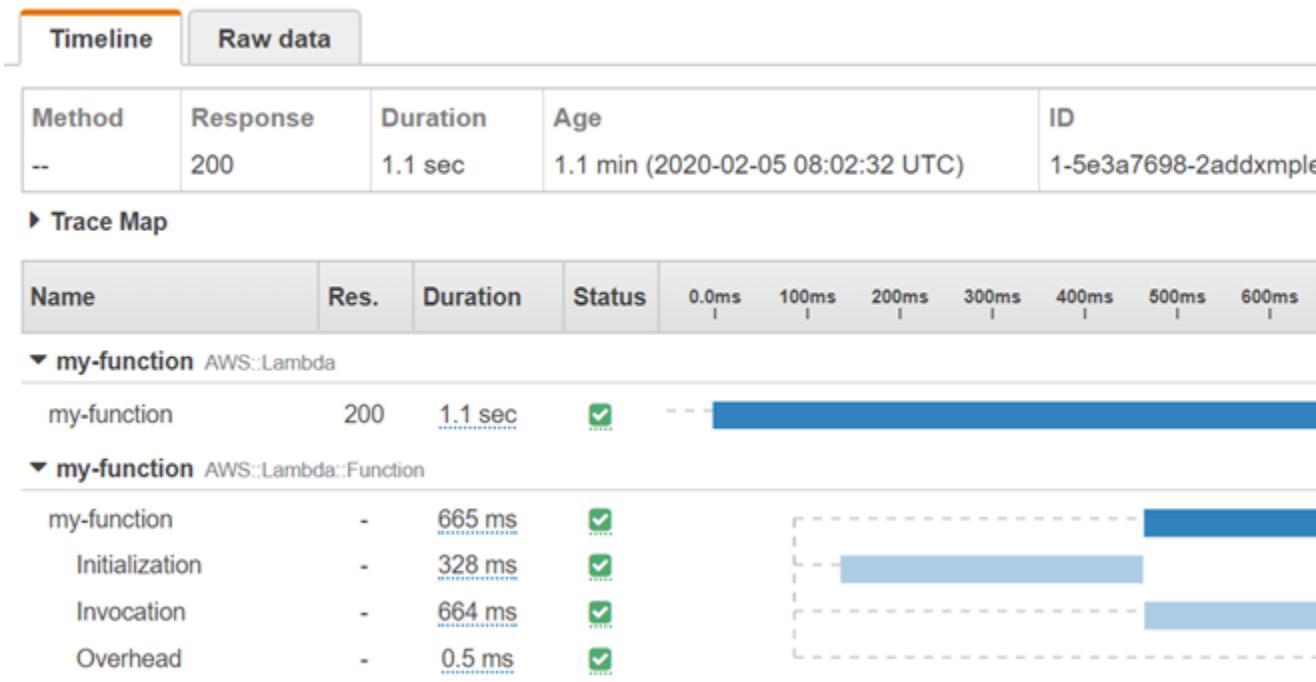
Les créations spécifiques varient d'un environnement d'exécution à l'autre, mais elles partagent toutes un modèle de programmation commun qui définit l'interface entre votre code et le code d'un environnement d'exécution. Vous indiquez à l'environnement d'exécution quelle méthode exécuter en définissant un gestionnaire dans la configuration de la fonction, et l'environnement d'exécution l'exécute. L'environnement d'exécution transmet les objets au gestionnaire qui contiennent l'événement d'appel et le contexte, tels que le nom de la fonction et l'ID de demande.

Lorsque le gestionnaire termine le traitement du premier événement, le runtime lui en envoie un autre. La classe de la fonction reste en mémoire, de sorte que les clients et variables déclarés en dehors de la

méthode du gestionnaire dans le code d'initialisation peuvent être réutilisés. Pour économiser du temps de traitement sur les événements suivants, créez des ressources réutilisables telles que des clients AWS SDK lors de l'initialisation. Une fois initialisée, chaque instance de votre fonction peut traiter des milliers de demandes.

L'initialisation est facturée dans le cadre de la durée du premier appel traité par une instance de votre fonction. Lorsque le [suivi X-Ray \(p. 296\)](#) est activé, l'environnement d'exécution enregistre des sous-segments distincts pour l'initialisation et l'exécution.

Traces > Details



Votre fonction a également accès au stockage local dans le répertoire `/tmp`. Les instances de votre fonction qui répondent aux demandes restent actives pendant quelques heures avant d'être recyclées.

L'environnement d'exécution capture la sortie de journalisation de votre fonction et l'envoie à Amazon CloudWatch Logs. En plus de consigner la sortie de votre fonction, le runtime consigne également les entrées lorsque l'exécution démarre et se termine. Cela inclut un journal de rapport avec l'ID de demande, la durée facturée, la durée d'initialisation et d'autres détails. Si votre fonction génère une erreur, le runtime renvoie cette erreur au mécanisme d'appel.

Note

La journalisation est soumise à des [limites CloudWatch Logs](#). Les données des journaux peuvent être perdues en raison de la limitation ou, dans certains cas, lorsqu'une instance de votre fonction est arrêtée.

Pour une introduction pratique au modèle de programmation dans le langage de programmation de votre choix, consultez les sections suivantes.

- [Création de fonctions Lambda avec Node.js \(p. 312\)](#)
- [Création de fonctions Lambda avec Python \(p. 329\)](#)
- [Création de fonctions Lambda avec Ruby \(p. 347\)](#)

- [Création de fonctions Lambda avec Java \(p. 363\)](#)
- [Création de fonctions Lambda avec Go \(p. 398\)](#)
- [Création de fonctions Lambda avec C# \(p. 413\)](#)
- [Création de fonctions Lambda avec PowerShell \(p. 435\)](#)

Lambda met à l'échelle votre fonction en exécutant des instances supplémentaires à mesure que la demande augmente et en arrêtant des instances à mesure que la demande diminue. Sauf indication contraire, les demandes entrantes peuvent être traitées dans le désordre ou simultanément. Stockez l'état de votre application dans d'autres services et ne comptez pas sur la longue durée de vie des instances de votre fonction. Utilisez le stockage local et les objets de niveau classe pour améliorer les performances, mais réduisez au minimum la taille de votre package de déploiement et la quantité de données que vous transférez vers l'environnement d'exécution.

Package de déploiement

Le code de votre fonction se compose de scripts ou de programmes compilés et leurs dépendances. Lorsque vous créez des fonctions dans la console Lambda ou un kit d'outils, le client crée une archive ZIP de votre code appelée un package de déploiement. Le client envoie ensuite le package au service Lambda. Lorsque vous gérez des fonctions avec l'API Lambda, les outils de ligne de commande ou les kits de développement logiciel (SDK), vous créez le package de déploiement. Vous devez également créer un package de déploiement manuellement pour les langages compilés et pour ajouter des dépendances à votre fonction.

Pour obtenir des instructions spécifiques au langage, consultez les sections suivantes :

- [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#)
- [Package de déploiement AWS Lambda dans Python \(p. 332\)](#)
- [Package de déploiement AWS Lambda en Ruby \(p. 350\)](#)
- [Package de déploiement AWS Lambda en Java \(p. 366\)](#)
- [Package de déploiement AWS Lambda dans Go \(p. 398\)](#)
- [Package de déploiement AWS Lambda dans C# \(p. 414\)](#)
- [Package de déploiement AWS Lambda dans PowerShell \(p. 436\)](#)

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 de plus amples informations, veuillez consulter [Couches AWS Lambda \(p. 76\)](#).

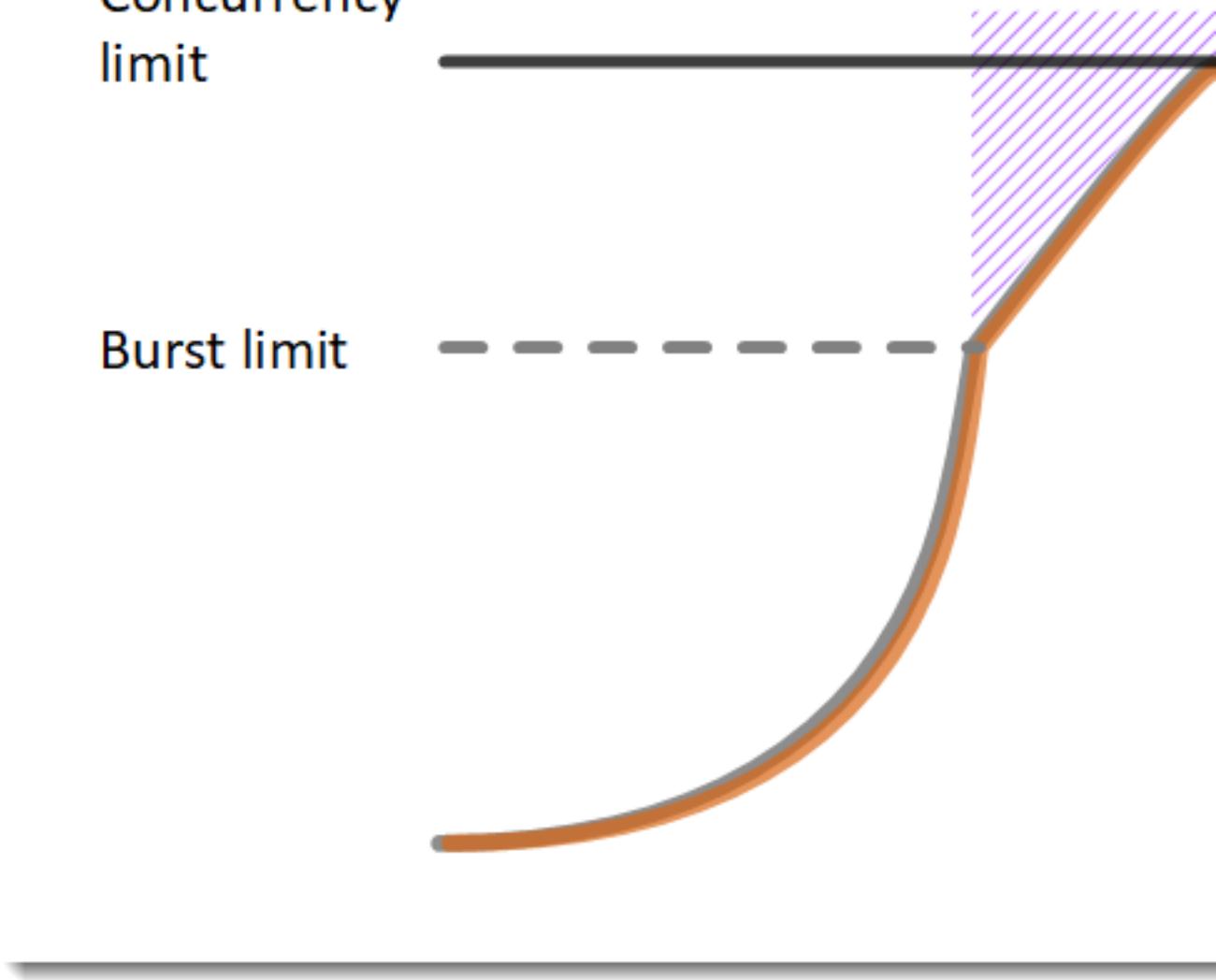
Dimensionnement

Lambda gère l'infrastructure qui exécute votre code et se dimensionne automatiquement en fonction des demandes entrantes. Lorsque votre fonction est appelée plus rapidement qu'une simple instance de votre fonction peut traiter les événements, Lambda évolue en exécutant des instances supplémentaires. Lorsque le trafic diminue, les instances inactives sont bloquées ou arrêtées. Vous payez uniquement pour la durée pendant laquelle votre fonction s'initialise ou traite des événements.

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

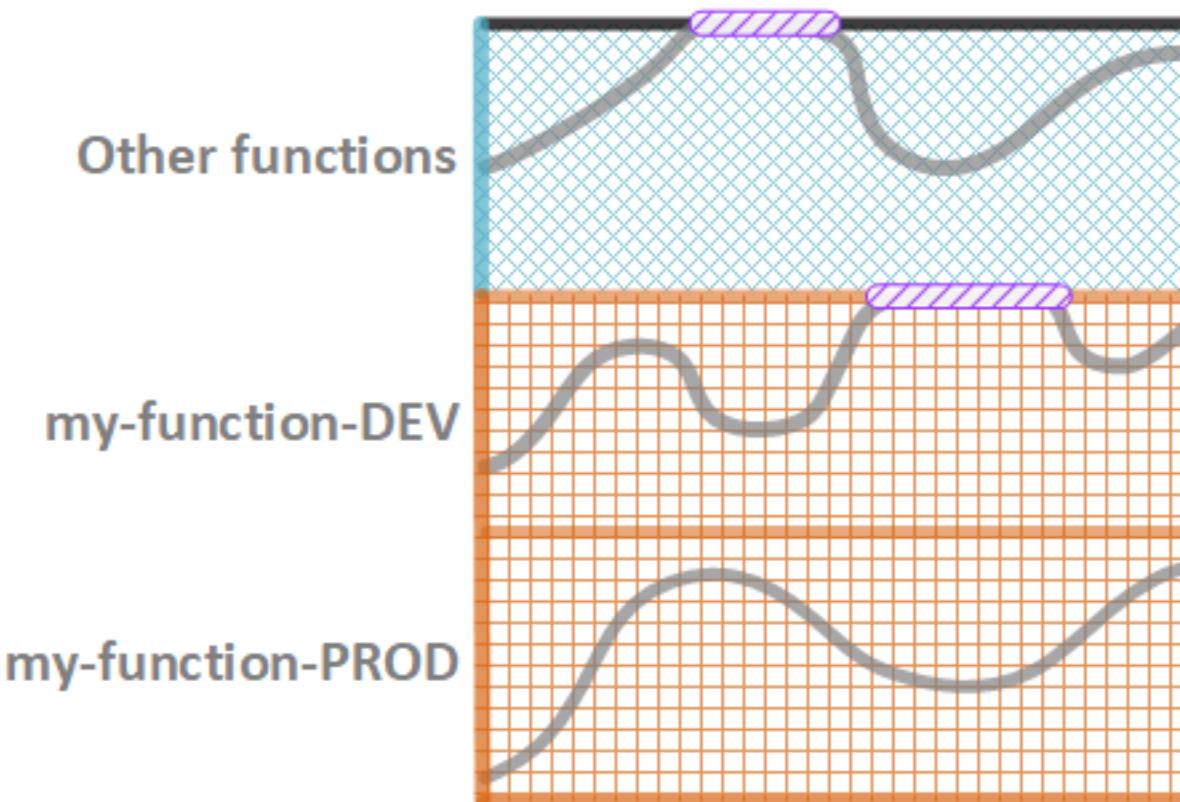


Pour de plus amples informations, veuillez consulter [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#).

Contrôles de simultanéité

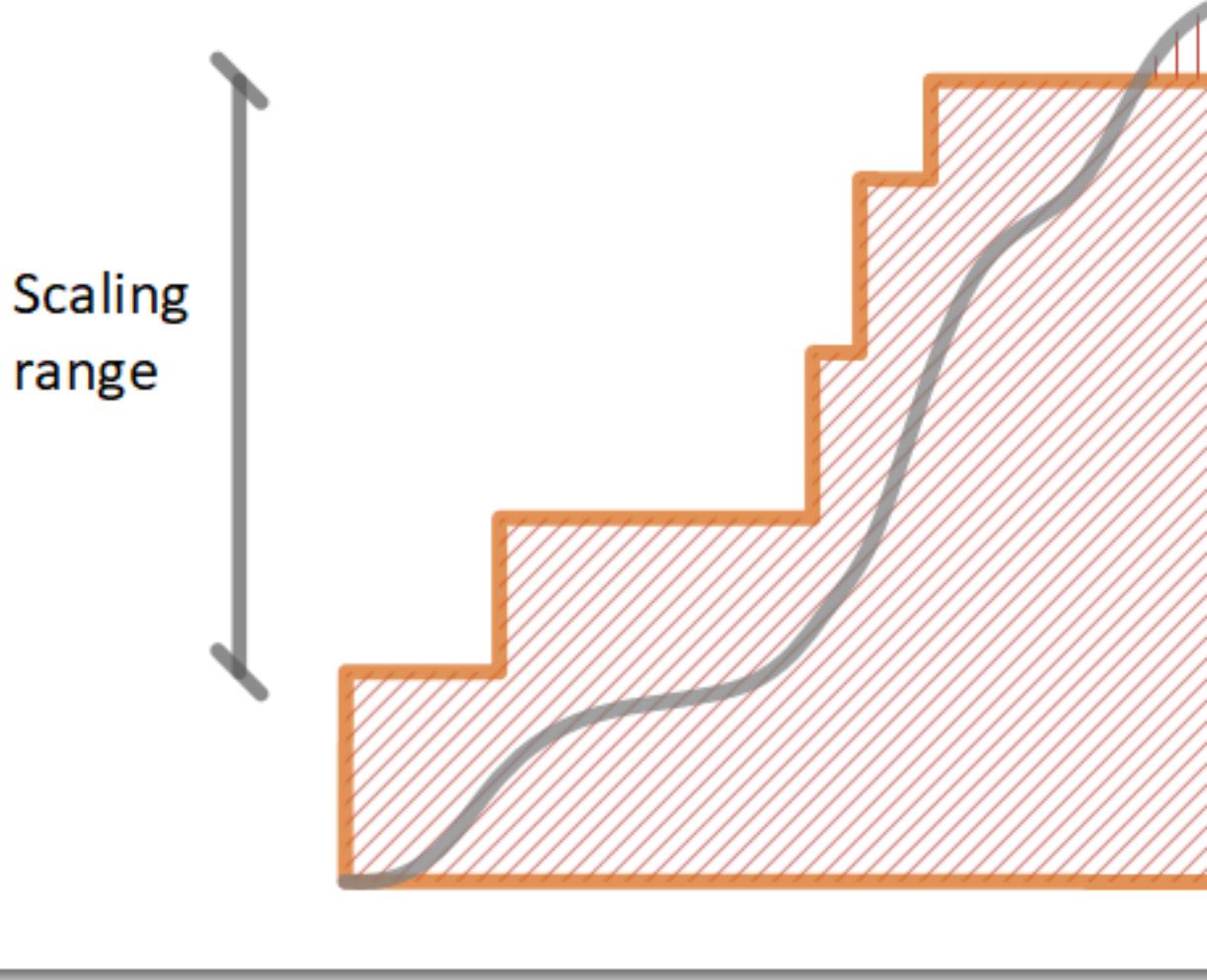
Utilisez les paramètres de simultanéité pour vous assurer que vos applications de production sont hautement disponibles et hautement réactives. Pour empêcher une fonction d'utiliser trop de simultanéité et réserver une partie de la simultanéité de votre compte disponible pour une fonction, utilisez la simultanéité réservée. La simultanéité réservée divise la simultanéité disponible en sous-ensembles. Une fonction avec simultanéité réservée utilise uniquement la simultanéité de l'ensemble qui lui est dédié.

Reserved Concurrency



Pour permettre l'évolutivité des fonctions sans fluctuation de latence, utilisez la simultanéité provisionnée. Pour les fonctions dont l'initialisation prend longtemps ou qui nécessitent une latence extrêmement faible pour tous les appels, la simultanéité provisionnée vous permet de pré-initialiser les instances de la fonction et d'en maintenir l'exécution à tout moment. Lambda s'intègre à Application Auto Scaling pour prendre en charge le dimensionnement automatique de la simultanéité provisionnée en fonction de l'utilisation.

Autoscaling with Provisioned Concurrency

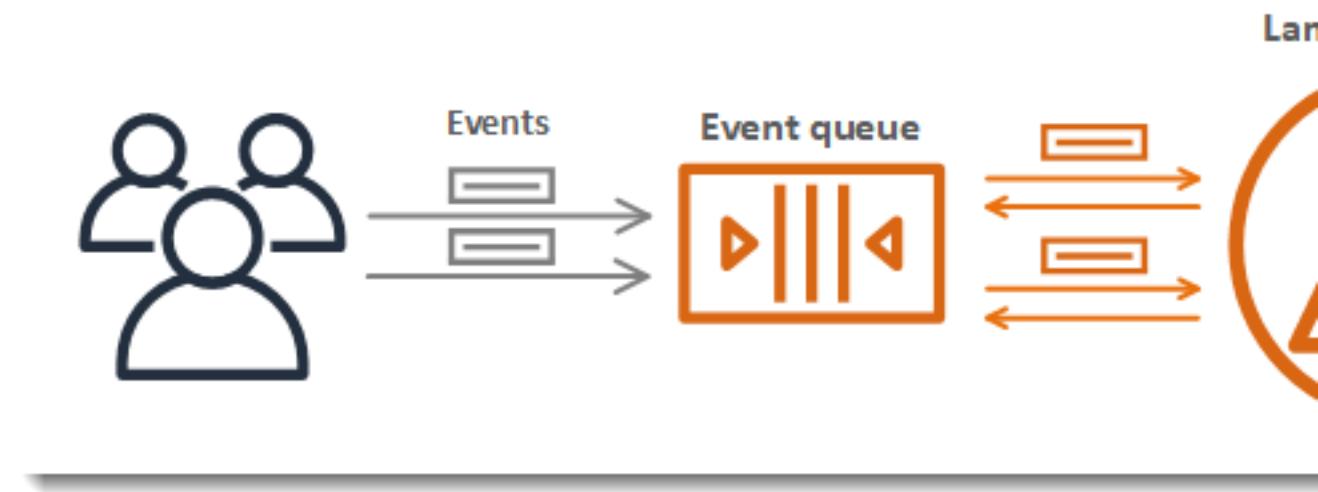


Pour plus d'informations, consultez [Gestion de la simultanéité pour une fonction Lambda \(p. 61\)](#).

Appel asynchrone

Lorsque vous appelez une fonction, vous pouvez choisir de le faire de façon synchrone ou asynchrone. Avec un [appel synchrone](#) (p. 92), vous attendez de la fonction qu'elle traite l'événement et renvoie une réponse. Avec l'appel asynchrone, Lambda place en file d'attente l'événement à traiter et renvoie une réponse immédiatement.

Asynchronous Invocation



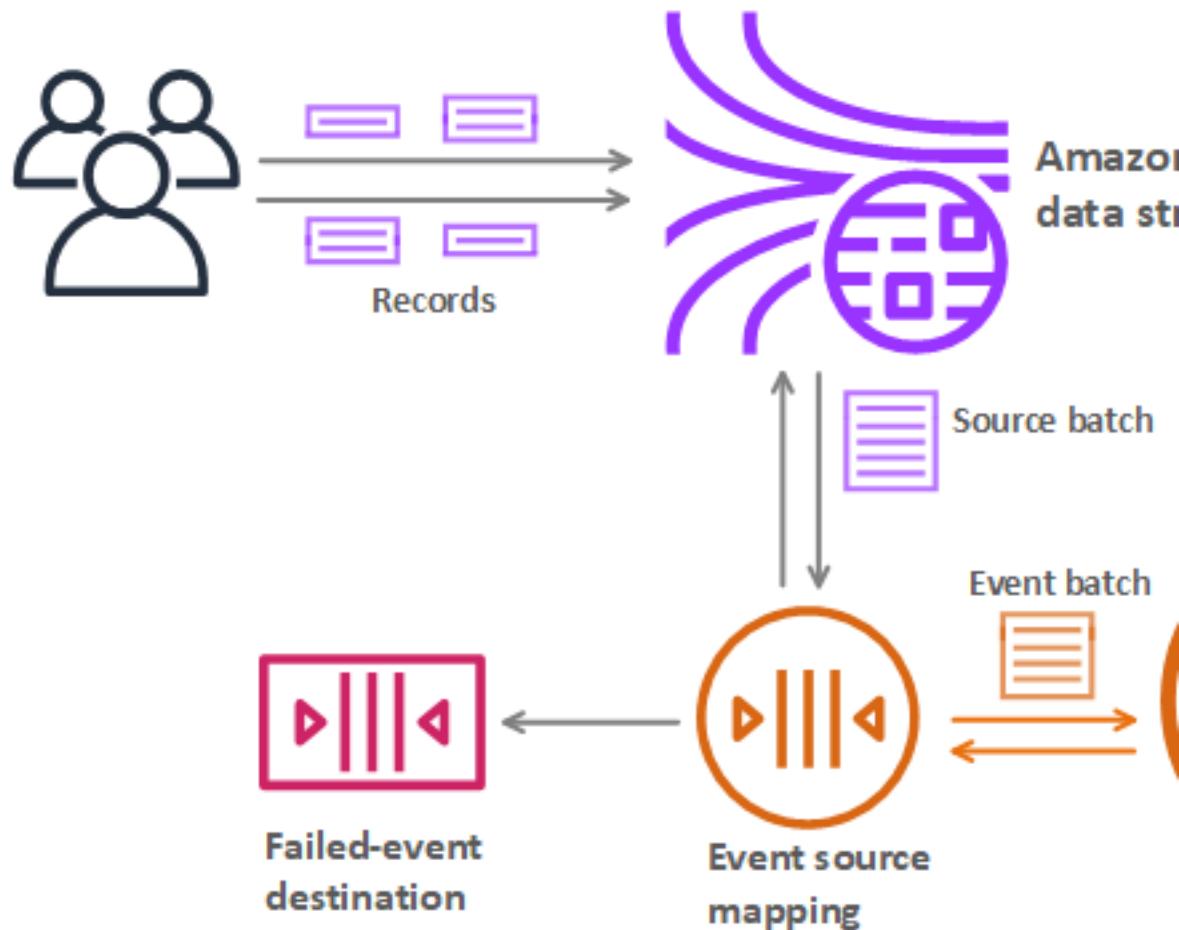
Pour les appels asynchrones, Lambda gère les nouvelles tentatives si la fonction renvoie une erreur ou si elle est limitée. Pour personnaliser ce comportement, vous pouvez configurer les paramètres de gestion des erreurs sur une fonction, une version ou un alias. Vous pouvez également configurer Lambda pour envoyer des événements dont le traitement a échoué vers une file d'attente de lettres mortes ou pour envoyer un enregistrement de chaque appel à une [destination](#) (p. 26).

Pour de plus amples informations, veuillez consulter [Appel asynchrone](#) (p. 93).

Mappages de source d'événement

Pour traiter les éléments à partir d'un flux ou d'une file d'attente, vous pouvez créer un [mappage de source d'événement](#) (p. 101). Un mappage de source d'événement est une ressource Lambda qui lit des éléments d'une file d'attente Amazon SQS, un flux Amazon Kinesis ou un flux Amazon DynamoDB et les envoie à votre fonction par lots. Chaque événement traité par votre fonction peut contenir des centaines ou des milliers d'éléments.

Event Source Mapping with Kinesis Stream



Les mappages de source d'événement gèrent une file d'attente locale d'éléments non traités, ainsi que les nouvelles tentatives si la fonction renvoie une erreur ou est limitée. Vous pouvez configurer un mappage de source d'événement pour personnaliser le comportement de traitement par lots et la gestion des erreurs, ou pour envoyer un enregistrement d'éléments dont le traitement échoue vers une [destination](#) (p. 26).

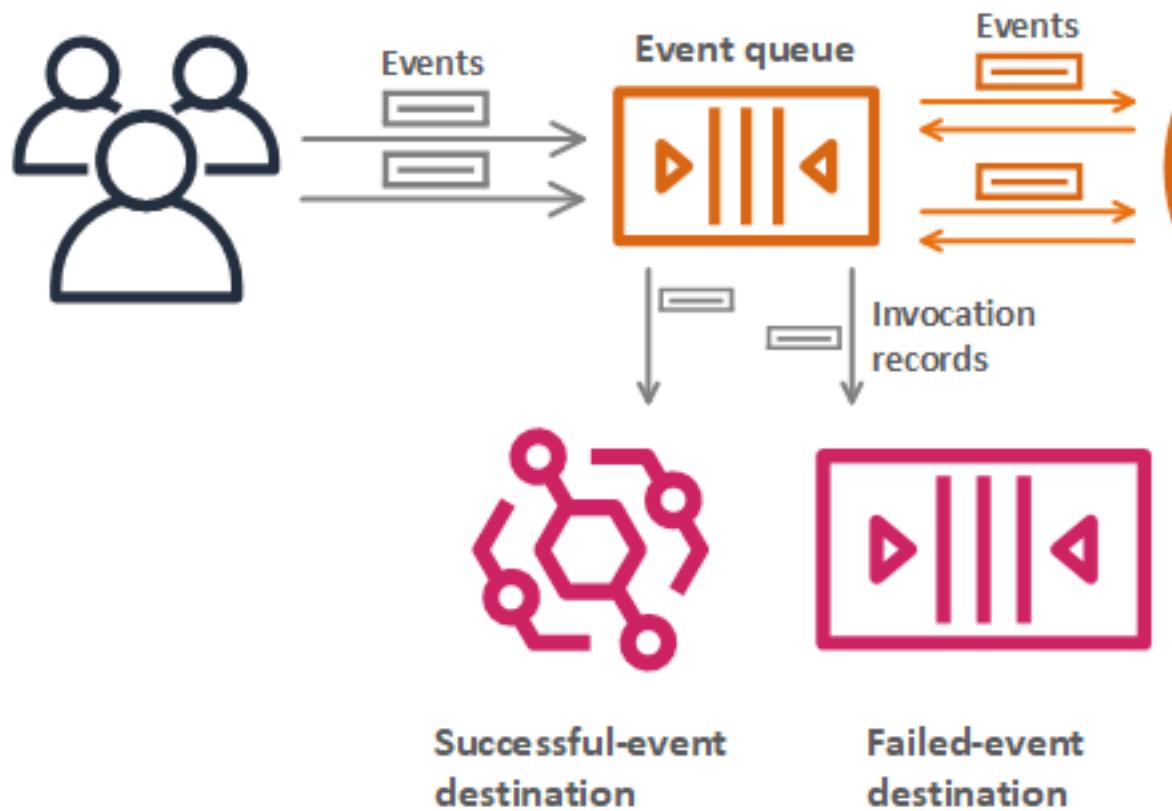
Pour de plus amples informations, veuillez consulter [Mappages de source d'événement AWS Lambda](#) (p. 101).

Destinations

Une destination est une ressource AWS qui reçoit des enregistrements d'appel pour une fonction. Pour les [appels asynchrones](#) (p. 24), vous pouvez configurer Lambda afin d'envoyer des enregistrements d'appel à une file d'attente, à une rubrique, à une fonction ou à un bus d'événements. Vous pouvez configurer des destinations distinctes pour les appels réussis et les événements dont le traitement a

échoué. L'enregistrement de l'appel contient des détails sur l'événement, la réponse à la fonction et la raison pour laquelle l'enregistrement a été envoyé.

Destinations for Asynchronous Invocation



Pour les [mappages de source d'événement \(p. 25\)](#) lus à partir de flux, vous pouvez configurer Lambda pour envoyer un enregistrement des lots ayant dont le traitement a échoué vers une file d'attente ou une rubrique. Un enregistrement d'échec pour un mappage de source d'événement contient des métadonnées sur le lot et pointe vers les éléments du flux.

Pour de plus amples informations, veuillez consulter [Configuration des destinations pour les appels asynchrones \(p. 96\)](#) ainsi que les sections relatives à la gestion des erreurs de [Utilisation de AWS Lambda avec Amazon DynamoDB \(p. 208\)](#) et de [Utilisation de AWS Lambda avec Amazon Kinesis \(p. 239\)](#).

Plans de fonction

Lorsque vous créez une fonction dans la console Lambda, vous pouvez choisir de commencer à partir de rien, d'utiliser un plan ou de déployer une application à partir du [AWS Serverless Application Repository](#). Un plan fournit un exemple de code qui montre comment utiliser Lambda avec un service AWS ou une

application tierce populaire. Les plans incluent des exemples de code et des préréglages de configuration de fonction pour les environnements d'exécution Node.js et Python.

Les plans sont fournis pour une utilisation sous la licence [Creative Commons Zero](#). Ils sont disponibles uniquement dans la console Lambda.

Modèles d'application

Vous pouvez utiliser la console Lambda pour créer une application avec un pipeline de distribution continue. Les modèles d'application de la console Lambda incluent du code pour une ou plusieurs fonctions, un modèle d'application qui définit les fonctions et les ressources AWS prises en charge, et un modèle d'infrastructure qui définit un pipeline AWS CodePipeline. Le pipeline comprend des étapes de construction et de déploiement qui s'exécutent chaque fois que vous poussez des modifications dans le référentiel Git inclus.

Les modèles d'application sont fournis pour une utilisation sous la licence [MIT No Attribution](#). Ils sont disponibles uniquement dans la console Lambda.

Pour plus d'informations, consultez [Création d'une application avec distribution continue dans la console Lambda \(p. 139\)](#).

Conseils d'utilisation de AWS Lambda

Outre la console Lambda, vous pouvez gérer et appeler des ressources Lambda grâce aux outils suivants.

Outils

- [AWS Command Line Interface \(p. 28\)](#)
- [Modèle d'application sans serveur AWS \(p. 28\)](#)
- [INTERFACE DE LIGNE DE COMMANDE SAM \(p. 29\)](#)
- [Outils de création de code \(p. 29\)](#)

AWS Command Line Interface

Installez l'AWS Command Line Interface pour gérer et utiliser des fonctions Lambda depuis l'interface de ligne de commande. Les didacticiels de ce guide utilisent l'AWS CLI, qui comporte des commandes pour toutes les actions d'API Lambda. Certaines fonctionnalités ne sont pas disponibles dans la console Lambda et sont uniquement accessibles à l'aide de l'AWS CLI ou du kit AWS SDK.

Pour configurer l'AWS CLI, 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](#)

Pour vérifier que la configuration de l'AWS CLI est correcte, exécutez la commande `list-functions` pour afficher une liste de vos fonctions Lambda dans la région actuelle.

```
$ aws lambda list-functions
```

Modèle d'application sans serveur AWS

AWS SAM est une extension du langage de modèle de langage 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.

Pour plus d'informations sur les modèles AWS SAM, consultez [Concepts de base des modèles AWS SAM](#) dans le Guide du développeur Modèle d'application sans serveur AWS.

INTERFACE DE LIGNE DE COMMANDE SAM

L'interface de ligne de commande AWS SAM est un outil de ligne de commande distinct que vous pouvez utiliser pour gérer et tester des applications AWS SAM. Outre les commandes permettant de charger des artefacts et de lancer des piles AWS CloudFormation également disponibles dans l'AWS CLI, l'interface de ligne de commande SAM offre des commandes supplémentaires permettant de valider des modèles et d'exécuter des applications localement dans un conteneur Docker.

Pour configurer l'interface de ligne de commande AWS SAM, consultez [Installation de l'interface de ligne de commande AWS SAM](#) dans le Guide du développeur Modèle d'application sans serveur AWS.

Outils de création de code

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 [Environnements d'exécution AWS Lambda \(p. 122\)](#). 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 |
| Java | <ul style="list-style-type: none">• Eclipse, avec AWS Toolkit pour Eclipse (voir Utilisation d'AWS Lambda avec AWS Toolkit pour Eclipse)• IntelliJ, avec le AWS Toolkit pour IntelliJ• Votre propre environnement de création |
| 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 |
| Python | <ul style="list-style-type: none">• console AWS Lambda• PyCharm, avec le AWS Toolkit pour PyCharm• Votre propre environnement de création |

| Langage | Outils et options pour la création de code |
|------------|---|
| Ruby | <ul style="list-style-type: none"> console AWS Lambda Votre propre environnement de création |
| Go | <ul style="list-style-type: none"> Votre propre environnement de création |
| 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 3.1 (voir Téléchargements .NET) Module AWSLambdaPSCore (voir PowerShell Gallery) |

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 | 1,000 |
| Stockage de couche et fonction | 75 GB |
| Elastic network interfaces par VPC (p. 81) | 250 |

Pour en savoir plus sur la simultanéité et la façon dont Lambda met à l'échelle votre simultanéité de fonction en réponse au trafic, veuillez consulter [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#).

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. 53) des fonctions | De 128 Mo à 3,008 MB, par incrément de 64 Mo |
| Délai d'expiration (p. 53) des fonctions. | 900 secondes (15 minutes) |
| Variables d'environnement (p. 55) des fonctions | 4 KB |
| Stratégie de fonction basée sur les ressources (p. 36) | 20 KB |
| Couches de fonctions (p. 76) | 5 couches |
| Simultanéité en rafale (p. 106) des fonctions | 500 – 3 000 (varie selon la région (p. 106)) |
| Fréquence d'appel par région (demandes par seconde) | Limite de 10 exécutions simultanées (synchrone (p. 92) – toutes les sources) Limite de 10 exécutions simultanées (asynchrone (p. 93) – sources non AWS) |

| Ressource | Limite |
|---|--|
| | Illimité (asynchrone – sources des services AWS (p. 155)) |
| Fréquence d'appel par version ou alias de fonction (demandes par seconde) | 10 x simultanéité allouée (p. 61) Cette limite s'applique uniquement aux fonctions qui utilisent la simultanéité allouée. |
| Charge utile d'appel (p. 91) (demande et réponse) | 6 MB (synchrone) 256 KB (asynchrone) |
| Taille du package de déploiement (p. 21) | 50 MB (compressé, en chargement direct) 250 MB (décompressé, y compris les couches) 3 MB (éditeur de console) |
| Événements de test (éditeur de console) | 10 |
| Stockage dans le répertoire <code>/tmp</code> | 512 MB |
| Descripteurs de fichier | 1,024 |
| Processus/threads d'exécution | 1,024 |

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](#) et [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Autorisations AWS Lambda

Vous pouvez utiliser AWS Identity and Access Management (IAM) pour gérer l'accès à l'API et aux ressources Lambda, telles les fonctions et les couches. Concernant les utilisateurs et les applications dans votre compte qui utilisent Lambda, vous gérez les autorisations dans une stratégie d'autorisations que vous pouvez appliquer aux utilisateurs, aux groupes ou aux rôles IAM. Pour accorder des autorisations à d'autres comptes ou à des services AWS qui utilisent vos ressources Lambda, il convient d'utiliser une stratégie qui s'applique à la ressource elle-même.

Une fonction Lambda est également dotée d'une stratégie, appelée un [rôle d'exécution \(p. 33\)](#), qui lui accorde l'autorisation d'accéder aux services et aux ressources AWS. Au minimum, la fonction doit pouvoir accéder à Amazon CloudWatch Logs pour la diffusion des journaux. Si vous [utilisez AWS X-Ray pour le suivi de votre fonction \(p. 296\)](#) ou si votre fonction accède aux services avec le kit SDK AWS, vous lui accordez l'autorisation de les appeler dans le rôle d'exécution. Lambda utilise également le rôle d'exécution afin d'obtenir l'autorisation de lecture dans les sources d'événements lorsque vous utilisez un [mappage de source d'événement \(p. 101\)](#) pour déclencher la fonction.

Note

Si un accès réseau à une ressource (telle une base de données relationnelle inaccessible via les API AWS ou Internet) est nécessaire pour votre fonction, [configurez cette dernière pour qu'elle se connecte à votre VPC \(p. 81\)](#).

Utilisez les [stratégies basées sur les ressources \(p. 36\)](#) pour accorder à d'autres comptes et services AWS l'autorisation d'utiliser vos ressources Lambda. Les ressources Lambda incluent des fonctions, des versions, des alias et des versions de couches. Chacune de ces ressources est dotée d'une stratégie d'autorisations qui s'applique en cas d'accès à la ressource, outre les stratégies qui s'appliquent à l'utilisateur. Lorsqu'un service AWS tel qu'Amazon S3 appelle votre fonction Lambda, la stratégie basée sur les ressources lui attribue l'accès.

Afin de gérer les autorisations pour les utilisateurs et les applications dans vos comptes, [utilisez les stratégies gérées fournies par Lambda \(p. 41\)](#) ou écrivez vos propres stratégies. La console Lambda utilise plusieurs services afin d'obtenir des informations sur la configuration et les déclencheurs de votre fonction. Vous pouvez utiliser les stratégies gérées telles quelles ou comme point de départ pour créer des stratégies plus restrictives.

Vous pouvez restreindre les autorisations utilisateur en fonction de la ressource affectée par une action et, dans certains cas, en fonction de conditions supplémentaires. Par exemple, vous pouvez spécifier un modèle pour l'ARN (Amazon Resource Name) d'une fonction qui requiert qu'un utilisateur inclue son nom d'utilisateur dans le nom des fonctions qu'il crée. Par ailleurs, vous pouvez ajouter une condition qui nécessite que l'utilisateur configure des fonctions de façon à utiliser une couche spécifique, par exemple pour extraire dans le logiciel de journalisation. Pour connaître les ressources et les conditions prises en charge par chaque action, consultez [Ressources et conditions \(p. 45\)](#).

Pour plus d'informations sur IAM, consultez [Qu'est-ce qu'IAM ?](#) dans le IAM Guide de l'utilisateur.

Rubriques

- [Rôle d'exécution AWS Lambda \(p. 33\)](#)
- [Utilisation de stratégies basées sur les ressources pour AWS Lambda \(p. 36\)](#)
- [Stratégies IAM basées sur l'identité pour AWS Lambda \(p. 41\)](#)

- [Ressources et conditions pour les actions Lambda \(p. 45\)](#)
- [Utilisation des limites d'autorisations pour les applications AWS Lambda \(p. 49\)](#)

Rôle d'exécution AWS Lambda

Le rôle d'exécution d'une fonction AWS Lambda lui accorde l'autorisation d'accéder aux services et aux ressources AWS. Vous fournissez ce rôle lorsque vous créez une fonction et Lambda endosse le rôle lorsque votre fonction est appelée. Vous pouvez créer un rôle d'exécution pour un développement autorisé à envoyer des journaux à Amazon CloudWatch et à charger des données de suivi vers AWS X-Ray.

Pour afficher le rôle d'exécution d'une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Permissions.
4. Le résumé des ressources affiche les services et ressources auxquels la fonction a accès. L'exemple suivant illustre les autorisations CloudWatch Logs que Lambda ajoute à un rôle d'exécution lorsque vous le créez dans la console Lambda.

Resource summary

 **Amazon CloudWatch Logs**
3 actions, 2 resources

To view the resources and actions that your function has permission to access, choose a service.

[By action](#) [By resource](#)

| Resource | Actions |
|---|------------|
| arn:aws:logs:us-east-2:123456789012:* | Allow: log |
| arn:aws:logs:us-east-2:123456789012:log-group:/aws/lambda/my-function:* | Allow: log |
| arn:aws:logs:us-east-2:123456789012:log-group:/aws/lambda/my-function:* | Allow: log |

ⓘ Lambda obtained this information from the following policy statements:

- Managed policy AWSLambdaBasicExecutionRole-493eae43-bffa-xmpl-9e86-cf39eeae586c,
- Managed policy AWSLambdaBasicExecutionRole-493eae43-bffa-xmpl-9e86-cf39eeae586c,

- Choisissez un service dans le menu déroulant pour afficher les autorisations associées à ce service.

À tout moment, vous pouvez ajouter ou supprimer des autorisations à partir du rôle d'exécution d'une fonction ou configurer votre fonction afin d'utiliser un autre rôle. Ajoutez des autorisations pour des services appelés par votre fonction avec le kit SDK AWS et pour des services utilisés par Lambda afin d'activer des fonctions facultatives.

Lorsque vous ajoutez des autorisations à votre fonction, mettez également à jour son code ou sa configuration. Cela force l'arrêt et le remplacement des instances en cours d'exécution de votre fonction, qui ont des informations d'identification obsolètes.

Création d'un rôle d'exécution dans la console IAM

Par défaut, Lambda crée un rôle d'exécution avec des autorisations minimales lorsque vous [créez une fonction \(p. 3\)](#) dans la console Lambda. Vous pouvez également créer un rôle d'exécution dans la console IAM.

Pour créer un rôle d'exécution dans la console IAM

- Ouvrez la page [Rôles](#) dans la console IAM.
- Choisissez [Créer un rôle](#).
- Sous Cas d'utilisation courants, choisissez Lambda.
- Choisissez [Étape suivante : autorisations](#).
- Sous Attacher des stratégies d'autorisations, choisissez les stratégies gérées `AWSLambdaBasicExecutionRole` et `AWSXRayDaemonWriteAccess`.
- Choisissez [Next: Tags \(Suivant : Balises\)](#).
- Choisissez [Next: Review \(Suivant : Vérification\)](#).
- Pour [Role name \(Nom du rôle\)](#), entrez `lambda-role`.
- Choisissez [Créer un rôle](#).

Pour obtenir des instructions détaillées, consultez [Création d'un rôle](#) dans le IAM Guide de l'utilisateur.

Gestion des rôles avec l'API IAM

Un rôle d'exécution est un rôle IAM qu'Lambda a l'autorisation d'assumer lorsque vous appelez une fonction. Pour créer un rôle d'exécution avec l'interface de ligne de commande AWS, utilisez la commande `create-role`.

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document file://trust-policy.json
{
    "Role": {
        "Path": "/",
        "RoleName": "lambda-ex",
        "RoleId": "AROAQFOXMP6TZ6ITKWND",
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
        "CreateDate": "2020-01-17T23:19:12Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    }
                }
            ]
        }
    }
}
```

```
        },
        "Action": "sts:AssumeRole"
    ]
}
}
```

Le fichier `trust-policy.json` est un fichier JSON dans le répertoire actuel qui définit la [stratégie d'approbation](#) du rôle. Cette stratégie d'approbation permet à Lambda d'utiliser les autorisations du rôle en attribuant au principal du service l'autorisation `lambda.amazonaws.com` pour appeler l'action `AssumeRole` d'AWS Security Token Service.

Example trust-policy.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Vous pouvez également spécifier la stratégie d'approbation en ligne. Les conditions d'utilisation de guillemets d'échappement dans la chaîne JSON varient en fonction de votre shell.

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document
'{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}}]'
```

Pour ajouter des autorisations au rôle, utilisez la commande `attach-policy-to-role`. Commencez par ajouter la stratégie gérée `AWSLambdaBasicExecutionRole`.

```
$ aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Stratégies gérées pour les fonctionnalités Lambda

Les stratégies gérées ci-après fournissent les autorisations requises pour utiliser les fonctions Lambda suivantes :

- `AWSLambdaBasicExecutionRole` – Autorisation de charger des journaux vers CloudWatch.
- `AWSLambdaKinesisExecutionRole` – Autorisation de lire des événements à partir d'un flux de données ou d'un consommateur Amazon Kinesis.
- `AWSLambdaDynamoDBExecutionRole` – Autorisation de lire des enregistrements depuis un flux Amazon DynamoDB.
- `AWSLambdaSQSQueueExecutionRole` – Autorisation de lire un message d'une file d'attente Amazon Simple Queue Service (Amazon SQS).
- `AWSLambdaVPCAccessExecutionRole` – Autorisation de gérer les interfaces réseau Elastic afin de connecter la fonction à un VPC.

- AWSXRayDaemonWriteAccess– Autorisation de télécharger des données de trace vers X-Ray.

Pour certaines fonctionnalités, la console Lambda tente d'ajouter les autorisations manquantes au rôle d'exécution dans une stratégie gérée par le client. Ces stratégies peuvent être excessivement nombreuses. Ajoutez les stratégies gérées pertinentes au rôle d'exécution avant d'activer les fonctionnalités, afin d'éviter la création de stratégies supplémentaires.

Lorsque vous utilisez un [mappage de source d'événement \(p. 101\)](#) pour appeler votre fonction, Lambda utilise le rôle d'exécution en vue de lire les données d'événement. Par exemple, un mappage de source d'événement pour Amazon Kinesis lit les événements d'un flux de données et les envoie à votre fonction par lots. Vous pouvez utiliser les mappages de source d'événement avec les services suivants :

Services à partir desquels Lambda lit les événements

- [Amazon Kinesis \(p. 239\)](#)
- [Amazon DynamoDB \(p. 208\)](#)
- [Amazon Simple Queue Service \(p. 285\)](#)

Outre les stratégies gérées, la console Lambda fournit des modèles permettant de créer une stratégie personnalisée dotée des autorisations liées à d'autres cas d'utilisation. Lorsque vous créez une fonction dans la console Lambda, vous pouvez choisir de créer un autre rôle d'exécution doté des autorisations issues d'un ou de plusieurs modèles. Ces modèles s'appliquent également automatiquement lorsque vous créez une fonction à partir d'un plan ou lorsque vous configurez les options qui nécessitent l'accès à d'autres services. Vous trouverez des exemples de modèles dans le [référentiel GitHub](#) de ce guide.

Utilisation de stratégies basées sur les ressources pour AWS Lambda

AWS Lambda prend en charge les stratégies d'autorisations basées sur les ressources pour les fonctions et les couches Lambda. Les stratégies basées sur les ressources permettent d'accorder une autorisation à d'autres comptes en fonction des ressources. Vous pouvez également appliquer une stratégie basée sur les ressources pour autoriser un service AWS à appeler votre fonction.

Pour les fonctions Lambda, vous pouvez [accorder une autorisation de compte \(p. 38\)](#) afin d'appeler ou de gérer une fonction. Vous pouvez ajouter plusieurs déclarations afin d'accorder l'accès à plusieurs comptes ou laisser n'importe quel compte appeler votre fonction. Pour les fonctions appelées par un autre service AWS en réponse à une activité dans votre compte, vous utilisez la stratégie pour [accorder l'autorisation d'appeler le service \(p. 38\)](#).

Pour afficher la stratégie basée sur les ressources d'une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Permissions.
4. La stratégie basée sur les ressources affiche les autorisations qui sont appliquées lorsqu'un autre compte ou service AWS tente d'accéder à la fonction. L'exemple suivant montre une déclaration qui autorise Amazon S3 à appeler une fonction nommée `my-function` pour un compartiment nommé `my-bucket` dans le compte `123456789012`.

Resource-based policy [Info](#)

```
1  {  
2      "Version": "2012-10-17",  
3      "Id": "default",  
4      "Statement": [  
5          {  
6              "Sid": "lambda-d49ff629-xmpl-454d-8473-04c11fdc424c",  
7              "Effect": "Allow",  
8              "Principal": {  
9                  "Service": "s3.amazonaws.com"  
10             },  
11             "Action": "lambda:InvokeFunction",  
12             "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
13             "Condition": {  
14                 "StringEquals": {  
15                     "AWS:SourceAccount": "123456789012"  
16                 },  
17                 "ArnLike": {  
18                     "AWS:SourceArn": "arn:aws:s3:::my-bucket"  
19                 }  
20             }  
21         }  
22     ]  
23 }
```

Pour les couches Lambda, vous utilisez une stratégie basée sur les ressources sur une version de la couche afin que les autres comptes puissent l'utiliser. Outre les stratégies qui accordent l'autorisation à un seul compte ou à tous les comptes, vous pouvez aussi, pour les couches, accorder une autorisation à tous les comptes d'une organisation.

Note

Vous pouvez uniquement mettre à jour les stratégies basées sur des ressources pour les ressources Lambda comprises dans la portée des actions d'API [AddPermission \(p. 490\)](#) et [AddLayerVersionPermission \(p. 487\)](#). Vous ne pouvez pas créer de stratégies pour vos ressources Lambda dans JSON ni utiliser les conditions qui ne correspondent pas aux paramètres de ces actions.

Les stratégies basées sur les ressources s'appliquent à une seule fonction, version ou version de couche ou à un seul alias. Elles accordent l'autorisation à un ou à plusieurs services et comptes. Pour les comptes approuvés qui doivent pouvoir accéder à plusieurs ressources ou pour utiliser les actions d'API non prises en charge par les stratégies basées sur les ressources, vous pouvez utiliser les [rôles entre comptes \(p. 41\)](#).

Rubriques

- [Attribution de l'accès à la fonction aux services AWS \(p. 38\)](#)
- [Octroi de l'accès à la fonction à d'autres comptes \(p. 38\)](#)
- [Octroi de l'accès de la couche à d'autres comptes \(p. 40\)](#)

- Nettoyage des stratégies basées sur les ressources (p. 40)

Attribution de l'accès à la fonction aux services AWS

Lorsque vous [utilisez un service AWS pour appeler votre fonction \(p. 155\)](#), vous accordez l'autorisation dans une déclaration sur une stratégie basée sur les ressources. Vous pouvez appliquer la déclaration à la fonction ou la limiter à une seule version ou à un seul alias.

Note

Lorsque vous ajoutez un déclencheur à la fonction à l'aide de la console Lambda, celle-ci met à jour la stratégie basée sur les ressources de la fonction afin d'autoriser le service à l'appeler. Pour accorder des autorisations à d'autres comptes ou services non disponibles dans la console Lambda, utilisez l'interface de ligne de commande AWS.

Ajoutez une déclaration avec la commande `add-permission`. La déclaration de stratégie basée sur les ressources la plus simple autorise un service à appeler une fonction. La commande ci-après accorde à Amazon SNS l'autorisation d'appeler une fonction nommée `my-function`.

```
$ aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --statement-id sns \
--principal sns.amazonaws.com --output text
{"Sid":"sns","Effect":"Allow","Principal": \
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-2:123456789012:function:my-function"}
```

Amazon SNS peut ainsi appeler la fonction, mais la rubrique Amazon SNS qui déclenche l'appel n'est pas restreinte. Afin de vous assurer que votre fonction est appelée uniquement par une ressource spécifique, spécifiez l'ARN (Amazon Resource Name) de la ressource avec l'option `source-arn`. La commande ci-après autorise uniquement Amazon SNS à appeler la fonction pour des abonnements à une rubrique nommée `my-topic`.

```
$ aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --statement-id sns-my-topic \
--principal sns.amazonaws.com --source-arn arn:aws:sns:us-east-2:123456789012:my-topic
```

Certains services peuvent appeler des fonctions dans d'autres comptes. Cela ne pose pas de problème si vous spécifiez un ARN source qui contient votre ID de compte. Pour Amazon S3, cependant, la source est un compartiment dont l'ARN ne contient pas d'ID de compte. Il est possible que vous puissiez supprimer le compartiment et qu'un autre compte crée un compartiment du même nom. Utilisez l'option `account-id` pour vous assurer que seules les ressources de votre compte peuvent appeler la fonction.

```
$ aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --statement-id s3-account \
--principal s3.amazonaws.com --source-arn arn:aws:s3:::my-bucket-123456 --source-account 123456789012
```

Octroi de l'accès à la fonction à d'autres comptes

Pour accorder des autorisations à un autre compte AWS, spécifiez l'ID du compte comme paramètre `principal`. L'exemple suivant accorde au compte 210987654321 l'autorisation d'appeler la fonction `my-function` avec l'alias `prod`.

```
$ aws lambda add-permission --function-name my-function:prod --statement-id xaccount --action lambda:InvokeFunction \
```

```
--principal 210987654321 --output text
{"Sid":"xaccount","Effect":"Allow","Principal":*
{"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function"}
```

La stratégie basée sur les ressources accorde à l'autre compte l'autorisation d'accéder à la fonction, mais n'autorise pas les utilisateurs de ce compte à dépasser leurs autorisations. Les utilisateurs de l'autre compte doivent disposer des [autorisations utilisateur \(p. 41\)](#) correspondantes pour utiliser l'API Lambda.

Pour limiter l'accès à un utilisateur, un groupe ou un rôle dans un autre compte, spécifiez l'ARN complet de l'identité en tant que mandataire. Par exemple, `arn:aws:iam::123456789012:user/developer`.

L'[alias \(p. 72\)](#) limite la version pouvant être appelée par l'autre compte. L'autre compte doit alors inclure l'alias dans l'ARN de la fonction.

```
$ aws lambda invoke --function-name arn:aws:lambda:us-west-2:123456789012:function:my-
function:prod out
{
    "StatusCode": 200,
    "ExecutedVersion": "1"
}
```

Vous pouvez ensuite mettre à jour l'alias pour qu'il pointe vers de nouvelles versions, en fonction de vos besoins. Lorsque vous mettez à jour l'alias, l'autre compte n'a pas besoin de changer son code pour utiliser la nouvelle version et il est uniquement autorisé à appeler la version que vous choisissez.

Vous pouvez accorder l'accès entre comptes pour n'importe quelle action d'API qui [s'exécute sur une fonction existante \(p. 47\)](#). Par exemple, vous pouvez accorder l'accès à `lambda>ListAliases` afin qu'un compte puisse obtenir une liste des alias ou à `lambda:GetFunction` pour qu'il puisse télécharger le code de votre fonction. Ajoutez chaque autorisation séparément ou utilisez `lambda:*` pour accorder l'accès à toutes les actions pour la fonction spécifiée.

API inter-comptes

- [Invoke \(p. 565\)](#)
- [GetFunction \(p. 538\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [UpdateFunctionCode \(p. 636\)](#)
- [DeleteFunction \(p. 519\)](#)
- [PublishVersion \(p. 601\)](#)
- [ListVersionsByFunction \(p. 594\)](#)
- [CreateAlias \(p. 494\)](#)
- [GetAlias \(p. 531\)](#)
- [ListAliases \(p. 572\)](#)
- [UpdateAlias \(p. 626\)](#)
- [DeleteAlias \(p. 513\)](#)
- [GetPolicy \(p. 560\)](#)
- [PutFunctionConcurrency \(p. 608\)](#)
- [DeleteFunctionConcurrency \(p. 521\)](#)
- [ListTags \(p. 592\)](#)
- [TagResource \(p. 622\)](#)
- [UntagResource \(p. 624\)](#)

Pour accorder à d'autres comptes l'autorisation d'accès à plusieurs fonctions ou à des actions qui ne s'exécutent pas sur une fonction, utilisez des [rôles](#) (p. 41).

Octroi de l'accès de la couche à d'autres comptes

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.

```
$ aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id xaccount \
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text
e210fffdc-e901-43b0-824b-5fc0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal": {"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

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.

Pour accorder l'autorisation à tous les comptes de l'organisation, utilisez l'option `organization-id`. 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 --output text
b0cd9796-d4eb-4564-939f-de7fe0b42236 {"Sid":"engineering-org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

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

Nettoyage des stratégies basées sur les ressources

Pour afficher une stratégie basée sur les ressources d'une fonction, utilisez la commande `get-policy`.

```
$ aws lambda get-policy --function-name my-function --output text
{"Version":"2012-10-17","Id":"default","Statement": [{"Sid":"sns","Effect":"Allow","Principal": {"Service":"s3.amazonaws.com"}, "Action":"lambda:InvokeFunction", "Resource":"arn:aws:lambda:us-east-2:123456789012:function:my-function", "Condition":{"ArnLike": {"AWS:SourceArn":"arn:aws:sns:us-east-2:123456789012:lambda*"}}}]}      7c681fc9-b791-4e91-acdf-eb847fdcaa0f0
```

Pour les versions et les alias, ajoutez le numéro de version ou l'alias au nom de la fonction.

```
$ aws lambda get-policy --function-name my-function:PROD
```

Pour supprimer les autorisations de votre fonction, utilisez `remove-permission`.

```
$ aws lambda remove-permission --function-name example --statement-id sns
```

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

```
$ aws lambda get-layer-version-policy --layer-name my-layer --version-number 3 --output text
b0cd9796-d4eb-4564-939f-de7fe0b42236 {"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"}}}"
$ aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3 --statement-id engineering-org
```

Stratégies IAM basées sur l'identité pour AWS Lambda

Vous pouvez utiliser des stratégies basées sur l'identité dans AWS Identity and Access Management (IAM) afin d'accorder aux utilisateurs de votre compte l'accès à Lambda. Les stratégies basées sur l'identité peuvent s'appliquer directement aux utilisateurs, ou aux groupes et aux rôles associés à un utilisateur. Vous pouvez également accorder aux utilisateurs d'un autre compte l'autorisation d'assumer un rôle dans votre compte et d'accéder à vos ressources Lambda.

Lambda fournit des stratégies gérées qui accordent l'accès aux actions d'API de Lambda et, dans certains cas, l'accès à d'autres services utilisés pour développer et gérer des ressources Lambda. Lambda met à jour les stratégies gérées en fonction des besoins, afin de s'assurer que vos utilisateurs ont accès aux nouvelles fonctions lorsqu'elles sont publiées.

- **AWSLambdaFullAccess** – Accorde un accès complet aux actions et à d'autres services AWS Lambda utilisés pour développer et gérer les ressources Lambda.
- **AWSLambdaReadOnlyAccess** : accorde l'accès en lecture seule aux ressources AWS Lambda.
- **AWSLambdaRole** – Accorde les autorisations requises pour appeler les fonctions Lambda.

Les stratégies gérées accordent l'autorisation aux actions d'API sans pour autant restreindre les fonctions ou les couches qu'un utilisateur peut modifier. Pour bénéficier d'un contrôle plus précis, vous pouvez créer vos propres stratégies qui limitent la portée des autorisations d'un utilisateur.

Sections

- [Développement des fonctions \(p. 41\)](#)
- [Développement et utilisation des couches \(p. 44\)](#)
- [Rôles entre comptes \(p. 45\)](#)

Développement des fonctions

Voici un exemple de stratégie d'autorisations avec une portée limitée. Elle permet à un utilisateur de créer et de gérer des fonctions Lambda nommées avec un préfixe distinct (`intern-`) et configurées avec un rôle d'exécution distinct.

Example Stratégie de développement des fonctions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyPermissions",
```

```
        "Effect": "Allow",
        "Action": [
            "lambda:GetAccountSettings",
            "lambda>ListFunctions",
            "lambda>ListTags",
            "lambda:GetEventSourceMapping",
            "lambda>ListEventSourceMappings",
            "iam>ListRoles"
        ],
        "Resource": "*"
    },
    {
        "Sid": "DevelopFunctions",
        "Effect": "Allow",
        "NotAction": [
            "lambda>AddPermission",
            "lambda:PutFunctionConcurrency"
        ],
        "Resource": "arn:aws:lambda:*::function:intern-*"
    },
    {
        "Sid": "DevelopEventSourceMappings",
        "Effect": "Allow",
        "Action": [
            "lambda>DeleteEventSourceMapping",
            "lambda:UpdateEventSourceMapping",
            "lambda>CreateEventSourceMapping"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "lambda:FunctionArn": "arn:aws:lambda:*::function:intern-*"
            }
        }
    },
    {
        "Sid": "PassExecutionRole",
        "Effect": "Allow",
        "Action": [
            "iam>ListRolePolicies",
            "iam>ListAttachedRolePolicies",
            "iam>GetRole",
            "iam>GetRolePolicy",
            "iam>PassRole",
            "iam>SimulatePrincipalPolicy"
        ],
        "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
    },
    {
        "Sid": "ViewExecutionRolePolicies",
        "Effect": "Allow",
        "Action": [
            "iam>GetPolicy",
            "iam>GetPolicyVersion"
        ],
        "Resource": "arn:aws:iam::aws:policy/*"
    },
    {
        "Sid": "ViewLogs",
        "Effect": "Allow",
        "Action": [
            "logs:)"
        ],
        "Resource": "arn:aws:logs::*:log-group:/aws/lambda/intern-*"
    }
]
```

}

Les autorisations de la stratégie sont organisées en déclarations reposant sur les [ressources et conditions \(p. 45\)](#) qu'elles prennent en charge.

- **ReadOnlyPermissions** – La console Lambda utilise ces autorisations lorsque vous parcourez et affichez les fonctions. Les conditions ou modèles de ressources ne sont pas pris en charge.

```
"Action": [
    "lambda:GetAccountSettings",
    "lambda>ListFunctions",
    "lambda>ListTags",
    "lambda:GetEventSourceMapping",
    "lambda>ListEventSourceMappings",
    "iam>ListRoles"
],
"Resource": "*"
```

- **DevelopFunctions** – Utilisez n'importe quelle action Lambda qui s'exécute sur les fonctions ayant le préfixe `intern-`, à l'exception de `AddPermission` et de `PutFunctionConcurrency`. `AddPermission` modifie la [stratégie basée sur les ressources \(p. 36\)](#) de la fonction et peut avoir des conséquences en matière de sécurité. `PutFunctionConcurrency` réserve la capacité de mise à l'échelle pour une fonction et peut retirer la capacité d'autres fonctions.

```
"NotAction": [
    "lambda>AddPermission",
    "lambda>PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*::function:intern-*"
```

- **DevelopEventSourceMappings** – Gérez les mappages de source d'événement sur les fonctions qui sont préfixées avec `intern-`. Ces actions s'exécutent sur des mappages de source d'événement, mais vous pouvez les restreindre par fonction à l'aide d'une condition.

```
"Action": [
    "lambda>DeleteEventSourceMapping",
    "lambda>UpdateEventSourceMapping",
    "lambda>CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
    "StringLike": {
        "lambda>FunctionArn": "arn:aws:lambda::*:function:intern-*"
    }
}
```

- **PassExecutionRole** – Affichez et transmettez uniquement un rôle nommé `intern-lambda-execution-role`, qui doit être créé et géré par un utilisateur avec des autorisations IAM. `PassRole` est utilisé lorsque vous attribuez un rôle d'exécution pour une fonction.

```
"Action": [
    "iam>ListRolePolicies",
    "iam>ListAttachedRolePolicies",
    "iam>GetRole",
    "iam>GetRolePolicy",
    "iam>PassRole",
    "iam>SimulatePrincipalPolicy"
```

```
],
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- **ViewExecutionRolePolicies** – Affichez les stratégies gérées fournies par AWS qui sont attachées au rôle d'exécution. Vous pouvez ainsi afficher les autorisations de la fonction dans la console, mais vous ne pouvez pas afficher les stratégies qui ont été créées par d'autres utilisateurs dans le compte.

```
"Action": [
    "iam:GetPolicy",
    "iam:GetPolicyVersion"
],
"Resource": "arn:aws:iam::aws:policy/*"
```

- **ViewLogs** – Utilisez CloudWatch Logs pour afficher les journaux relatifs aux fonctions préfixées avec `intern-`.

```
"Action": [
    "logs:)"
],
"Resource": "arn:aws:logs:***:log-group:/aws/lambda/intern-*"
```

Cette stratégie permet à un utilisateur de démarrer avec Lambda, sans nuire aux ressources des autres utilisateurs. Elle ne permet pas à un utilisateur de configurer une fonction qui doit être déclenchée par d'autres services AWS ou appeler ces derniers ; pour cela, des autorisations IAM plus étendues sont nécessaires. Elle n'inclut pas d'autorisation d'accès aux services qui ne prennent pas en charge les stratégies de portée limitée, telles que CloudWatch et X-Ray. Pour ces services, utilisez les stratégies en lecture seule afin d'accorder à l'utilisateur l'accès aux métriques et aux données de suivi.

Lorsque vous configurez des déclencheurs pour votre fonction, vous devez être autorisé à utiliser le service AWS qui appelle votre fonction. Par exemple, pour configurer un déclencheur Amazon S3, vous avez besoin d'une autorisation d'accès aux actions Amazon S3 afin de gérer les notifications de compartiment. Plusieurs de ces autorisations sont incluses dans la stratégie gérée `AWSLambdaFullAccess`. Vous trouverez des exemples de stratégies dans le [référentiel GitHub](#) de ce guide.

Développement et utilisation des couches

La stratégie suivante accorde à un utilisateur l'autorisation de créer des couches et de les utiliser avec des fonctions. Les modèles de ressources permettent à l'utilisateur de travailler dans n'importe quelle région AWS, avec n'importe quelle version de couche, à condition que le nom de la couche commence par `test-`.

Example stratégie de développement des couches

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublishLayers",
            "Effect": "Allow",
            "Action": [
                "lambda:PublishLayerVersion"
            ],
            "Resource": "arn:aws:lambda:***:layer:test-*"
        },
        {
            "Sid": "ManageLayerVersions",
            "Effect": "Allow",
```

```
        "Action": [
            "lambda:GetLayerVersion",
            "lambda>DeleteLayerVersion"
        ],
        "Resource": "arn:aws:lambda:*:*:layer:test-*;*"
    }
}
```

Vous pouvez également appliquer l'utilisation des couches durant la création et la configuration de la fonction avec la condition `lambda:Layer`. Par exemple, vous pouvez empêcher les utilisateurs d'utiliser les couches publiées par d'autres comptes. La stratégie ci-dessous ajoute une condition aux actions `CreateFunction` et `UpdateFunctionConfiguration` afin d'exiger que toutes les couches spécifiées proviennent du compte 123456789012.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureFunctions",
            "Effect": "Allow",
            "Action": [
                "lambda>CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "lambda:Layer": [
                        "arn:aws:lambda:*:123456789012:layer:*;*"
                    ]
                }
            }
        }
    ]
}
```

Pour que la condition s'applique, vérifiez qu'aucune autre déclaration n'accorde d'autorisation utilisateur à ces actions.

Rôles entre comptes

Vous pouvez appliquer l'une des stratégies et des déclarations précédentes à un rôle, que vous pouvez ensuite partager avec un autre compte pour lui attribuer l'accès à vos ressources Lambda. Contrairement à un utilisateur IAM, un rôle ne dispose pas d'informations d'identification pour l'authentification. En revanche, il dispose d'une stratégie d'approbation qui spécifie qui peut endosser le rôle et utiliser ses autorisations.

Vous pouvez utiliser les rôles entre comptes pour donner aux comptes que vous approuvez l'accès aux actions et aux ressources Lambda. Si vous souhaitez simplement accorder l'autorisation d'appeler une fonction ou d'utiliser une couche, utilisez plutôt des [stratégies basées sur les ressources](#) (p. 36).

Pour plus d'informations, consultez [Rôles IAM](#) dans le IAM Guide de l'utilisateur.

Ressources et conditions pour les actions Lambda

Vous pouvez restreindre la portée des autorisations d'un utilisateur en spécifiant des ressources et des conditions dans une stratégie IAM. Chaque action d'API prend en charge une combinaison de ressources et types d'état qui varie en fonction du comportement de l'action.

Chaque déclaration de stratégie IAM accorde une autorisation pour une action effectuée sur une ressource. Lorsque l'action n'agit pas sur une ressource nommée, ou lorsque vous accordez l'autorisation d'effectuer l'action sur toutes les ressources, la valeur de la ressource de la stratégie est un caractère générique (*). Pour la plupart des actions d'API, vous pouvez limiter les ressources qu'un utilisateur peut modifier en spécifiant l'ARN (Amazon Resource Name) d'une ressource ou un modèle ARN qui correspond à plusieurs ressources.

Pour limiter les autorisations par ressource, spécifiez la ressource par son ARN.

Format ARN des ressources Lambda

- Fonction – arn:aws:lambda:**us-west-2:123456789012:function:my-function**
- Version de la fonction – arn:aws:lambda:**us-west-2:123456789012:function:my-function:1**
- Alias de la fonction – arn:aws:lambda:**us-west-2:123456789012:function:my-function:TEST**
- Mappage de source d'événement – arn:aws:lambda:**us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47**
- Couche – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer**
- Version de la couche – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer:1**

Par exemple, la stratégie ci-dessous autorise un utilisateur du compte 123456789012 à appeler une fonction nommée `my-function` dans la région USA Ouest (Oregon).

Example appeler une stratégie de fonction

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Invoke",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"  
        }  
    ]  
}
```

Il s'agit d'un cas particulier, où l'identificateur de l'action (`lambda:InvokeFunction`) diffère de l'opération d'API ([Invoke \(p. 565\)](#)). Pour les autres actions, l'identificateur de l'action est le nom de l'opération avec le préfixe `lambda:`.

Les conditions sont facultatives dans la stratégie, et appliquent une logique supplémentaire pour déterminer si une action est autorisée. Outre les [conditions courantes](#) prises en charge par toutes les actions, Lambda définit les types de condition que vous pouvez utiliser pour restreindre les valeurs des paramètres supplémentaires sur certaines actions.

Par exemple, la condition `lambda:Principal` permet de restreindre le service ou le compte auquel un utilisateur peut accorder l'accès selon la stratégie basée sur les ressources d'une fonction. La stratégie suivante permet à un utilisateur d'accorder une autorisation à des rubriques SNS d'appeler une fonction nommée `test`.

Example gestion des autorisations d'une stratégie de fonction

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "sns",  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:sns:us-west-2:123456789012:test",  
            "Condition": {  
                "lambda:Principal": "arn:aws:sns:us-west-2:123456789012"  
            }  
        }  
    ]  
}
```

```

"Statement": [
    {
        "Sid": "ManageFunctionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission",
            "lambda:RemovePermission"
        ],
        "Resource": "arn:aws:lambda:us-west-2:123456789012:function:test:*",
        "Condition": {
            "StringEquals": {
                "lambda:Principal": "sns.amazonaws.com"
            }
        }
    }
]
}

```

La condition nécessite que le mandataire soit Amazon SNS et pas un autre service ou compte. Le modèle de ressource exige que le nom de la fonction soit `test` et inclue un numéro de version ou un alias. Par exemple, `test:v1`.

Pour plus d'informations sur les ressources et les conditions pour Lambda et d'autres services AWS, consultez [Actions, ressources et clés de condition](#) dans le Guide de l'utilisateur d'IAM.

Sections

- [Fonctions \(p. 47\)](#)
- [Mappages de source d'événement \(p. 48\)](#)
- [Couches \(p. 49\)](#)

Fonctions

Les actions sur une fonction peuvent être limitées à une fonction spécifique selon l'ARN de la fonction, de la version ou de l'alias, tel que décrit dans le tableau suivant. Les actions qui ne prennent pas en charge les restrictions de ressources peuvent être accordées uniquement pour toutes les ressources (*).

Fonctions

| Action | Ressource | Condition |
|---|--|-------------------------------|
| AddPermission (p. 490) | Fonction | <code>lambda:Principal</code> |
| RemovePermission (p. 620) | Version de la fonction Alias de la fonction | |
| Invoke (p. 565) Autorisation : <code>lambda:InvokeFunction</code> | Fonction Version de la fonction Alias de la fonction | Aucune |
| CreateFunction (p. 504) UpdateFunctionConfiguration (p. 643) | Fonction | <code>lambda:Layer</code> |
| CreateAlias (p. 494) DeleteAlias (p. 513) | Fonction | Aucune |

| Action | Ressource | Condition |
|---|-----------|-----------|
| DeleteFunction (p. 519) DeleteFunctionConcurrency (p. 521) GetAlias (p. 531) GetFunction (p. 538) GetFunctionConfiguration (p. 543) GetPolicy (p. 560) ListAliases (p. 572) ListVersionsByFunction (p. 594) PublishVersion (p. 601) PutFunctionConcurrency (p. 608) UpdateAlias (p. 626) UpdateFunctionCode (p. 636) | | |
| GetAccountSettings (p. 529) ListFunctions (p. 581) ListTags (p. 592) TagResource (p. 622) UntagResource (p. 624) | * | Aucune |

Mappages de source d'événement

Pour les mappages de source d'événement, la suppression et la mise à jour des autorisations peuvent se limiter à une source d'événement spécifique. La condition `lambda:FunctionArn` vous permet de limiter les fonctions qu'un utilisateur peut configurer pour appeler une source d'événement.

Pour ces actions, la ressource est le mappage de source d'événement. Lambda fournit une condition qui vous permet de restreindre les autorisations selon la fonction appelée par le mappage de source d'événement.

Mappages de source d'événement

| Action | Ressource | Condition |
|---|-------------------------------|---------------------------------|
| DeleteEventSourceMapping (p. 515) | Mappage de source d'événement | <code>lambda:FunctionArn</code> |
| UpdateEventSourceMapping (p. 630) | | |
| CreateEventSourceMapping (p. 498) | * | <code>lambda:FunctionArn</code> |
| GetEventSourceMapping (p. 534) | * | Aucune |
| ListEventSourceMappings (p. 575) | | |

Couches

Les actions sur les couches vous permettent de limiter les couches qu'un utilisateur peut gérer ou utiliser avec une fonction. Les actions liées à l'utilisation et aux autorisations relatives aux couches, agissent sur les versions de couche, alors que `PublishLayerVersion` agit sur les noms de couche. Vous pouvez utiliser des caractères génériques pour restreindre les couches qu'un utilisateur peut utiliser par leur nom.

Couches

| Action | Ressource | Condition |
|---|----------------------|-----------|
| AddLayerVersionPermission (p. 487) | Version de la couche | Aucune |
| RemoveLayerVersionPermission (p. 618) | | |
| GetLayerVersion (p. 552) | | |
| GetLayerVersionPolicy (p. 558) | | |
| DeleteLayerVersion (p. 525) | | |
| PublishLayerVersion (p. 597) | Couche | Aucune |
| ListLayers (p. 584) | * | Aucune |
| ListLayerVersions (p. 586) | | |

Utilisation des limites d'autorisations pour les applications AWS Lambda

Lorsque vous [créez une application \(p. 139\)](#) dans la console AWS Lambda, Lambda applique une limite d'autorisations aux rôles IAM de l'application. La limite d'autorisations restreint la portée du [Rôle d'exécution \(p. 33\)](#) que le modèle de l'application crée pour chacune de ses fonctions, ainsi que les rôles que vous ajoutez au modèle. La limite d'autorisations empêche les utilisateurs disposant d'un accès en écriture au référentiel Git de l'application d'escalader les autorisations de l'application au-delà de la portée de ses propres ressources.

Les modèles d'application de la console Lambda incluent une propriété globale qui applique une limite d'autorisations à toutes les fonctions qu'ils créent.

```
Globals:
  Function:
    PermissionsBoundary: !Sub 'arn:${AWS::Partition}:iam::${AWS::AccountId}:policy/${AppId}-${AWS::Region}-PermissionsBoundary'
```

La limite restreint les autorisations des rôles des fonctions. Vous pouvez ajouter une autorisation au rôle d'exécution d'une fonction dans le modèle, mais cette autorisation n'est effective que si elle est également autorisée par la limite d'autorisations. Le rôle assumé par AWS CloudFormation pour déployer l'application applique l'utilisation de la limite d'autorisations. Ce rôle a uniquement l'autorisation de créer et de transmettre des rôles auxquels la limite d'autorisations de l'application est attachée.

Par défaut, la limite d'autorisations d'une application permet aux fonctions d'effectuer des actions sur les ressources de l'application. Par exemple, si l'application inclut une table Amazon DynamoDB, la limite permet d'accéder à toute action d'API qui peut être restreinte pour fonctionner sur des tables spécifiques avec des autorisations de niveau ressource. Vous pouvez utiliser des actions qui ne prennent pas en

charge les autorisations de niveau ressource uniquement si elles sont expressément autorisées dans la limite. Celles-ci comprennent les actions d'API Amazon CloudWatch Logs et AWS X-Ray pour la journalisation et le suivi.

Example limite d'autorisations

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "*"  
            ],  
            "Resource": [  
                "arn:aws:lambda:us-east-2:123456789012:function:my-app-getAllItemsFunction-*",  
                "arn:aws:lambda:us-east-2:123456789012:function:my-app getByIdFunction-*",  
                "arn:aws:lambda:us-east-2:123456789012:function:my-app-putItemFunction-*",  
                "arn:aws:dynamodb:us-east-1:123456789012:table/my-app-SampleTable-*"  
            ],  
            "Effect": "Allow",  
            "Sid": "StackResources"  
        },  
        {  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs>DescribeLogGroups",  
                "logs>PutLogEvents",  
                "xray:Put*"  
            ],  
            "Resource": "*",  
            "Effect": "Allow",  
            "Sid": "StaticPermissions"  
        },  
        ...  
    ]  
}
```

Pour accéder à d'autres ressources ou actions d'API, vous ou un administrateur devez étendre la limite d'autorisations pour inclure ces ressources. Vous devrez peut-être également mettre à jour le rôle d'exécution ou le rôle de déploiement d'une application pour autoriser l'utilisation d'actions supplémentaires.

- Limite d'autorisations – Étendez la limite d'autorisations de l'application lorsque vous ajoutez des ressources à votre application, ou que le rôle d'exécution doit avoir accès à plus d'actions. Dans IAM, ajoutez des ressources à la limite pour permettre l'utilisation d'actions d'API qui prennent en charge les autorisations de niveau ressource sur le type de cette ressource. Pour les actions qui ne prennent pas en charge les autorisations de niveau ressource, ajoutez-les dans une instruction étendue à aucune ressource.
- Rôle d'exécution– Étendez le rôle d'exécution d'une fonction lorsque des actions supplémentaires doivent être utilisées. Dans le modèle d'application, ajoutez des stratégies au rôle d'exécution. L'intersection des autorisations dans le rôle de limite et d'exécution est accordée à la fonction.
- Rôle de déploiement– Étendez le rôle de déploiement de l'application lorsqu'elle a besoin d'autorisations supplémentaires pour créer ou configurer des ressources. Dans IAM, ajoutez des stratégies au rôle de déploiement de l'application. Le rôle de déploiement nécessite les mêmes autorisations utilisateur que celles dont vous avez besoin pour déployer ou mettre à jour une application dans AWS CloudFormation.

Pour accéder à un didacticiel décrivant l'ajout de ressources à une application et l'extension de ses autorisations, consultez [??? \(p. 139\)](#).

Pour plus d'informations, consultez [Limites d'autorisations pour les entités IAM](#) dans le IAM Guide de l'utilisateur.

Gestion 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 \(p. 53\)](#) incluent la description, le rôle et le runtime 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 \(p. 55\)](#) sont toujours chiffrées au repos et peuvent également être chiffrées côté client. 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 \(p. 70\)](#) sont des ressources secondaires que vous pouvez créer pour gérer le déploiement et l'appel des fonctions. Publiez les [versions \(p. 70\)](#) 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 \(p. 72\)](#) 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 \(p. 76\)](#) 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 \(p. 81\)](#). La connexion de votre fonction à un VPC vous permet d'accéder aux ressources d'un sous-réseau privé, telles que les bases de données relationnelles et les caches. Vous pouvez également [créer un proxy de base de données \(p. 84\)](#) pour les instances de base de données MySQL et Aurora. Un proxy de base de données permet à une fonction d'atteindre des niveaux de simultanéité élevés sans épuiser les connexions de base de données.

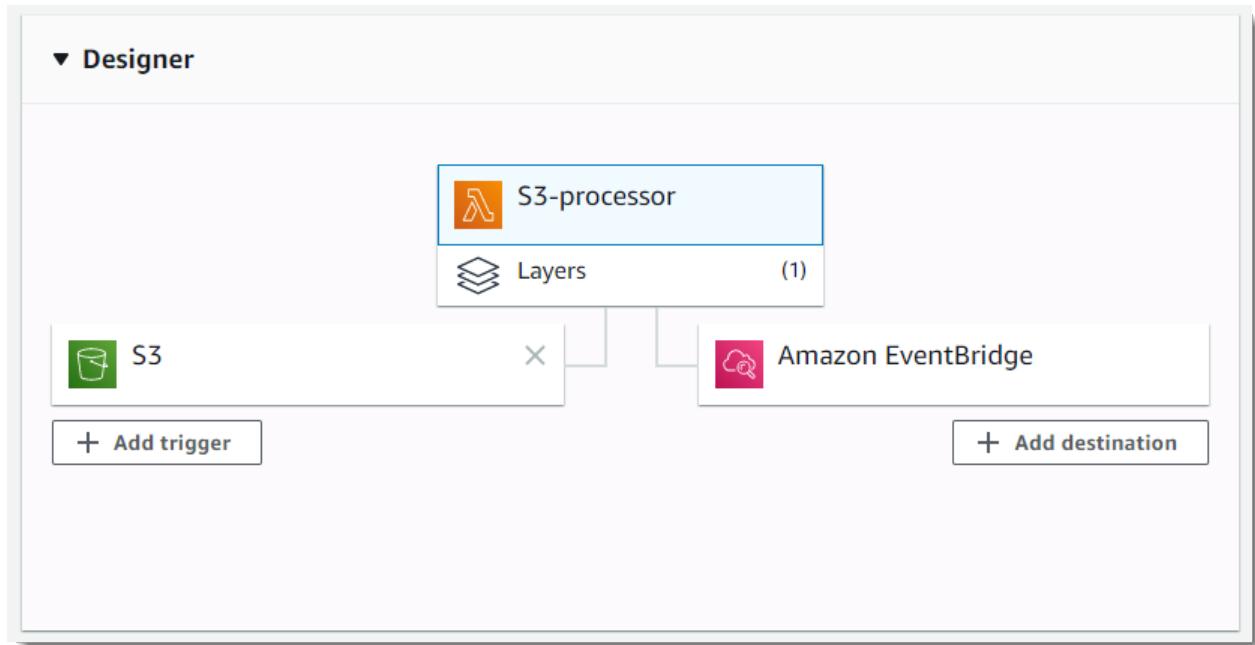
Rubriques

- [Configuration des fonctions dans la console AWS Lambda \(p. 53\)](#)
- [Utilisation des variables d'environnement AWS Lambda \(p. 55\)](#)
- [Gestion de la simultanéité pour une fonction Lambda \(p. 61\)](#)
- [Versions de fonction AWS Lambda \(p. 70\)](#)
- [Alias de fonctions AWS Lambda \(p. 72\)](#)
- [Couches AWS Lambda \(p. 76\)](#)
- [Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC \(p. 81\)](#)
- [Configuration de l'accès à une base de données pour une fonction Lambda \(p. 84\)](#)
- [Balisage des fonctions Lambda \(p. 87\)](#)

Configuration des fonctions dans la console AWS Lambda

Vous pouvez utiliser la console Lambda pour configurer les paramètres de fonction, ajouter des déclencheurs et des destinations, et mettre à jour et tester votre code.

Pour gérer une fonction, ouvrez la [page Fonctions](#) de la console Lambda et choisissez une fonction. Le concepteur de fonctions se trouve en haut de la page de configuration.



Le concepteur présente une vue d'ensemble de votre fonction et de ses ressources en amont et en aval. Vous pouvez l'utiliser pour configurer des déclencheurs, des couches et des destinations.

- Déclencheurs – Les déclencheurs sont des services et des ressources que vous avez configurés pour appeler votre fonction. Choisissez Ajouter un déclencheur pour créer un[mappage de source d'événement Lambda \(p. 101\)](#) ou pour configurer un déclencheur dans un autre service auquel la console Lambda s'intègre. Pour plus d'informations sur l'ensemble des services, consultez [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).
- Couches – Choisissez le nœud Layers (Couches) pour ajouter des [couches \(p. 76\)](#) à votre application. Une couche est une archive ZIP qui contient des bibliothèques, un environnement d'exécution personnalisé ou d'autres dépendances.
- Destinations– Ajoutez une destination à votre fonction pour envoyer des détails sur les résultats de l'appel à un autre service. Vous pouvez envoyer des enregistrements d'appel lorsque votre fonction est appelée de manière [asynchrone \(p. 93\)](#), ou via un [mappage de source d'événement \(p. 101\)](#) qui lit à partir d'un flux.

Lorsque le nœud de fonction est sélectionné dans le concepteur, vous pouvez modifier les paramètres suivants.

Paramètres des fonctions

- Code– Code et dépendances de votre fonction. Pour les langages de script, vous pouvez modifier votre code de fonction dans l'[éditeur \(p. 6\)](#) intégré. Pour ajouter des bibliothèques, ou pour les langages que

L'éditeur ne prend pas en charge, chargez un [package de déploiement \(p. 21\)](#). Si votre package de déploiement est supérieur à 50 Mo, choisissez Charger un fichier à partir d'Amazon S3.

- Runtime (Exécution) – [Runtime Lambda \(p. 122\)](#) qui exécute votre fonction.
- Handler (Gestionnaire) – Méthode que le runtime exécute lorsque votre fonction est appelée, par exemple `index.handler`. La première valeur est le nom du fichier ou module. La deuxième valeur est le nom de la méthode.
- Environment variables (Variables d'environnement) – Paires clé-valeur que Lambda définit dans l'environnement d'exécution. [Utilisez des variables d'environnement \(p. 55\)](#) pour étendre la configuration de votre fonction en dehors du code.
- Tags (Balises) – Paires clé-valeur que Lambda attache à votre ressource de fonction. [Utilisez des balises \(p. 87\)](#) pour organiser les fonctions Lambda en groupes pour les rapports de coût et le filtrage dans la console Lambda.

Les balises s'appliquent à la fonction dans son intégralité, y compris à l'ensemble des versions et des alias.

- Execution role (Rôle d'exécution) – [Rôle IAM \(p. 33\)](#) que AWS Lambda endosse lors de l'exécution de votre fonction.
- Description – Description de la fonction.
- Memory (Mémoire) – Quantité de mémoire disponible pour la fonction pendant son exécution. Choisissez un volume compris [entre 128 Mo et 3,008 MB \(p. 30\)](#) par incrément de 64 Mo.

Lambda alloue la puissance d'UC de façon linéaire en fonction de la quantité de mémoire configurée. À 1 792 Mo, une fonction possède l'équivalent d'1 vCPU entier (un vCPU-seconde de crédits par seconde).

- Timeout (Expiration) – Temps autorisé par Lambda pour l'exécution d'une fonction avant de l'arrêter. La durée par défaut est 3 secondes. La valeur maximale autorisée est 900 secondes.
- Virtual private cloud (VPC) – Si votre fonction a besoin d'un accès réseau à des ressources qui ne sont pas disponibles sur Internet, [configurez-la pour qu'elle se connecte à un VPC \(p. 81\)](#).
- Proxies de base de données – [Créez un proxy de base de données \(p. 84\)](#) pour les fonctions qui utilisent une instance de base de données ou un cluster Amazon RDS.
- Active tracing (Traçage actif) – Exemples de demandes entrantes et [suivi des exemples de demandes avec AWS X-Ray \(p. 296\)](#).
- Concurrency (Simultanéité) – [Réservez de la simultanéité pour une fonction \(p. 61\)](#) afin de définir le nombre maximal d'exécutions simultanées pour une fonction. Allouez de la simultanéité pour garantir qu'une fonction peut évoluer sans fluctuation de latence.

La simultanéité réservée s'applique à la fonction dans son intégralité, y compris à l'ensemble des versions et des alias.

- Asynchronous invocation (Appel asynchrone) – [Configurez le comportement de gestion des erreurs \(p. 93\)](#) pour réduire le nombre de nouvelles tentatives effectuées par Lambda ou la durée pendant laquelle les événements non traités restent en file d'attente avant que Lambda les supprime. [Configurez une file d'attente de lettres mortes \(p. 99\)](#) pour conserver les événements supprimés.

Vous pouvez configurer les paramètres de gestion des erreurs sur une fonction, une version ou un alias.

Sauf indication contraire dans la liste précédente, vous ne pouvez modifier les paramètres d'une fonction que sur sa version non publiée. Lorsque vous publiez une version, le code et la plupart des paramètres sont verrouillés afin de garantir une expérience cohérente pour les utilisateurs de cette version. Utilisez des alias (p. 72) pour propager les modifications de la configuration de manière contrôlée.

Pour configurer les fonctions avec l'API Lambda, utilisez les actions suivantes :

- [UpdateFunctionCode \(p. 636\)](#) – Mettre à jour le code de la fonction.
- [UpdateFunctionConfiguration \(p. 643\)](#) – Mettre à jour les paramètres spécifiques à la version.

- [TagResource \(p. 622\)](#) – Baliser une fonction.
- [AddPermission \(p. 490\)](#) – Modifier la [stratégie basée sur les ressources \(p. 36\)](#) d'une fonction, d'une version ou d'un alias.
- [PutFunctionConcurrency \(p. 608\)](#) – Configurer la simultanéité réservée d'une fonction.
- [PublishVersion \(p. 601\)](#) – Créer une version immuable avec le code et la configuration actuels.
- [CreateAlias \(p. 494\)](#) – Créer des alias pour les versions de la fonction.
- [PutFunctionEventInvokeConfig](#) – Configurez la gestion des erreurs pour l'appel asynchrone.

Par exemple, pour mettre à jour le paramètre de mémoire d'une fonction avec l'AWS CLI, utilisez la commande `update-function-configuration`.

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

Pour connaître les bonnes pratiques en matière de configuration des fonctions, consultez [Configuration de fonctions \(p. 152\)](#).

Utilisation des variables d'environnement AWS Lambda

Vous pouvez utiliser des variables d'environnement pour stocker les secrets en toute sécurité et ajuster le comportement de votre fonction sans mettre à jour le code. Une variable d'environnement est une paire de chaînes qui sont stockées dans la configuration spécifique à la version d'une fonction. L'exécution de Lambda met les variables d'environnement à la disposition de votre code et définit les variables d'environnement supplémentaires qui contiennent des informations sur la fonction et la demande d'appel.

Vous définissez des variables d'environnement sur la version non publiée de votre fonction en spécifiant une clé et une valeur. Lorsque vous publiez une version, les variables d'environnement sont verrouillées pour cette version ainsi que d'autres [configurations spécifiques à la version \(p. 53\)](#).

Pour définir des variables d'environnement dans la console Lambda

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Variables d'environnement, choisissez Modifier.
4. Choisissez Ajouter une variable d'environnement.
5. Entrez une clé et une valeur.

Prérequis

- Les clés commencent par une lettre et comportent au moins deux caractères.
 - Les clés contiennent uniquement des lettres, des chiffres et le caractère de soulignement (_).
 - Les clés ne sont pas [réservées par Lambda \(p. 57\)](#).
 - La taille totale de toutes les variables d'environnement ne dépasse pas 4 Ko.
6. Choisissez Enregistrer.

Utilisez des variables d'environnement pour transmettre des paramètres spécifiques à l'environnement à votre code. Vous pouvez par exemple avoir deux fonctions avec le même code mais une configuration différente. Une fonction se connecte à une base de données de test et l'autre à une base de données de production. Dans ce cas, vous utilisez des variables d'environnement pour indiquer à la fonction le nom

d'hôte, ainsi que d'autres détails de connexion pour la base de données. Vous pouvez également définir une variable d'environnement pour configurer votre environnement de test afin d'utiliser une journalisation ou un suivi plus détaillé.

| ENVIRONMENT | DEVELOPMENT | Remove |
|--------------|-----------------------------|--------|
| databaseHost | lambdadb | Remove |
| databaseName | rd1owwlydynnm5.cuovuayfg087 | Remove |
| Key | Value | Remove |

Pour récupérer les variables d'environnement dans le code de votre fonction, utilisez la méthode standard pour votre langage de programmation.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
region = os.environ['AWS_REGION']
```

Ruby

```
region = ENV["AWS_REGION"]
```

Java

```
String region = System.getenv("AWS_REGION");
```

Go

```
var region = os.Getenv("AWS_REGION")
```

C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

PowerShell

```
$region = $env:AWS_REGION
```

Lambda stocke les variables d'environnement en toute sécurité en les chiffrant au repos. Vous pouvez configurer Lambda afin d'utiliser une clé de chiffrement différente (p. 58), chiffrer les valeurs des variables d'environnement côté client ou définir des variables d'environnement dans un modèle AWS CloudFormation avec AWS Secrets Manager.

Sections

- [Variables d'environnement d'exécution \(p. 57\)](#)
- [Sécurisation des variables d'environnement \(p. 58\)](#)
- [Configuration de variables d'environnement avec l'API Lambda \(p. 60\)](#)
- [Exemples de code et de modèles \(p. 60\)](#)

Variables d'environnement d'exécution

Les [environnements d'exécution Lambda \(p. 122\)](#) définissent plusieurs variables d'environnement lors de l'initialisation. La plupart des variables d'environnement fournissent des informations sur la fonction ou l'exécution. Les clés de ces variables d'environnement sont réservées et ne peuvent pas être définies dans la configuration de la fonction.

Variables d'environnement réservées

- `_HANDLER` – Emplacement de gestionnaire configuré au niveau de la fonction.
- `AWS_REGION` – Région AWS où la fonction Lambda est exécutée.
- `AWS_EXECUTION_ENV` – [Identifiant d'exécution \(p. 122\)](#), doté du préfixe `AWS_Lambda_`—par exemple, `AWS_Lambda_java8`.
- `AWS_LAMBDA_FUNCTION_NAME` – Nom de la fonction.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE` – Quantité de mémoire disponible pour la fonction en Mo.
- `AWS_LAMBDA_FUNCTION_VERSION` – Version de la fonction en cours d'exécution.
- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME` – Nom du groupe Amazon CloudWatch Logs et du flux pour la fonction.
- `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` – Clés d'accès obtenues à partir du [rôle d'exécution \(p. 33\)](#) de la fonction.
- `AWS_LAMBDA_RUNTIME_API` – ([Environnement d'exécution personnalisé \(p. 126\)](#)) Hôte et port de l'[API d'exécution \(p. 128\)](#).
- `LAMBDA_TASK_ROOT` – Chemin vers le code de la fonction Lambda.
- `LAMBDA_RUNTIME_DIR` – Chemin vers les bibliothèques d'exécution.
- `TZ` – Fuseau horaire de l'environnement (UTC). L'environnement d'exécution utilise NTP pour synchroniser l'horloge système.

Les variables d'environnement supplémentaires suivantes ne sont pas réservées et peuvent être étendues dans la configuration de la fonction.

Variables d'environnement non réservées

- `LANG` – Paramètres régionaux de l'environnement d'exécution (`en_US.UTF-8`).
- `PATH` – Chemin d'exécution (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).
- `LD_LIBRARY_PATH` – Chemin de la bibliothèque système (`/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- `NODE_PATH` – ([Node.js \(p. 312\)](#)) Chemin de la bibliothèque Node.js (`/opt/nodejs/node12/node_modules:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `PYTHONPATH` – ([Python 2.7, 3.6, 3.8 \(p. 329\)](#)) Chemin de la bibliothèque Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH` – ([Ruby \(p. 347\)](#)) Chemin de la bibliothèque Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).

- `_X_AMZN_TRACE_ID` – [En-tête de suivi X-Ray \(p. 296\)](#).
- `AWS_XRAY_CONTEXT_MISSING` – Pour le suivi X-Ray, Lambda définit cette valeur sur `LOG_ERROR` pour éviter de générer des erreurs d'exécution à partir du kit SDK X-Ray.
- `AWS_XRAY_DAEMON_ADDRESS` – Pour le suivi X-Ray, adresse IP et port du démon X-Ray

Les valeurs d'exemple affichées reflètent les derniers moteurs d'exécution. La présence de variables spécifiques ou leurs valeurs peuvent varier dans les anciens environnements d'exécution.

Sécurisation des variables d'environnement

Lambda chiffre les variables d'environnement à l'aide d'une clé que le service crée dans votre compte (une clé CMK gérée par AWS). Cette clé est utilisée gratuitement. Vous pouvez également choisir de fournir votre propre clé afin qu'elle soit utilisée par Lambda à la place de la clé par défaut.

Lorsque vous fournissez la clé, seuls les utilisateurs de votre compte ayant accès à la clé peuvent afficher ou gérer les variables d'environnement sur la fonction. Votre organisation peut également avoir des exigences internes ou externes pour gérer les clés utilisées pour le chiffrement et pour contrôler leur rotation.

Pour utiliser une clé CMK gérée par le client

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Variables d'environnement, choisissez Modifier.
4. Développez Configuration du chiffrement.
5. Choisissez Utiliser une clé principale client.
6. Choisissez votre clé CMK gérée par le client.
7. Choisissez Enregistrer.

Les clés CMK gérées par le client entraînent des [frais AWS KMS standard](#).

Aucune autorisation AWS KMS n'est requise pour votre utilisateur ou le rôle d'exécution de la fonction pour utiliser la clé de chiffrement par défaut. Vous devez disposer des autorisations appropriées pour pouvoir utiliser une clé CMK gérée par le client. Lambda exploite ces autorisations pour créer un octroi sur la clé. Cela permet à Lambda de l'utiliser pour le chiffrement.

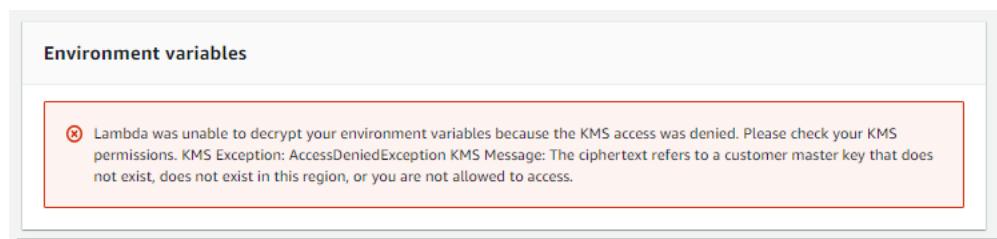
- `kms>ListAliases` – Permet d'afficher les clés dans la console Lambda.
- `kms>CreateGrant, kms:Encrypt` – Permet de configurer une clé CMK gérée par le client au niveau d'une fonction.
- `kms:Decrypt` – Permet d'afficher et de gérer les variables d'environnement chiffrées à l'aide d'une clé CMK gérée par le client.

Vous pouvez obtenir ces autorisations à partir de votre compte d'utilisateur ou à partir de la stratégie d'autorisation basée sur les ressources d'une clé. `ListAliases` est fourni par les [stratégies gérées pour Lambda \(p. 41\)](#). Les stratégies de clé accordent les autorisations restantes aux utilisateurs du groupe Key users (Utilisateurs de clés).

Les utilisateurs sans autorisation `Decrypt` peuvent gérer les fonctions, mais ils ne peuvent pas afficher les variables d'environnement ni les gérer dans la console Lambda. Pour empêcher un utilisateur d'afficher des variables d'environnement, ajoutez une instruction aux autorisations de cet utilisateur afin de refuser l'accès à la clé par défaut, à une clé gérée par le client ou à toutes les clés.

Example Stratégie IAM – Refuser l'accès en fonction de l'ARN de la clé

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Deny",  
            "Action": [  
                "kms:Decrypt"  
            ],  
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/3be10e2d-xmpl-4be4-  
bc9d-0405a71945cc"  
        }  
    ]  
}
```



Pour plus d'informations sur la gestion des autorisations des clés, consultez [Utilisation de stratégies de clé dans AWS KMS](#).

Vous pouvez également chiffrer les valeurs des variables d'environnement côté client avant de les envoyer à Lambda, et les déchiffrer dans le code de votre fonction. Les valeurs secrètes sont ainsi masquées dans la console Lambda et dans la sortie de l'API, même pour les utilisateurs autorisés à utiliser la clé. Dans votre code, vous récupérez la valeur chiffrée de l'environnement et la déchiffrez à l'aide de l'API AWS KMS.

Pour chiffrer des variables d'environnement côté client

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Variables d'environnement, choisissez Modifier.
4. Développez Configuration du chiffrement.
5. Choisissez Enable helpers for encryption in transit (Activer les assistants pour le chiffrement en transit).
6. Choisissez Chiffrer en regard d'une variable pour chiffrer sa valeur.
7. Choisissez Enregistrer.

Note

Lorsque vous utilisez les assistants de chiffrement de la console, votre fonction a besoin d'une autorisation pour appeler l'opération d'API `kms:Decrypt` dans son [rôle d'exécution](#) (p. 33).

Pour afficher un exemple de code pour la langue de votre fonction, choisissez Code en regard d'une variable d'environnement. L'exemple de code montre comment récupérer une variable d'environnement dans une fonction et déchiffrer sa valeur.

Une autre option consiste à stocker les mots de passe dans des secrets AWS Secrets Manager. Vous pouvez référencer le secret dans vos modèles AWS CloudFormation pour définir des mots de passe sur les

bases de données. Vous pouvez également définir la valeur d'une variable d'environnement sur la fonction Lambda. Pour obtenir un exemple, veuillez consulter la section suivante.

Configuration de variables d'environnement avec l'API Lambda

Pour gérer les variables d'environnement avec l'AWS CLI ou un kit SDK AWS, utilisez les opérations d'API suivantes.

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple suivant définit deux variables d'environnement sur une fonction nommée `my-function`.

```
$ aws lambda update-function-configuration --function-name my-function \
--environment "Variables={BUCKET=my-bucket,KEY=file.txt}"
```

Lorsque vous appliquez des variables d'environnement avec la commande `update-function-configuration`, l'ensemble du contenu de la structure `Variables` est remplacé. Pour conserver les variables d'environnement existantes lorsque vous en ajoutez une nouvelle, incluez toutes les valeurs existantes dans votre demande.

Pour obtenir la configuration actuelle, utilisez la commande `get-function-configuration`.

```
$ aws lambda get-function-configuration --function-name my-function
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs12.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "my-bucket",
            "KEY": "file.txt"
        }
    },
    "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",
    ...
}
```

Pour vous assurer que les valeurs ne changent pas entre la lecture et la mise à jour de la configuration, vous pouvez transmettre l'ID de révision de la sortie de `get-function-configuration` en tant que paramètre à `update-function-configuration`.

Pour configurer la clé de chiffrement d'une fonction, définissez l'option `KMSKeyARN`.

```
$ aws lambda update-function-configuration --function-name my-function \
--kms-key-arn arn:aws:kms:us-east-2:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Exemples de code et de modèles

Des exemples d'applications dans le référentiel GitHub de ce manuel illustrent l'utilisation de variables d'environnement dans le code et les modèles AWS CloudFormation de fonctions.

Exemples d'applications

- [Fonction vide \(p. 301\)](#) – Créez une fonction et une rubrique Amazon SNS dans le même modèle. Transmettez le nom de la rubrique à la fonction dans une variable d'environnement. Lisez les variables d'environnement dans le code (plusieurs langues).
- [RDS MySQL](#) – Créez un VPC et une instance de base de données Amazon RDS dans un modèle, avec un mot de passe stocké dans Secrets Manager. Dans le modèle d'application, importez les détails de la base de données à partir de la pile VPC, lisez le mot de passe depuis Secrets Manager et transmettez l'ensemble de la configuration de connexion à la fonction dans les variables d'environnement.

Gestion de la simultanéité pour une fonction Lambda

La simultanéité est le nombre de demandes auxquelles votre fonction répond à un moment donné. Lorsque votre fonction est appelée, Lambda alloue une instance pour traiter l'événement. Lorsque le code de la fonction a fini de s'exécuter, il peut gérer une autre demande. Si la fonction est appelée à nouveau alors qu'une demande est toujours en cours de traitement, une autre instance est allouée, ce qui augmente la simultanéité de la fonction.

La simultanéité est soumise à une [limite \(p. 30\)](#) régionale partagée par toutes les fonctions figurant dans une région. Pour vous assurer qu'une fonction peut toujours atteindre un certain niveau de simultanéité, vous pouvez la configurer avec la [simultanéité réservée \(p. 62\)](#). Lorsqu'une fonction dispose de la simultanéité réservée, aucune autre fonction ne peut utiliser cette simultanéité. La simultanéité réservée limite également la simultanéité maximale pour la fonction et s'applique à la fonction dans son ensemble, y compris les versions et les alias.

Si Lambda alloue une instance de votre fonction, l'[environnement d'exécution \(p. 122\)](#) charge le code de votre fonction et exécute le code d'initialisation que vous définissez en dehors du gestionnaire. Si votre code et les dépendances sont volumineux, ou si vous créez des clients SDK pendant l'initialisation, ce processus peut prendre un certain temps. Au fur et à mesure que votre fonction [évolue \(p. 106\)](#), la partie des demandes servies par les nouvelles instances subissent une latence plus élevée.

Pour permettre à votre fonction d'évoluer sans fluctuations de latence, utilisez la [simultanéité provisionnée \(p. 64\)](#). En allouant la simultanéité provisionnée avant une augmentation des appels, vous pouvez vous assurer que toutes les demandes sont servies par des instances initialisées avec une latence très faible. Vous pouvez configurer la simultanéité provisionnée sur une version d'une fonction ou sur un alias.

Lambda s'intègre également à [Application Auto Scaling](#). Vous pouvez configurer Application Auto Scaling pour gérer la simultanéité provisionnée selon une planification ou en fonction de l'utilisation. Utilisez la mise à l'échelle planifiée pour augmenter la simultanéité provisionnée en prévision du trafic de pointe. Pour augmenter automatiquement la simultanéité provisionnée en fonction des besoins, utilisez [l'API Application Auto Scaling \(p. 67\)](#) pour enregistrer une cible et créer une stratégie de dimensionnement.

La simultanéité provisionnée compte dans les limites de simultanéité réservée et régionales d'une fonction. Si la simultanéité provisionnée sur les versions et les alias d'une fonction s'ajoute à la simultanéité réservée de la fonction, tous les appels s'exécutent sur la simultanéité provisionnée. Cette configuration a également pour effet de limiter la version non publiée de la fonction (`$_LATEST`), ce qui empêche son exécution.

Sections

- [Configuration de la simultanéité réservée \(p. 62\)](#)
- [Configuration de la simultanéité provisionnée \(p. 64\)](#)
- [Configuration de la simultanéité avec l'API Lambda \(p. 67\)](#)

Configuration de la simultanéité réservée

Pour gérer les paramètres de simultanéité réservée pour une fonction, utilisez la console Lambda.

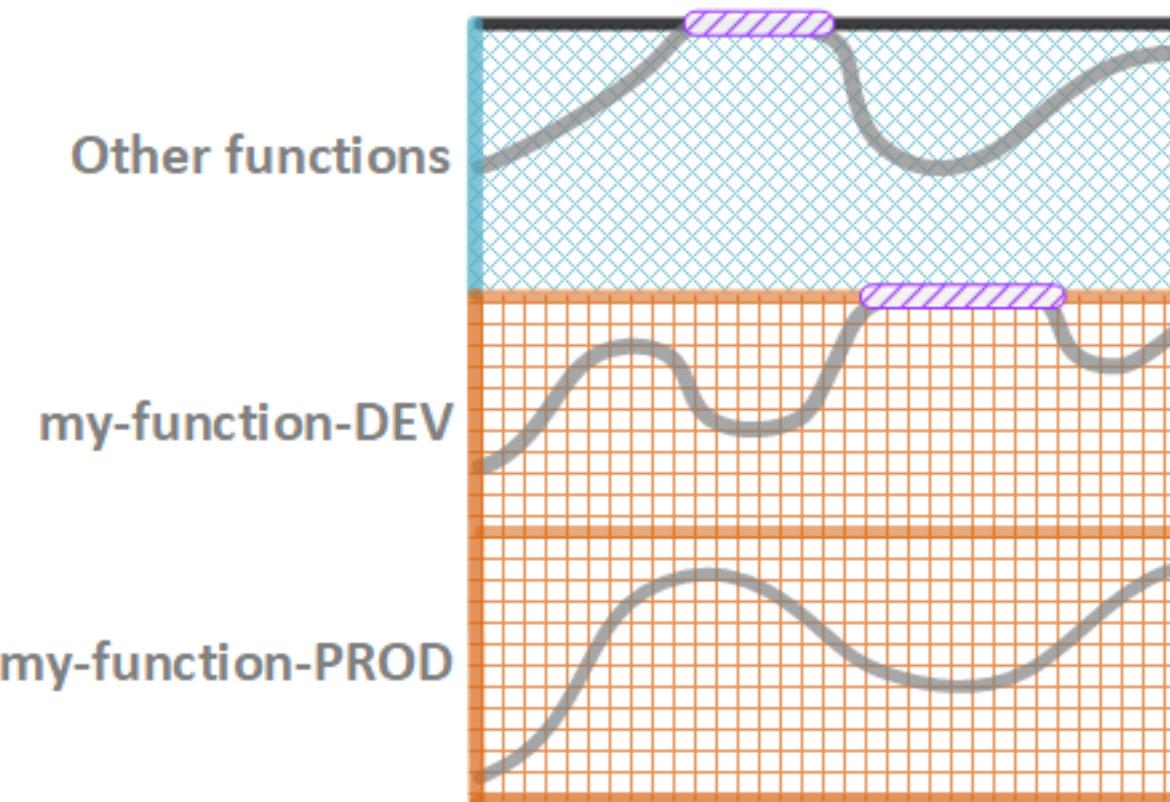
Réserver la simultanéité pour une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Concurrency (Simultanéité), choisissez Reserve concurrency (Réserver simultanéité).
4. Saisissez la quantité de simultanéité à réserver pour la fonction.
5. Choisissez Save.

Vous pouvez réserver jusqu'à la valeur affichée Unreserved account concurrency (Simultanéité de compte non réservée), moins 100 pour les fonctions qui n'ont pas de simultanéité réservée. Pour limiter une fonction, définissez la simultanéité réservée à zéro. Cela empêche tout événement d'être traité jusqu'à ce que vous supprimiez la limite.

L'exemple suivant montre deux fonctions avec des pools de simultanéité réservée, ainsi que le pool de simultanéité non réservée utilisé par d'autres fonctions. Des erreurs de limitation se produisent lorsque toute la simultanéité d'un pool est en cours d'utilisation.

Reserved Concurrency



Légende

- Simultanéité de la fonction
- Simultanéité réservée
- Simultanéité non réservée
- Limitation

Les effets de la réservation de la simultanéité sont les suivants.

- D'autres fonctions ne peuvent pas empêcher votre fonction d'évoluer – Toutes les fonctions de votre compte dans la même région sans simultanéité réservée partagent le groupe de simultanéité non réservée. Sans simultanéité réservée, d'autres fonctions peuvent utiliser toute la simultanéité disponible. Cela empêche l'évolution de votre fonction si nécessaire.
- Votre fonction ne peut pas évoluer sans contrôle – La simultanéité réservée limite également votre fonction pour utiliser la simultanéité à partir du groupe non réservé, plafonnant ainsi sa simultanéité

maximale. Vous pouvez réserver de la simultanéité pour empêcher votre fonction d'utiliser toute la simultanéité disponible dans la région, ou de surcharger les ressources en aval.

Le fait de définir la simultanéité par fonction peut avoir des conséquences sur le groupe de simultanéité disponible pour d'autres fonctions. Pour éviter les problèmes, limitez le nombre d'utilisateurs pouvant faire appel aux opérations d'API `DeleteFunctionConcurrency` et `PutFunctionConcurrency`.

Configuration de la simultanéité provisionnée

Pour gérer les paramètres de simultanéité provisionnée pour une version ou un alias, utilisez la console Lambda.

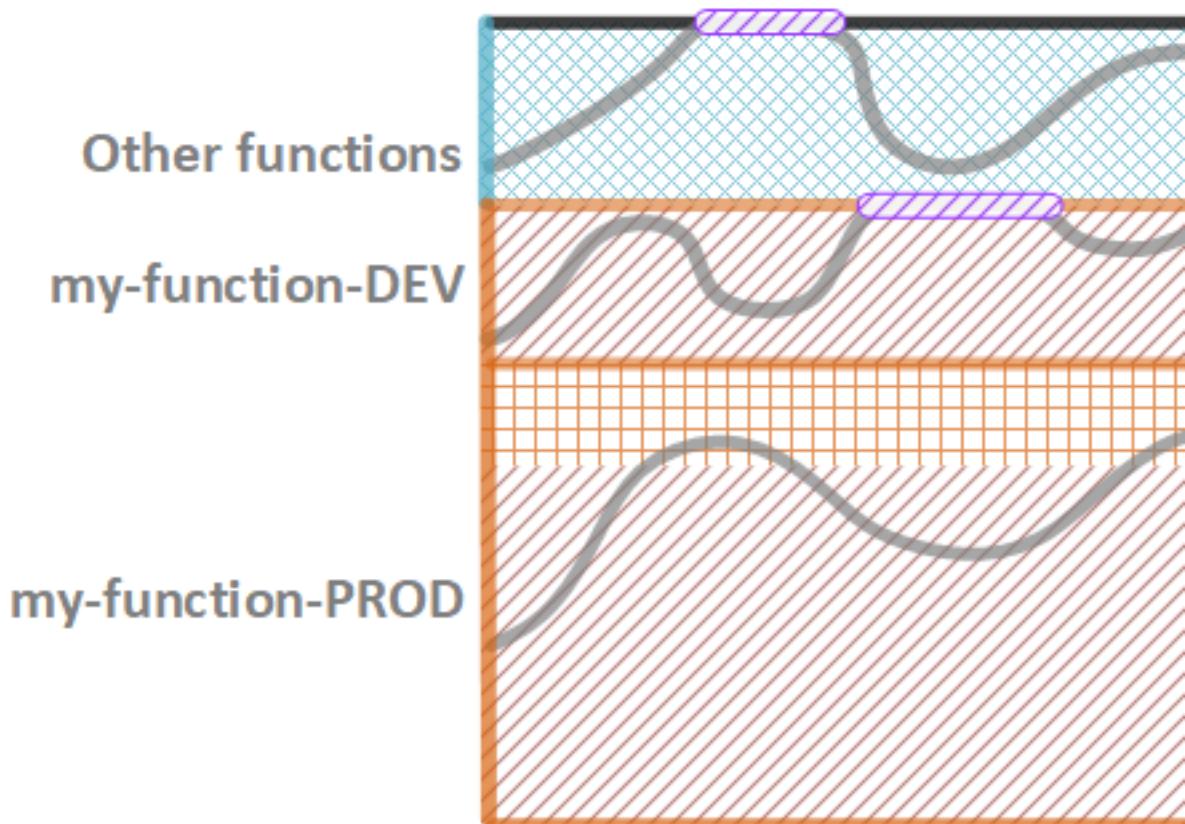
Pour réserver de la simultanéité pour un alias

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Provisioned concurrency configurations (Configurations de simultanéité provisionnée), choisissez Add (Ajouter).
4. Choisissez un alias ou une version.
5. Entrez le montant de la simultanéité provisionnée à allouer.
6. Choisissez Enregistrer.

Vous pouvez gérer la simultanéité provisionnée pour tous les alias et toutes les versions à partir de la page de configuration de la fonction. La liste des configurations de simultanéité provisionnée indique la progression de l'allocation de chaque configuration. Les paramètres de simultanéité provisionnée sont également disponibles sur la page de configuration de chaque version et de chaque alias.

Dans l'exemple suivant, les fonctions `my-function-DEV` et `my-function-PROD` sont configurées avec la simultanéité réservée et provisionnée. Pour `my-function-DEV`, le pool complet de la simultanéité réservée est également provisionné. Dans ce cas, tous les appels s'exécutent au niveau de la simultanéité provisionnée ou sont limités. Pour `my-function-PROD`, une partie du pool de simultanéité réservée correspond à la simultanéité standard. Lorsque l'ensemble de la simultanéité provisionnée est utilisée, la fonction exploite la simultanéité standard pour répondre aux demandes supplémentaires.

Provisioned Concurrency with Reserved Concurrency



Légende

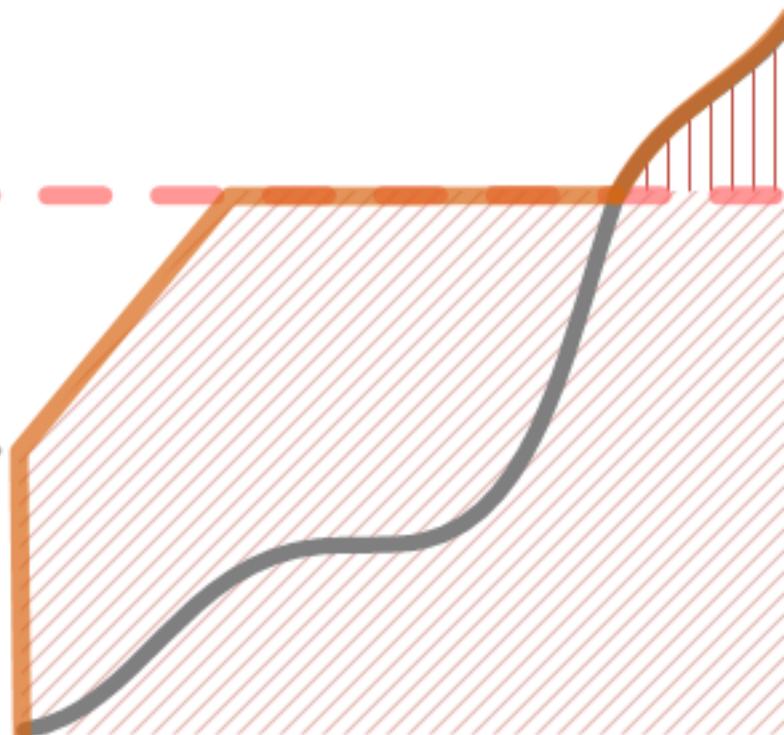
- Simultanéité de la fonction
- Simultanéité réservée
- Simultanéité provisionnée
- Simultanéité non réservée
- Limitation

La simultanéité allouée n'est pas mise en ligne immédiatement après la configuration. Lambda commence à allouer la simultanéité allouée après une phase de préparation d'une ou deux minutes. De la même manière que les fonctions sont mises à l'échelle en fonction de la charge (p. 106), jusqu'à 3 000 instances de la fonction peuvent être initialisées en même temps, selon la région. Après le lancement initial en mode rafale, les instances sont allouées à un taux constant de 500 par minute jusqu'à ce que la demande soit satisfaite. Lorsque vous demandez la simultanéité allouée pour plusieurs fonctions ou versions d'une fonction dans la même région, des limites de mise à l'échelle s'appliquent à toutes les demandes.

Function Scaling with Provisioned Concurrency

Requested
provisioned
concurrency

Burst limit



Légende

- Instances de la fonction
- Demandes ouvertes
- Simultanéité allouée
- Simultanéité standard

Le [code d'initialisation \(p. 19\)](#) de votre fonction s'exécute pendant l'allocation et à intervalle de quelques heures, car les instances de votre fonction qui s'exécutent sont recyclées. Vous pouvez consulter le temps d'initialisation dans les journaux et les [traces \(p. 296\)](#) après le traitement d'une demande par une instance. Toutefois, l'initialisation est facturée même si l'instance ne traite pas une demande. La simultanéité allouée s'exécute en continu et est facturée séparément des coûts d'initialisation et d'appel. Pour plus d'informations, consultez [Tarification AWS Lambda](#).

Chaque version d'une fonction ne peut avoir qu'une seule configuration de simultanéité provisionnée. Cela peut être directement sur la version elle-même ou sur un alias qui pointe vers la version. Deux alias ne peuvent pas allouer la simultanéité provisionnée pour la même version. En outre, vous ne pouvez pas allouer la simultanéité provisionnée sur un alias qui pointe vers la version non publiée (`$LATEST`).

Lorsque vous modifiez la version vers laquelle un alias pointe, la simultanéité provisionnée est désallouée de l'ancienne version, puis allouée à la nouvelle version. Vous pouvez ajouter une configuration de routage à un alias doté d'une simultanéité provisionnée. Toutefois, vous ne pouvez pas gérer les paramètres de simultanéité provisionnée sur l'alias lorsque la configuration de routage est en place.

Lambda émet les métriques suivantes pour la simultanéité provisionnée :

Métriques de simultanéité provisionnée

- `ProvisionedConcurrentExecutions`
- `ProvisionedConcurrencyInvocations`
- `ProvisionedConcurrencySpilloverInvocations`
- `ProvisionedConcurrencyUtilization`

Pour plus d'informations, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

Configuration de la simultanéité avec l'API Lambda

Pour gérer les paramètres de simultanéité avec l'AWS CLI ou le kit SDK AWS, utilisez les opérations d'API suivantes.

- [PutFunctionConcurrency \(p. 608\)](#)
- [GetFunctionConcurrency](#)
- [DeleteFunctionConcurrency \(p. 521\)](#)
- [PutProvisionedConcurrencyConfig](#)
- [GetProvisionedConcurrencyConfig](#)
- [ListProvisionedConcurrencyConfigs](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [GetAccountSettings \(p. 529\)](#)
- (Application Auto Scaling) [RegisterScalableTarget](#)
- (Application Auto Scaling) [PutScalingPolicy](#)

Pour configurer la simultanéité réservée avec l'AWS CLI, utilisez la commande `put-function-concurrency`. La commande suivante réserve une simultanéité de 100 pour une fonction nommée `my-function` :

```
$ aws lambda put-function-concurrency --function-name my-function --reserved-concurrent-executions 100
{
    "ReservedConcurrentExecutions": 100
}
```

Pour allouer de la simultanéité provisionnée pour une fonction, utilisez `put-provisioned-concurrency-config`. La commande suivante alloue une simultanéité de 100 pour l'alias `BLUE` d'une fonction nommée `my-function` :

```
$ aws lambda put-provisioned-concurrency-config --function-name my-function \
--qualifier BLUE --provisioned-concurrent-executions 100
{
    "Requested ProvisionedConcurrentExecutions": 100,
    "Allocated ProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

Pour configurer Application Auto Scaling de sorte à gérer la simultanéité provisionnée, utilisez Application Auto Scaling pour configurer le [dimensionnement du suivi de la cible](#). Tout d'abord, enregistrez l'alias d'une fonction en tant que cible de dimensionnement. L'exemple suivant enregistre l'alias `BLUE` d'une fonction nommée `my-function` :

```
$ aws application-autoscaling register-scalable-target --service-namespace lambda \
--resource-id function:my-function:BLUE --min-capacity 1 --max-capacity 100 \
--scalable-dimension lambda:function:ProvisionedConcurrency
```

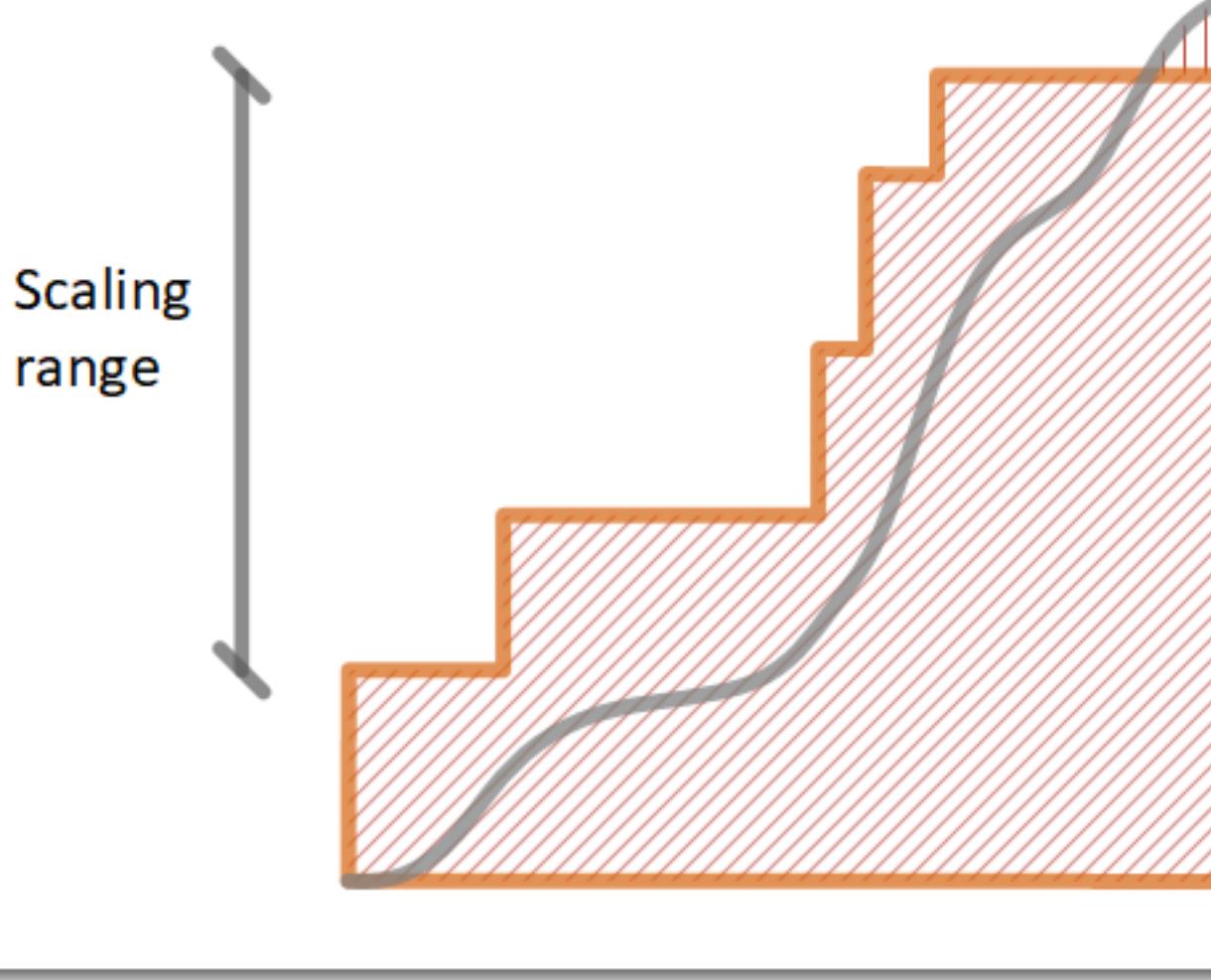
Ensuite, appliquez une stratégie de dimensionnement à la cible. L'exemple suivant configure Application Auto Scaling pour ajuster la configuration de simultanéité provisionnée pour un alias afin de maintenir l'utilisation proche de 70 %.

```
$ aws application-autoscaling put-scaling-policy --service-namespace lambda \
--scalable-dimension lambda:function:ProvisionedConcurrency --resource-id function:my-
function:BLUE \
--policy-name my-policy --policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration '{ "TargetValue": 
0.7, "PredefinedMetricSpecification": { "PredefinedMetricType": 
"LambdaProvisionedConcurrencyUtilization" }}'
{
    "PolicyARN": "arn:aws:autoscaling:us-east-2:123456789012:scalingPolicy:12266dbb-1524-
xmpl-a64e-9a0a34b996fa:resource/lambda/function:my-function:BLUE:policyName/my-policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
            "AlarmARN": "arn:aws:cloudwatch:us-east-2:123456789012:alarm:TargetTracking-
function:my-function:BLUE-AlarmHigh-aed0e274-xmpl-40fe-8cba-2e78f000c0a7"
        },
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7ela928e-
xmpl-4d2b-8c01-782321bc6f66",
            "AlarmARN": "arn:aws:cloudwatch:us-east-2:123456789012:alarm:TargetTracking-
function:my-function:BLUE-AlarmLow-7ela928e-xmpl-4d2b-8c01-782321bc6f66"
        }
    ]
}
```

Application Auto Scaling crée deux alarmes dans CloudWatch. La première alarme se déclenche lorsque l'utilisation de la simultanéité provisionnée dépasse systématiquement 70 %. Lorsque cela se produit, Application Auto Scaling alloue plus de simultanéité provisionnée afin de réduire l'utilisation. La deuxième alarme se déclenche lorsque l'utilisation est constamment inférieure à 63 % (90 % de la cible de 70 %). Lorsque cela se produit, Application Auto Scaling réduit la simultanéité provisionnée de l'alias.

Dans l'exemple suivant, une fonction évolue entre une valeur minimale et une valeur maximale de simultanéité allouée en fonction du niveau d'utilisation. Lorsque le nombre de demandes ouvertes augmente, Application Auto Scaling augmente la simultanéité allouée par paliers importants jusqu'à ce qu'elle atteigne la valeur maximale configurée. La fonction continue d'évoluer selon une simultanéité standard jusqu'à ce que l'utilisation commence à baisser. Lorsque l'utilisation est faible dans la durée, Application Auto Scaling diminue régulièrement la simultanéité allouée par paliers plus petits.

Autoscaling with Provisioned Concurrency



Légende

- — Instances de la fonction
- — Demandes ouvertes
- Simultanéité allouée
- Simultanéité standard

Pour afficher les limites de simultanéité de votre compte dans une région, utilisez `get-account-settings`.

```
$ aws lambda get-account-settings
{
    "AccountLimit": {
```

```
        "TotalCodeSize": 80530636800,  
        "CodeSizeUnzipped": 262144000,  
        "CodeSizeZipped": 52428800,  
        "ConcurrentExecutions": 1000,  
        "UnreservedConcurrentExecutions": 900  
    },  
    "AccountUsage": {  
        "TotalCodeSize": 174913095,  
        "FunctionCount": 52  
    }  
}
```

Versions de fonction AWS Lambda

Vous pouvez utiliser les versions pour gérer le déploiement de vos fonctions AWS Lambda. Par exemple, vous pouvez publier une nouvelle version d'une fonction à des fins de test bêta sans affecter les utilisateurs de la version de production stable.

Le système crée une nouvelle version de votre fonction Lambda chaque fois que vous publiez la fonction. La nouvelle version est une copie de la version non publiée de la fonction. La version de la fonction inclut les informations suivantes :

- Le code de la fonction et toutes les dépendances associées.
- L'environnement d'exécution Lambda qui exécute la fonction.
- Tous les paramètres de la fonction, y compris les variables d'environnement.
- Un ARN (Amazon Resource Name) unique pour identifier cette version de la fonction.

Vous pouvez modifier le code et les paramètres de la fonction uniquement sur la version non publiée d'une fonction. Lorsque vous publiez une version, le code et la plupart des paramètres sont verrouillés afin de garantir une expérience cohérente pour les utilisateurs de cette version. Pour de plus amples informations sur la configuration des paramètres de la fonction, veuillez consulter [Configuration des fonctions dans la console AWS Lambda \(p. 53\)](#).

Pour créer une nouvelle version d'une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez la fonction que vous souhaitez publier.
3. Dans Actions, choisissez Publish new version (Publier une nouvelle version).

Après avoir publié la première version d'une fonction, la console Lambda affiche un menu déroulant contenant les versions disponibles. Le panneau Designer (Concepteur) affiche un qualificateur de version à la fin du nom de la fonction.

The screenshot shows the AWS Lambda Functions console. At the top, the navigation path is Lambda > Functions > MyTestFunction > 1. To the right, the ARN is listed as ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function:1. Below the path, the function name "MyTestFunction:1" is displayed in large bold letters. To the right of the function name are three buttons: "Version: 1" with a dropdown arrow, "Actions" with a dropdown arrow, and "Test1". Underneath the function name, there are two tabs: "Configuration" (which is selected and highlighted in orange) and "Monitoring". The main content area is titled "▼ Designer". It contains a key icon and a "Add trigger" button. To the right, there is a panel with the Lambda logo and the text "MyTestFunction:1". Below this, there is a "Layers" section with a stack icon and "(0)" indicating no layers are attached.

Pour afficher les versions actuelles de la fonction, choisissez une fonction, puis choisissez Qualifiers (Qualificateurs). Dans le menu Qualifiers (Qualificateurs) développé, choisissez l'onglet Versions. Le panneau Versions affiche la liste des versions pour la fonction sélectionnée. Si vous n'avez pas publié de version de la fonction sélectionnée, le panneau Versions répertorie uniquement la version \$LATEST.

Gestion des versions avec l'API Lambda

Pour publier une version d'une fonction, utilisez l'action d'API [PublishVersion \(p. 601\)](#).

L'exemple suivant publie une nouvelle version d'une fonction. La réponse renvoie les informations de configuration relatives à la nouvelle version, y compris le numéro de version et l'ARN de la fonction avec le suffixe de version.

```
$ aws lambda publish-version --function-name my-function
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
  "Version": "1",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs12.x",
  ...
}
```

Utilisation des versions

Vous référez votre fonction Lambda en utilisant son 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.

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

Vous pouvez utiliser cet ARN non qualifié dans toutes les opérations appropriées. Cependant, vous ne pouvez pas l'utiliser pour créer un alias.

Si vous décidez de ne pas publier de versions de fonction, vous pouvez utiliser l'ARN qualifié ou non qualifié dans votre mappage source d'événement pour appeler la fonction.

Lambda publie une nouvelle version de fonction seulement si le code n'a jamais été publié ou si le code a changé par rapport à la version la plus récente publiée. S'il n'y a pas de changement, la version de la fonction la plus récente publiée reste active.

Chaque version de fonction Lambda a un ARN unique. Après avoir publié une version, vous ne pouvez pas modifier l'ARN ou le code de fonction.

Stratégies basées sur une ressource

Lorsque vous utilisez une [stratégie basée sur les ressources \(p. 36\)](#) pour donner à un service, une ressource ou un compte l'accès à votre fonction, la portée de cette autorisation varie selon que vous l'avez appliquée ou non à une fonction ou à une version d'une fonction :

- Si vous utilisez un nom de fonction qualifié (tel que `helloworld:1`), l'autorisation est valide pour appeler la version 1 de la fonction `helloworld` uniquement avec son ARN qualifié. L'utilisation de tout autre ARN entraîne une erreur d'autorisation.
- Si vous utilisez un nom de fonction non qualifié (tel que `helloworld`), l'autorisation est valide uniquement pour appeler la fonction `helloworld` avec l'ARN de fonction non qualifié. L'utilisation de toute autre ARN, y compris `$LATEST`, entraîne une erreur d'autorisation.
- Si vous utilisez le nom de fonction qualifié `$LATEST` (par exemple `helloworld:$LATEST`), l'autorisation est valide pour appeler la fonction `helloworld` uniquement avec son ARN qualifié. L'utilisation d'un ARN non qualifié entraîne une erreur d'autorisation.

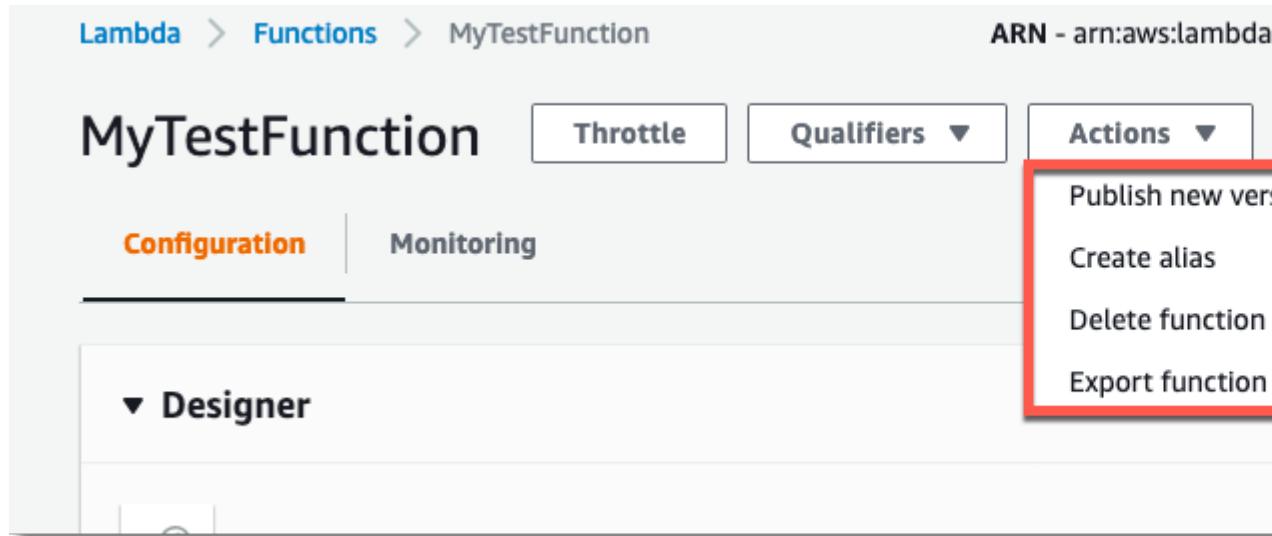
Vous pouvez simplifier la gestion des sources d'événements et des stratégies de ressources en utilisant les alias de fonction. Pour plus d'informations, consultez [Alias de fonctions AWS Lambda \(p. 72\)](#).

Alias de fonctions AWS Lambda

Vous pouvez créer un ou plusieurs alias pour une fonction AWS Lambda. Un alias Lambda s'apparente à un pointeur vers une version de fonction Lambda spécifique. Les utilisateurs peuvent accéder à la version de fonction à l'aide de l'ARN d'alias.

Pour créer un alias

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Dans Actions, choisissez Create alias (Créer un alias).



4. Dans le formulaire Create a new alias (Créer un nouvel alias), entrez un nom pour l'alias et une description facultative. Choisissez la version de fonction pour cet alias.

Pour afficher les alias actuellement définis pour une fonction, choisissez Qualifiers (Qualificateurs), puis l'onglet Alias.

Gestion des alias avec l'API Lambda

Pour créer un alias, utilisez la commande `create-alias`.

```
$ aws lambda create-alias --function-name my-function --name alias-name --function-version version-number --description " "
```

Pour modifier un alias de sorte qu'il pointe vers une nouvelle version de la fonction, utilisez la commande `update-alias`.

```
$ aws lambda update-alias --function-name my-function --name alias-name --function-version version-number
```

Les commandes de l'AWS CLI des étapes précédentes correspondent aux API AWS Lambda suivantes :

- [CreateAlias \(p. 494\)](#)
- [UpdateAlias \(p. 626\)](#)

Utilisation des alias

Chaque alias a un ARN unique. Un alias peut uniquement pointer vers une version de fonction, et non vers un autre alias. Vous pouvez mettre à jour un alias de sorte qu'il pointe vers une nouvelle version de la fonction.

Les sources d'événement telles que Amazon S3 appellent votre fonction Lambda. Ces sources d'événements gèrent un mappage qui identifie la fonction à appeler lorsque des événements se produisent. Si vous spécifiez un alias de fonction Lambda dans la configuration du mappage, vous n'avez pas besoin de mettre à jour le mappage lorsque la version de la fonction change.

Dans une stratégie de ressources, vous pouvez accorder des autorisations aux sources d'événements pour utiliser votre fonction Lambda. Si vous spécifiez un ARN d'alias dans la stratégie, vous n'avez pas besoin de mettre à jour la stratégie lorsque la version de la fonction change.

Stratégies basées sur une ressource

Lorsque vous utilisez une [stratégie basée sur les ressources \(p. 36\)](#) pour donner à un service, une ressource ou un compte l'accès à votre fonction, la portée de cette autorisation varie selon que vous l'avez appliquée ou non à un alias, à une version ou à la fonction. Si vous utilisez un nom d'alias (tel que `helloworld:PROD`), l'autorisation n'est valide que pour l'appel de la fonction `helloworld` à l'aide de l'ARN d'alias. Vous obtenez une erreur d'autorisation si vous utilisez un ARN de version ou l'ARN de fonction. Cela inclut l'ARN de version vers lequel pointe l'alias.

Par exemple, la commande d'AWS CLI suivante accorde des autorisations Amazon S3 pour invoquer l'alias PROD de la fonction Lambda `helloworld`. Notez que le paramètre `--qualifier` spécifie le nom de l'alias.

```
$ aws lambda add-permission --function-name helloworld \
--qualifier PROD --statement-id 1 --principal s3.amazonaws.com --action
lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket --source-account 123456789012
```

Dans ce cas, Amazon S3 est désormais en mesure d'appeler l'alias PROD. Lambda peut ensuite exécuter la version de la fonction Lambda `helloworld` référencée par l'alias PROD. Pour que cela fonctionne correctement, vous devez utiliser l'ARN de l'alias PROD dans la configuration des notifications du compartiment S3.

Configuration du routage d'alias

Utilisez la configuration de routage sur un alias pour envoyer une partie du trafic vers une deuxième version de la fonction. Par exemple, vous pouvez réduire le risque de déploiement d'une nouvelle version en configurant l'alias de sorte qu'il envoie la majeure partie du trafic vers la version existante, et seulement un faible pourcentage du trafic vers la nouvelle version.

Vous pouvez faire pointer un alias vers deux versions de fonction Lambda maximum. Les versions doivent répondre aux critères suivants :

- Les deux versions doivent avoir le même rôle d'exécution IAM.
- Les deux versions doivent avoir la même configuration de [file d'attente de lettres mortes](#) ou aucune configuration de file d'attente de lettres mortes.
- Les deux versions doivent être publiées. L'alias ne peut pas pointer vers `$LATEST`.

Pour configurer le routage sur un alias

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Vérifiez que la fonction a au moins deux versions publiées. Pour ce faire, choisissez Qualifiers (Qualificateurs) puis Versions pour afficher la liste des versions. Si vous devez créer des versions supplémentaires, suivez les instructions de la section [Versions de fonction AWS Lambda \(p. 70\)](#).
4. Dans le menu Actions, choisissez Create alias (Créer un alias).
5. Dans la fenêtre Create a new alias (Créer un nouvel alias) entrez une valeur pour Name (Nom), entrez éventuellement une valeur pour Description et choisissez la version de la fonction Lambda référencée par l'alias.
6. Dans Additional version (Version supplémentaire), spécifiez les éléments suivants :

- a. Choisissez la deuxième version de la fonction Lambda.
 - b. Entrez 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 (Version supplémentaire), 90 % du trafic est automatiquement affecté à la première version.
7. Sélectionnez Créer.

Configuration du routage d'alias

Utilisez les commandes `update-alias` et `create-alias` pour configurer la pondération du trafic entre deux versions de fonction. Lorsque vous créez ou mettez à jour l'alias, vous spécifiez la pondération du trafic dans le paramètre `routing-config`.

L'exemple suivant crée un alias (nommé `routing-alias`) pour une fonction Lambda. L'alias pointe vers la version 1 de la fonction. La version 2 de la fonction reçoit 3 % du trafic. Les 97 % restants du trafic sont acheminés vers la version 1.

```
$ aws lambda create-alias --name routing-alias --function-name my-function --function-version 1 \
--routing-config AdditionalVersionWeights={"2":0.03}
```

Utilisez la commande `update-alias` pour augmenter le pourcentage de trafic entrant vers la version 2. Dans l'exemple suivant, vous augmentez le trafic à 5 %.

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

Pour acheminer tout le trafic vers la version 2, utilisez la commande `UpdateAlias` pour modifier la propriété `function-version` afin que l'alias pointe vers la version 2. La commande réinitialise également la configuration de routage.

```
$ aws lambda update-alias --name routing-alias --function-name my-function \
--function-version 2 --routing-config AdditionalVersionWeights={}
```

Les commandes CLI des étapes précédentes correspondent aux opérations d'API AWS Lambda suivantes :

- [CreateAlias \(p. 494\)](#)
- [UpdateAlias \(p. 626\)](#)

Détermination de la version appelée

Lorsque vous configurez des pondérations de trafic entre deux versions de fonction, il existe deux façons de déterminer la version de fonction Lambda qui a été appelée :

- 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. Voici un exemple :

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

Pour les appels d'alias, Lambda utilise la dimension `Executed Version` pour filtrer les données de métriques en fonction de la version exécutée. Pour de plus amples informations, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

- 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 [environnement d'exécution personnalisé \(p. 126\)](#) 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 \(p. 6\)](#) tant que vous maintenez votre package de déploiement sous 3 MB.

Note

Une fonction peut utiliser un maximum de 5 couches 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 de plus amples informations, veuillez consulter [Limites AWS Lambda \(p. 30\)](#).

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 \(p. 80\)](#) pour l'octroi des autorisations d'utilisation des couches à certains comptes AWS spécifiques, à [AWS Organizations](#) ou à tous les comptes.

Les couches sont extraites dans le répertoire `/opt` de l'environnement d'exécution de la fonction. Chaque environnement d'exécution recherche les bibliothèques dans un autre emplacement sous `/opt`, en fonction du langage. [Structurez votre couche \(p. 79\)](#) 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 [Déclarer les ressources sans serveur](#) dans le Manuel du développeur Modèle d'application sans serveur AWS.

Sections

- [Configuration d'une fonction pour utiliser les couches \(p. 76\)](#)
- [Gestion des couches \(p. 77\)](#)
- [Inclusion de dépendances de bibliothèques dans une couche \(p. 79\)](#)
- [Autorisations d'utilisation des couches \(p. 80\)](#)
- [AWS CloudFormation and AWS SAM \(p. 80\)](#)
- [Exemples d'applications \(p. 81\)](#)

Configuration d'une fonction pour utiliser les couches

Vous pouvez spécifier jusqu'à 5 couches 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": "nodejs12.x",
    "Role": "arn:aws:iam::123456789012:role/service-role/lambda-role",
    "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
        }
    ],
    "RevisionId": "81cc64f5-5772-449a-b63e-12330476bcc4",
    ...
}
```

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 \(p. 122\)](#) 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",  
    "python3.8"  
]  
}
```

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 \(p. 80\)](#) 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 python3.8  
{  
    "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",  
                    "python3.8",  
                ]  
            }  
        }  
    ]  
}
```

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=GmvPV.3090EpkfN...",  
        "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",  
        "python3.8"  
    ]  
}
```

```
}
```

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. Les numéros de versions ne sont jamais réutilisés pour le nom d'une couche.

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 environnement d'exécution ou modifiez cette variable de chemin pour votre langage.

- 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/python3.8/site-packages` (répertoires du site)

Example Pillow

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

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

Example JSON

```
json.zip
# ruby/gems/2.5.0/
| build_info
| cache
| doc
| extensions
| gems
| # json-2.1.0
# specifications
# json-2.1.0.gemspec
```

- Java – `java/lib` (chemin de classe)

Example Jackson

```
jackson.zip
```

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

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

Example JQ

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

Pour plus d'informations sur les paramètres des chemins d'accès dans l'environnement d'exécution Lambda, consultez [Variables d'environnement d'exécution \(p. 57\)](#).

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 \(p. 41\)](#) ou à partir de la [stratégie basée sur les ressources \(p. 36\)](#) 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.

```
$ aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id xaccount \
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text
e210ffdc-e901-43b0-824b-5fc0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal": {"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

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.

Pour obtenir plus d'exemples, consultez [Octroi de l'accès de la couche à d'autres comptes \(p. 40\)](#).

AWS CloudFormation and AWS SAM

Utilisez le Modèle d'application sans serveur AWS (AWS SAM) dans vos modèles AWS CloudFormation pour automatiser la création et le mappage des couches dans votre application. Le type de ressource `AWS::Serverless::LayerVersion` crée une version de couche que vous pouvez référencer à partir de la configuration de votre fonction.

Example [blank-nodejs/template.yml](#) – Ressources sans serveur

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
```

```
Runtime: nodejs12.x
CodeUri: function/ .
Description: Call the AWS Lambda API
Timeout: 10
# Function's execution role
Policies:
  - AWSLambdaBasicExecutionRole
  - AWSLambdaReadOnlyAccess
  - AWSXrayWriteOnlyAccess
Tracing: Active
Layers:
  - !Ref libs
libs:
  Type: AWS::Serverless::LayerVersion
  Properties:
    LayerName: blank-nodejs-lib
    Description: Dependencies for the blank sample app.
    ContentUri: lib/ .
    CompatibleRuntimes:
      - nodejs12.x
```

Lorsque vous mettez à jour vos dépendances et que vous procédez au déploiement, AWS SAM crée une autre version de la couche et met à jour le mappage. Si vous déployez des modifications dans votre code sans modifier les dépendances, AWS SAM ignore la mise à jour de la couche, ce qui permet raccourcit le temps de chargement.

Exemples d'applications

Le référentiel GitHub de ce guide fournit des [exemples d'applications \(p. 300\)](#) qui démontrent l'utilisation de couches pour la gestion des dépendances.

- Node.js – [blank-nodejs](#)
- Python – [blank-python](#)
- Ruby – [blank-ruby](#)
- Java – [blank-java](#)

Pour plus d'informations sur l'exemple d'application « blank », consultez [Exemple d'application de fonction vide pour AWS Lambda \(p. 301\)](#).

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

Vous pouvez configurer une fonction pour qu'elle se connecte aux sous-réseaux privés d'un Virtual Private Cloud (VPC) de votre compte. Utilisez Amazon Virtual Private Cloud (Amazon VPC) pour créer un réseau privé pour les ressources telles que les bases de données, les instances de mémoire cache ou les services internes. Connectez votre fonction au VPC pour accéder à des ressources privées pendant l'exécution.

Pour connecter une fonction à un VPC

1. Ouvrez la [console Lambda](#) .
2. Choisissez une fonction.
3. Sous VPC, choisissez Modifier.
4. Choisissez VPC personnalisé.
5. Choisissez un VPC, des sous-réseaux et des groupes de sécurité.

Note

Connectez votre fonction à des sous-réseaux privés pour accéder aux ressources privées. Si votre fonction a besoin d'un accès Internet, utilisez [NAT \(p. 83\)](#). La connexion d'une fonction à un sous-réseau public ne lui donne pas accès à Internet ou à une adresse IP publique.

6. Choisissez Enregistrer.

Lorsque vous connectez une fonction à un VPC, Lambda crée une [interface réseau Elastic](#) pour chaque combinaison de groupe de sécurité et de sous-réseau dans la configuration VPC de votre fonction. Ce processus peut prendre environ une minute. Pendant ce temps, vous ne pouvez pas effectuer d'opérations supplémentaires qui ciblent la fonction, telles que la [création de versions \(p. 70\)](#) ou la mise à jour du code de la fonction. Pour les nouvelles fonctions, vous ne pouvez pas appeler la fonction jusqu'à ce que son état passe de `Pending` à `Active`. Pour les fonctions existantes, vous pouvez toujours appeler l'ancienne version pendant la mise à jour. Pour plus d'informations sur les états de fonction, consultez [Surveillance de l'état d'une fonction avec l'API Lambda \(p. 105\)](#).

Plusieurs fonctions connectées aux mêmes sous-réseaux partagent des interfaces réseau. Par conséquent, la connexion de fonctions supplémentaires à un sous-réseau qui dispose déjà d'une interface réseau gérée par Lambda est beaucoup plus rapide. Cependant, Lambda peut créer des interfaces réseau supplémentaires si vous avez de nombreuses fonctions ou des fonctions très occupées.

Si vos fonctions ne sont pas actives pendant une longue période, Lambda récupère ses interfaces réseau et la fonction devient `Idle`. Appelez une fonction inactive pour la réactiver. Le premier appel échoue et la fonction entre à nouveau dans un état en attente jusqu'à ce que l'interface réseau soit disponible.

Les fonctions Lambda ne peuvent pas se connecter directement à un VPC avec la [location d'instance dédiée](#). Pour vous connecter à des ressources dans un VPC dédié, [associez-les à un deuxième VPC avec une location par défaut](#).

Didacticiels de VPC

- Didacticiel : Configuration d'une fonction Lambda pour accéder à Amazon RDS dans un Amazon VPC (p. 258)
- Didacticiel : Configuration d'une fonction Lambda pour accéder à Amazon ElastiCache dans un Amazon VPC (p. 232)

Sections

- [Rôle d'exécution et autorisations utilisateur \(p. 82\)](#)
- [Configuration de l'accès au Amazon VPC avec l'API Lambda \(p. 83\)](#)
- [Accès à Internet et aux services pour des fonctions connectées à un VPC \(p. 83\)](#)
- [Exemples de configurations de VPC \(p. 84\)](#)

Rôle d'exécution et autorisations utilisateur

Lambda utilise les autorisations de votre fonction pour créer et gérer des interfaces réseau. Pour vous connecter à un VPC, le rôle d'exécution de votre fonction doit avoir les autorisations suivantes.

Autorisations du rôle d'exécution

- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DeleteNetworkInterface`

Ces autorisations sont incluses dans la stratégie gérée AWSLambdaVPCAccessExecutionRole.

Lorsque vous configurez la connectivité VPC, Lambda utilise vos autorisations pour vérifier les ressources réseau. Pour configurer une fonction pour vous connecter à un VPC, votre utilisateur IAM a besoin des autorisations suivantes.

Autorisations des utilisateurs

- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets
- ec2:DescribeVpcs

Configuration de l'accès au Amazon VPC avec l'API Lambda

Vous pouvez connecter une fonction à un VPC avec les API suivantes.

- [CreateFunction \(p. 504\)](#)
- [UpdateFunctionConfiguration \(p. 643\)](#)

Pour connecter votre fonction à un VPC lors de sa création avec l'AWS CLI, utilisez l'option `vpc-config` avec une liste d'ID de sous-réseaux privés et de groupes de sécurité. L'exemple suivant crée une fonction avec une connexion à un VPC avec deux sous-réseaux et un groupe de sécurité.

```
$ aws lambda create-function --function-name my-function \
--runtime nodejs12.x --handler index.js --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/lambda-role \
--vpc-config
SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912345678492fb
```

Pour connecter une fonction existante, utilisez l'option `vpc-config` avec la commande `update-function-configuration`.

```
$ aws lambda update-function-configuration --function-name my-function \
--vpc-config
SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912345678492fb
```

Pour déconnecter votre fonction d'un VPC, mettez à jour la configuration de la fonction avec une liste vide de sous-réseaux et de groupes de sécurité.

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

Accès à Internet et aux services pour des fonctions connectées à un VPC

Par défaut, Lambda exécute vos fonctions dans un VPC sécurisé avec accès aux services AWS et à Internet. Le VPC appartient à Lambda et ne se connecte pas au VPC par défaut de votre compte. Lorsque vous connectez une fonction à un VPC de votre compte, elle n'a pas accès à Internet, sauf si votre VPC le lui fournit.

Note

Plusieurs services offrent des [points de terminaison de VPC](#). Vous pouvez utiliser les points de terminaison d'un VPC pour vous connecter aux services AWS à partir d'un VPC sans accès à Internet.

L'accès à Internet depuis un sous-réseau privé nécessite la traduction d'adresses réseau (NAT). Pour accorder à votre fonction l'accès à Internet, acheminez le trafic sortant vers une passerelle NAT dans un sous-réseau public. La passerelle NAT possède une adresse IP publique et peut se connecter à Internet via la passerelle Internet du VPC. Pour plus d'informations, consultez [Passerelles NAT](#) dans le Amazon VPC Guide de l'utilisateur.

Exemples de configurations de VPC

Des exemples de modèle AWS CloudFormation pour les configurations de VPC que vous pouvez utiliser avec des fonctions Lambda sont disponibles dans le référentiel GitHub de ce guide. Il existe deux modèles :

- [vpc-private.yaml](#) – VPC avec deux sous-réseaux privés et points de terminaison de VPC pour Amazon Simple Storage Service et Amazon DynamoDB. Vous pouvez utiliser ce modèle pour créer un VPC pour les fonctions qui n'ont pas besoin d'un accès Internet. Cette configuration prend en charge l'utilisation d'Amazon S3 et de DynamoDB avec le kit de développement SDK AWS, ainsi que l'accès aux ressources de base de données dans le même VPC via une connexion réseau locale.
- [vpc-privatepublic.yaml](#) – VPC avec deux sous-réseaux privés, des points de terminaison de VPC, un sous-réseau public avec une passerelle NAT et une passerelle Internet. Le trafic lié à Internet depuis les fonctions des sous-réseaux privés est acheminé vers la passerelle NAT par une table de routage.

Pour utiliser un modèle afin de créer un VPC, choisissez Create stack (Créer une pile) dans la [console AWS CloudFormation](#) et suivez les instructions.

Configuration de l'accès à une base de données pour une fonction Lambda

Vous pouvez utiliser la console Lambda pour créer un proxy de base de données Amazon RDS Proxy pour votre fonction. Un proxy de base de données gère un groupe de connexions de base de données et relaie les requêtes d'une fonction. Cela permet à une fonction d'atteindre des niveaux de [simultanéité](#) (p. 18) élevés sans épuiser les connexions de base de données.

Pour créer un proxy de base de données

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Add database proxy (Ajouter un proxy de base de données).
4. Configurez les options suivantes.
 - Identifiant de proxy – Nom du proxy.
 - Instance de base de données RDS – Instance ou cluster de base de données [MySQL ou PostgreSQL pris en charge](#).
 - Secret – Secret Secrets Manager avec le nom d'utilisateur et le mot de passe de base de données.

Example secret

```
{
```

```
    "username": "admin",
    "password": "e2abcecxmpldc897"
}
```

- Rôle IAM – Rôle IAM autorisé à utiliser le secret et une stratégie d'approbation Amazon RDS permettant d'assumer le rôle.
 - Authentification – Méthode d'authentification et d'autorisation permettant de se connecter au proxy à partir de votre code de fonction.
5. Choisissez Add (Ajouter).

Tarification

Amazon RDS facture un prix horaire pour les proxies. Ce prix est déterminé par la taille de l'instance de votre base de données. Pour plus d'informations, consultez [Tarification du proxy RDS](#).

La création du proxy prend quelques minutes. Lorsque le proxy est disponible, configurez votre fonction de sorte qu'elle se connecte au point de terminaison de proxy au lieu du point de terminaison de base de données.

La [tarification standard Amazon RDS Proxy](#) s'applique. Pour plus d'informations, consultez [Gestion des connexions avec le proxy Amazon RDS](#) dans le Guide de l'utilisateur Amazon Aurora.

Rubriques

- [Utilisation des autorisations de la fonction pour l'authentification \(p. 85\)](#)
- [Exemple d'application \(p. 86\)](#)

Utilisation des autorisations de la fonction pour l'authentification

Par défaut, vous pouvez vous connecter à un proxy avec le même nom d'utilisateur et mot de passe que ceux utilisés pour vous connecter à la base de données. La seule différence dans votre code de fonction réside dans le point de terminaison auquel le client de base de données se connecte. L'inconvénient de cette méthode est que vous devez exposer le mot de passe au code de fonction, soit en le configurant dans une variable d'environnement sécurisée, soit en le récupérant depuis Secrets Manager.

Vous pouvez créer un proxy de base de données qui utilise les informations d'identification IAM de la fonction pour l'authentification et l'autorisation au lieu d'un mot de passe. Pour utiliser les autorisations de la fonction pour se connecter au proxy, définissez Authentication (Authentification) sur Execution role (Rôle d'exécution).

La console Lambda ajoute l'autorisation requise (`rds-db:connect`) au rôle d'exécution. Vous pouvez ensuite utiliser le kit SDK AWS pour générer un jeton qui lui permet de se connecter au proxy. L'exemple suivant montre comment configurer une connexion à la base de données avec la bibliothèque `mysql2` dans Node.js.

Example [dbadmin/index-iam.js](#) – Signataire du kit SDK AWS

```
const signer = new AWS.RDS.Signer({
  region: region,
  hostname: host,
  port: sqlport,
  username: username
})
```

```
exports.handler = async (event) => {
  let connectionConfig = {
    host      : host,
    user      : username,
    database  : database,
    ssl: 'Amazon RDS',
    authPlugins: { mysql_clear_password: () => () => signer.getAuthToken() }
  }
  var connection = mysql.createConnection(connectionConfig)
  var query = event.query
  var result
  connection.connect()
```

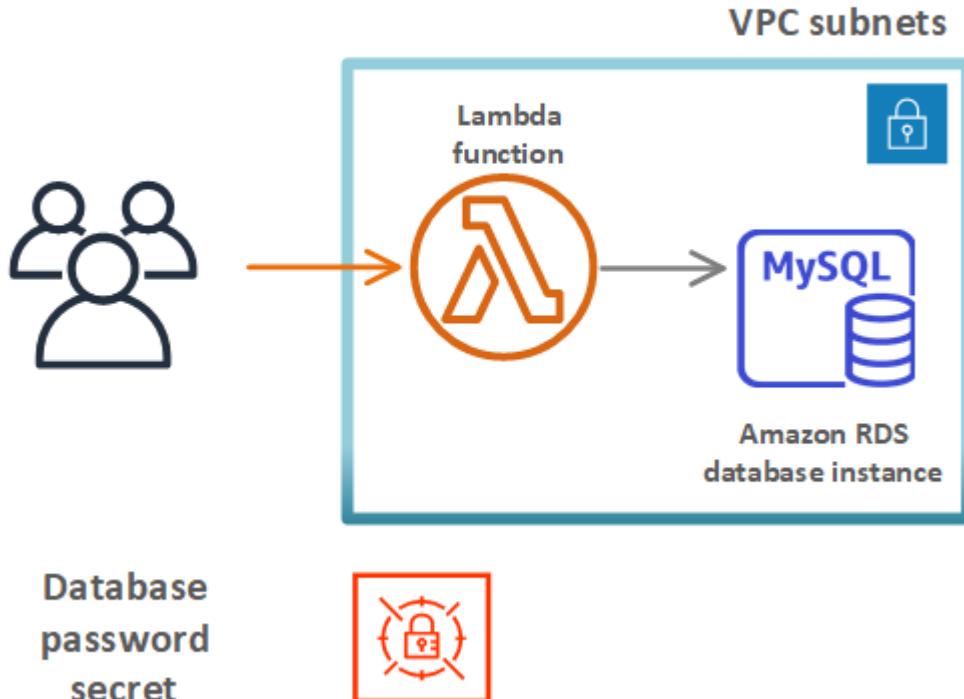
Pour plus d'informations, consultez [Authentification de base de données IAM](#) dans le Amazon RDS Guide de l'utilisateur.

Exemple d'application

Des exemples d'application démontrant l'utilisation d'Lambda avec une base de données Amazon RDS sont disponibles dans le référentiel GitHub de ce guide. Il y a deux applications :

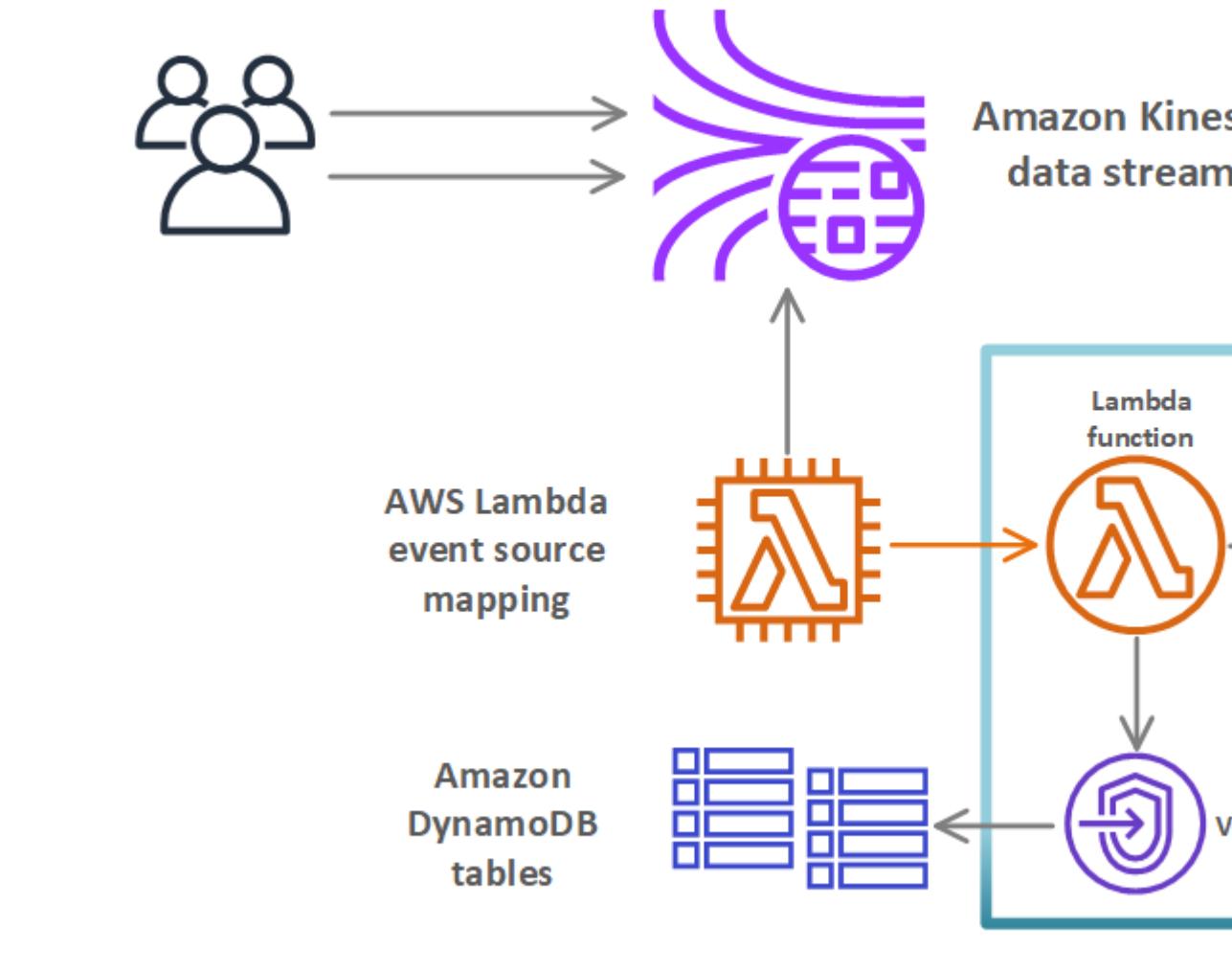
- [RDS MySQL](#) – Le modèle AWS CloudFormation template-vpcrds.yml crée une base de données MySQL 5.7 dans un VPC privé. Dans l'exemple d'application, une fonction Lambda relaie des requêtes sur la base de données. Les modèles de fonction et de base de données utilisent tous les deux Secrets Manager pour accéder aux informations d'identification de la base de données.

RDS MySQL Application



- [List Manager](#) – Une fonction de processeur lit les événements d'un flux Kinesis. Il utilise les données des événements pour mettre à jour les tables DynamoDB et stocke une copie de l'événement dans une base de données MySQL.

List Manager Application



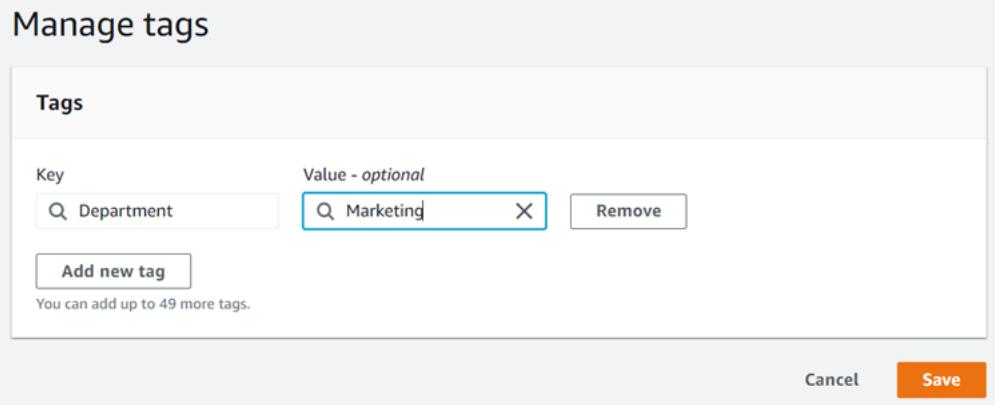
Pour utiliser les exemples d'application, suivez les instructions du référentiel GitHub : [RDS MySQL, List Manager](#).

Balisage des fonctions Lambda

Vous pouvez marquer les fonctions Lambda pour les organiser par propriétaire, projet ou service. Les balises sont des paires clé-valeur de forme libre qui sont prises en charge par les services AWS pour filtrer les ressources et ajouter des détails aux rapports de facturation.

Pour ajouter des balises à une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Balises, choisissez Gérer les balises.
4. Entrez une clé et une valeur. Pour ajouter des balises supplémentaires, choisissez Ajouter une nouvelle balise.

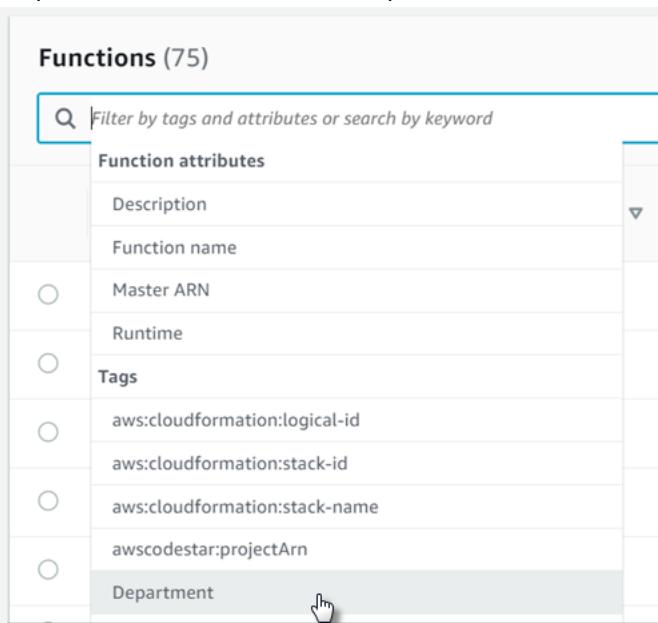


5. Choisissez Enregistrer.

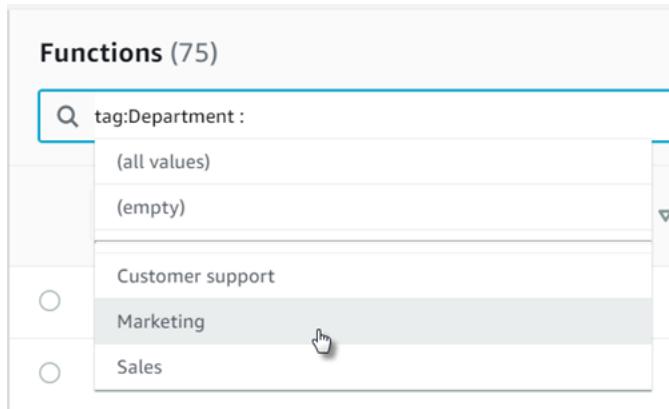
Vous pouvez filtrer les fonctions en fonction de la présence ou de la valeur d'une balise avec la console Lambda ou avec l'API Groupes de ressources AWS. Les balises s'appliquent au niveau de la fonction, pas aux versions ni aux alias. Les balises ne font pas partie de la configuration spécifique à la version qui fait l'objet d'un instantané lorsque vous publiez une version.

Pour filtrer des fonctions avec des balises

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Cliquez dans la barre de recherche pour afficher la liste des attributs de fonction et des clés de balise.



3. Choisissez une clé de balise pour afficher la liste des valeurs utilisées dans la région actuelle.
4. Choisissez une valeur pour voir les fonctions avec cette valeur, ou choisissez (toutes les valeurs) pour voir toutes les fonctions qui ont une balise avec cette clé.



La barre de recherche prend également en charge la recherche de clés de balise. Tapez tag pour afficher uniquement une liste de clés de balise, ou commencez à taper le nom d'une clé pour la rechercher dans la liste.

Avec AWS Billing and Cost Management, vous pouvez utiliser des balises pour personnaliser les rapports de facturation et créer des rapports de répartition des coûts. Pour de plus amples informations, veuillez consulter le [rapport mensuel sur la répartition des coûts](#) et [Utilisation des balises de répartition des coûts](#) dans le Guide de l'utilisateur AWS Billing and Cost Management.

Sections

- [Utilisation des balises avec AWS CLI \(p. 89\)](#)
- [Exigences relatives à la clé et à la valeur des balises \(p. 90\)](#)

Utilisation des balises avec AWS CLI

Lorsque vous créez une nouvelle fonction Lambda, vous pouvez inclure des balises avec l'option --tags.

```
$ aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pour ajouter des balises dans une fonction existante, utilisez la commande tag-resource.

```
$ aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pour supprimer des balises, utilisez la commande untag-resource.

```
$ aws lambda untag-resource --resource function arn \
--tag-keys Department
```

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 \(p. 592\)](#) : 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
```

- [GetFunction \(p. 538\)](#) : vous fournissez le nom de votre fonction Lambda pour afficher la liste des balises associées à cette fonction :

```
$ aws lambda get-function --function-name my-function
```

Vous pouvez également utiliser l'API [GetResources](#) du service de balisage AWS pour filtrer vos ressources par balises. 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](#).

Exigences relatives à la clé et à la valeur des balises

Les exigences 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 : + - = . _ : / @.

Appel de fonctions AWS Lambda

Vous pouvez appeler les fonctions Lambda directement [avec la console Lambda \(p. 4\)](#) l'API Lambda, le kit AWS SDK, l'AWS CLI et les boîtes à outils AWS. Vous pouvez également configurer d'autres services AWS pour appeler votre fonction, ou configurer Lambda pour lire à partir d'un flux ou d'une file d'attente et appeler votre fonction.

Lorsque vous appelez une fonction, vous pouvez choisir de la faire de façon synchrone ou asynchrone. Avec un [appel synchrone \(p. 92\)](#), vous attendez de la fonction qu'elle traite l'événement et renvoie une réponse. Avec l'[appel asynchrone \(p. 93\)](#), Lambda met dans la file d'attente l'événement à traiter et renvoie une réponse immédiatement. Pour les appels asynchrones, Lambda gère les nouvelles tentatives et peut envoyer des enregistrements d'appel à une [destination \(p. 96\)](#).

Pour utiliser votre fonction pour traiter automatiquement les données, ajoutez un ou plusieurs déclencheurs. Un déclencheur est une ressource Lambda ou une ressource dans un autre service que vous configurez pour appeler votre fonction en réponse à des événements du cycle de vie, à des demandes externes ou à une planification. Votre fonction peut avoir plusieurs déclencheurs. Chaque déclencheur agit comme un client appelant indépendamment votre fonction. Chaque événement que Lambda transmet à votre fonction n'a que les données d'un seul client ou d'un seul déclencheur.

Pour traiter les éléments à partir d'un flux ou d'une file d'attente, vous pouvez créer un [mappage de source d'événement \(p. 101\)](#). Un mappage de source d'événement est une ressource Lambda qui lit des éléments d'une file d'attente Amazon SQS, un flux Amazon Kinesis ou un flux Amazon DynamoDB et les envoie à votre fonction par lots. Chaque événement traité par votre fonction peut contenir des centaines ou des milliers d'éléments.

D'autres services et ressources AWS appellent votre fonction directement. Par exemple, vous pouvez configurer CloudWatch Events pour appeler votre fonction selon un minuteur, ou configurer Amazon S3 pour appeler votre fonction lorsqu'un objet est créé. La méthode de chaque service pour appeler votre fonction est différente, ainsi que la structure de l'événement et la façon dont vous le configurer. Pour en savoir plus consultez [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

En fonction de qui invoque votre fonction et de la façon dont elle est invoquée, le comportement de dimensionnement et les types d'erreurs qui surviennent peuvent changer. Lorsque vousappelez une fonction synchrone, vous recevez des erreurs dans la réponse et pouvez réessayer. Lorsque vousappelez de façon asynchrone, utilisez un mappage de source d'événement ou configurez un autre service pour appeler votre fonction, les exigences de nouvelles tentatives et la façon dont votre fonction est dimensionnée pour gérer un grand nombre d'événements peuvent varier. Pour plus d'informations, veuillez consulter [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#) et [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).

Rubriques

- [Appel synchrone \(p. 92\)](#)
- [Appel asynchrone \(p. 93\)](#)
- [Mappages de source d'événement AWS Lambda \(p. 101\)](#)
- [Surveillance de l'état d'une fonction avec l'API Lambda \(p. 105\)](#)
- [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#)
- [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#)
- [Appel de fonctions Lambda avec AWS Mobile SDK pour Android \(p. 113\)](#)

Appel synchrone

Lorsque vous appelez une fonction de façon synchrone, Lambda exécute la fonction et attend une réponse. Lorsque l'exécution de la fonction s'achève, Lambda renvoie la réponse du code de la fonction avec des données supplémentaires, telles que la version de la fonction qui a été exécutée. Pour appeler une fonction de façon synchrone avec AWS CLI, utilisez la commande `invoke`.

```
$ aws lambda invoke --function-name my-function --payload '{ "key": "value" }'  
response.json  
{  
    "ExecutedVersion": "$LATEST",  
    "StatusCode": 200  
}
```

Le diagramme suivant montre des clients qui appellent une fonction Lambda de manière synchrone. Lambda envoie les événements directement à la fonction et renvoie la réponse de la fonction à l'appelant.

Synchronous Invocation



`payload` est une chaîne qui contient un événement au format JSON. Le nom du fichier où l'AWS CLI écrit la réponse de la fonction est `response.json`. Si la fonction renvoie un objet ou une erreur, la réponse est l'objet ou d'une erreur au format JSON. Si la fonction se termine sans erreur, la réponse est `null`.

La sortie de la commande, qui s'affiche dans le terminal, inclut des informations provenant des en-têtes de la réponse à partir de Lambda. Cela inclut la version qui a traité l'événement (utile lorsque vous utilisez des [alias](#) (p. 72)) et le code d'état renvoyé par Lambda. Si Lambda a été en mesure d'exécuter la fonction, le code d'état est 200, même si la fonction a renvoyé une erreur.

Note

Pour les fonctions avec un long délai d'attente, votre client peut être déconnecté pendant l'appel synchrone, pendant qu'il attend une réponse. Configurez votre client HTTP, SDK, pare-feu, proxy ou système d'exploitation pour permettre des connexions longues avec des paramètres de délai d'attente ou de keep-alive.

Si Lambda n'est pas en mesure d'exécuter la fonction, l'erreur s'affiche dans la sortie.

```
$ aws lambda invoke --function-name my-function --payload value response.json  
An error occurred (InvalidRequestContentException) when calling the Invoke operation: Could  
not parse request body into json: Unrecognized token 'value': was expecting ('true',  
'false' or 'null')
```

```
at [Source: (byte[])"value"; line: 1, column: 11]
```

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour en savoir plus sur l'API `Invoke`, y compris une liste complète de paramètres, d'en-têtes et d'erreurs, consultez [Invoke \(p. 565\)](#).

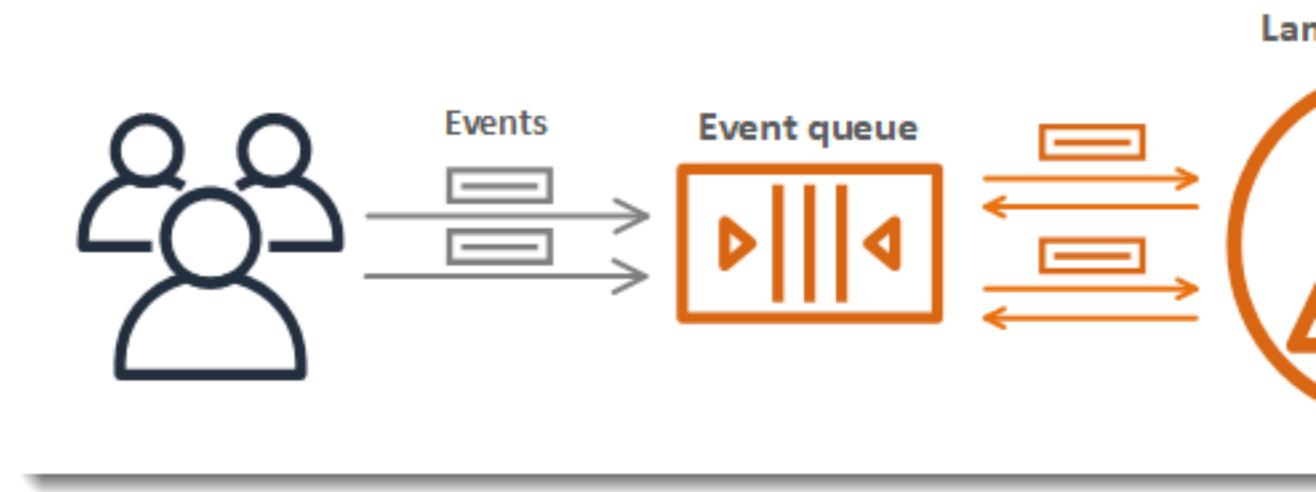
Lorsque vous appelez une fonction directement, vous pouvez vérifier la réponse pour les erreurs et réessayer. La AWS CLI et le kit SDK AWS réessaient automatiquement selon les délais d'expiration du client, des blocages et des erreurs de service. Pour de plus amples informations, veuillez consulter [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).

Appel asynchrone

Plusieurs services AWS, comme Amazon Simple Storage Service (Amazon S3) et Amazon Simple Notification Service (Amazon SNS), appellent les fonctions de manière asynchrone pour traiter les événements. Lorsque vous appelez une fonction de manière asynchrone, vous n'attendez pas de réponse de la part du code de la fonction. Vous donnez l'événement à Lambda, et Lambda se charge du reste. Vous pouvez configurer la façon dont Lambda gère les erreurs et envoyer les enregistrements d'appel à une ressource en aval pour regrouper les composants de votre application.

Le diagramme suivant montre des clients qui appellent une fonction Lambda de manière asynchrone. Lambda met les événements en file d'attente avant de les envoyer à la fonction.

Asynchronous Invocation



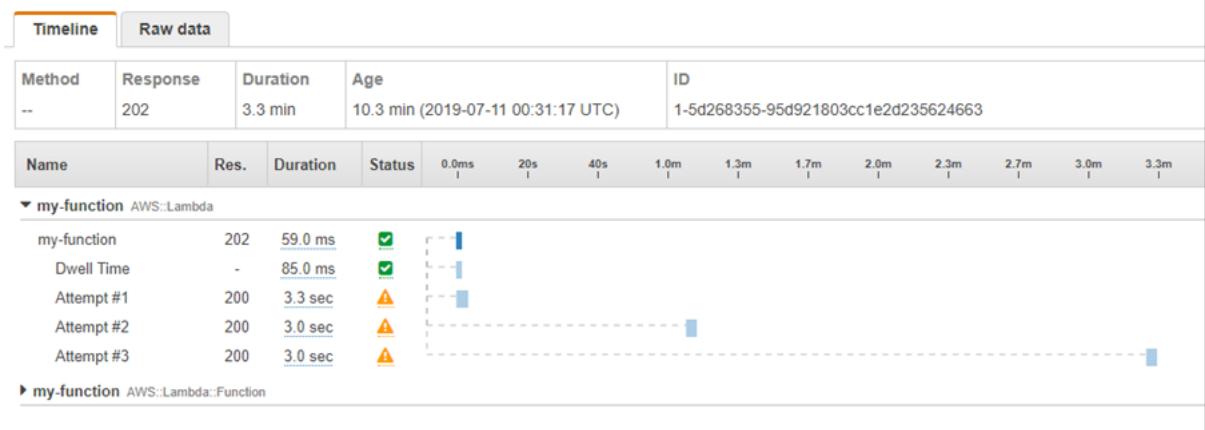
Pour un appel asynchrone, Lambda place l'événement dans une file d'attente et renvoie une réponse indiquant sa réussite, sans informations supplémentaires. Un processus distinct lit les événements à partir de la file d'attente et les envoie à votre fonction. Pour appeler une fonction de façon asynchrone, définissez le paramètre du type d'appel sur Event.

```
$ aws lambda invoke --function-name my-function --invocation-type Event --payload
'{ "key": "value" }' response.json
{
    "StatusCode": 202
}
```

Le fichier de sortie (`response.json`) ne contient pas d'informations, mais il est tout de même créé lorsque vous exécutez cette commande. Si Lambda n'est pas en mesure d'ajouter l'événement à la file d'attente, le message d'erreur s'affiche dans la sortie de la commande.

Lambda gère la file d'attente d'événements asynchrones de la fonction et effectue de nouvelles tentatives en cas d'erreur. Si la fonction renvoie une erreur, Lambda tente de l'exécuter deux nouvelles fois, avec une minute d'attente entre les deux premières tentatives et deux minutes entre la deuxième et la troisième tentatives. Les erreurs de fonction comprennent des erreurs renvoyées par le code de la fonction et des erreurs renvoyées par l'environnement d'exécution de la fonction, comme les délais d'expiration.

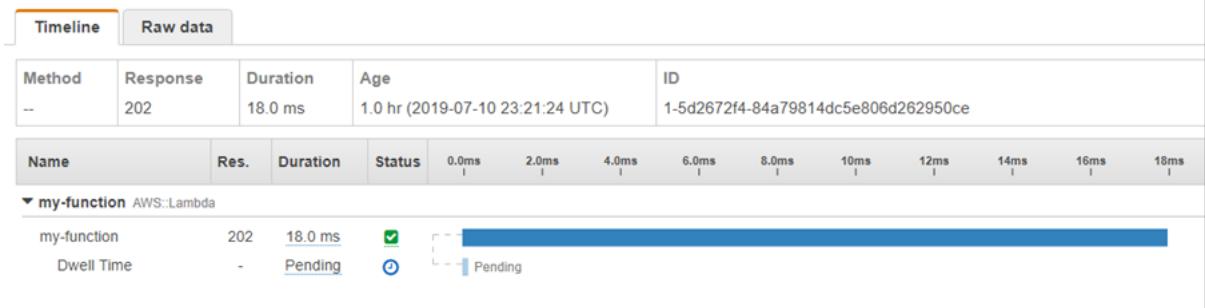
Traces > Details



Si la fonction n'a pas suffisamment de simultanéité disponible pour traiter tous les événements, des demandes supplémentaires sont bloquées. Pour les erreurs de limitation (erreur 429) et les erreurs système (série 500), Lambda renvoie l'événement à la file d'attente et tente d'exécuter la fonction à nouveau pour une durée maximale de 6 heures. L'intervalle de nouvelle tentative augmente de manière exponentielle de 1 seconde après la première tentative jusqu'à un maximum de 5 minutes. Toutefois, il peut être plus long si la file d'attente est sauvegardée. Lambda réduit également le débit de lecture des événements à partir de la file d'attente.

L'exemple suivant montre un événement qui a été ajouté avec succès à la file d'attente, mais qui est toujours en attente une heure plus tard en raison de la limitation.

Traces > Details



Même si votre fonction ne renvoie pas d'erreur, elle peut recevoir le même événement plusieurs fois à partir de Lambda, car la file d'attente elle-même est cohérente. Si la fonction ne peut pas à prendre en charge les événements entrants, ils peuvent également être supprimés de la file d'attente sans être envoyés à la fonction. Assurez-vous que le code de votre fonction gère correctement les événements dupliqués et que vous avez suffisamment de simultanéité disponible pour gérer tous les appels.

Lorsque la file d'attente est sauvegardée, les nouveaux événements peuvent expirer avant qu'Lambda n'ait la possibilité de les envoyer à votre fonction. Lorsqu'un événement expire ou échoue à toutes les tentatives de traitement, Lambda le rejette. Vous pouvez configurer la gestion des erreurs (p. 96) pour une fonction afin de réduire le nombre de nouvelles tentatives effectuées par Lambda, ou pour supprimer les événements non traités plus rapidement.

Vous pouvez également configurer Lambda pour envoyer un enregistrement d'appel à un autre service. Lambda prend en charge les destinations (p. 96) suivantes pour l'appel asynchrone.

- Amazon SQS – File d'attente SQS standard.
- Amazon SNS – Rubrique SNS.

- AWS Lambda – Fonction Lambda.
- Amazon EventBridge – Bus d'événements EventBridge.

L'enregistrement d'appel contient des détails sur la demande et la réponse au format JSON. Vous pouvez configurer des destinations distinctes pour les événements dont le traitement échoue et ceux dont le traitement aboutit. Vous pouvez également configurer une file d'attente SQS ou une rubrique SNS en tant que [file d'attente de lettres mortes \(p. 99\)](#) pour les événements ignorés. Pour les files d'attente de lettres mortes, Lambda envoie uniquement le contenu de l'événement, sans informations supplémentaires sur la réponse.

Sections

- [Configuration de la gestion des erreurs pour les appels asynchrones \(p. 96\)](#)
- [Configuration des destinations pour les appels asynchrones \(p. 96\)](#)
- [API de configuration d'appel asynchrone \(p. 98\)](#)
- [Files d'attente de lettres mortes de fonction AWS Lambda \(p. 99\)](#)

Configuration de la gestion des erreurs pour les appels asynchrones

Utilisez la console Lambda pour configurer les paramètres de gestion des erreurs sur une fonction, une version ou un alias.

Pour configurer la gestion des erreurs

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Asynchronous invocation (Appel asynchrone), choisissez Edit (Modifier).
4. 以下を設定します。
 - Maximum age of event (Âge maximal de l'événement) – Durée maximale pendant laquelle Lambda conserve un événement dans la file d'attente d'événements asynchrones (jusqu'à 6 heures).
 - Retry attempts (Nouvelles tentatives) – Nombre de nouvelles tentatives effectuées par Lambda lorsque la fonction renvoie une erreur (entre 0 et 2).
5. Choisissez Enregistrer.

Lorsqu'un événement d'appel dépasse l'âge maximal ou échoue une fois le nombre maximal de tentatives atteint, Lambda le rejette. Pour conserver une copie des événements supprimés, configurez une destination en cas d'échec.

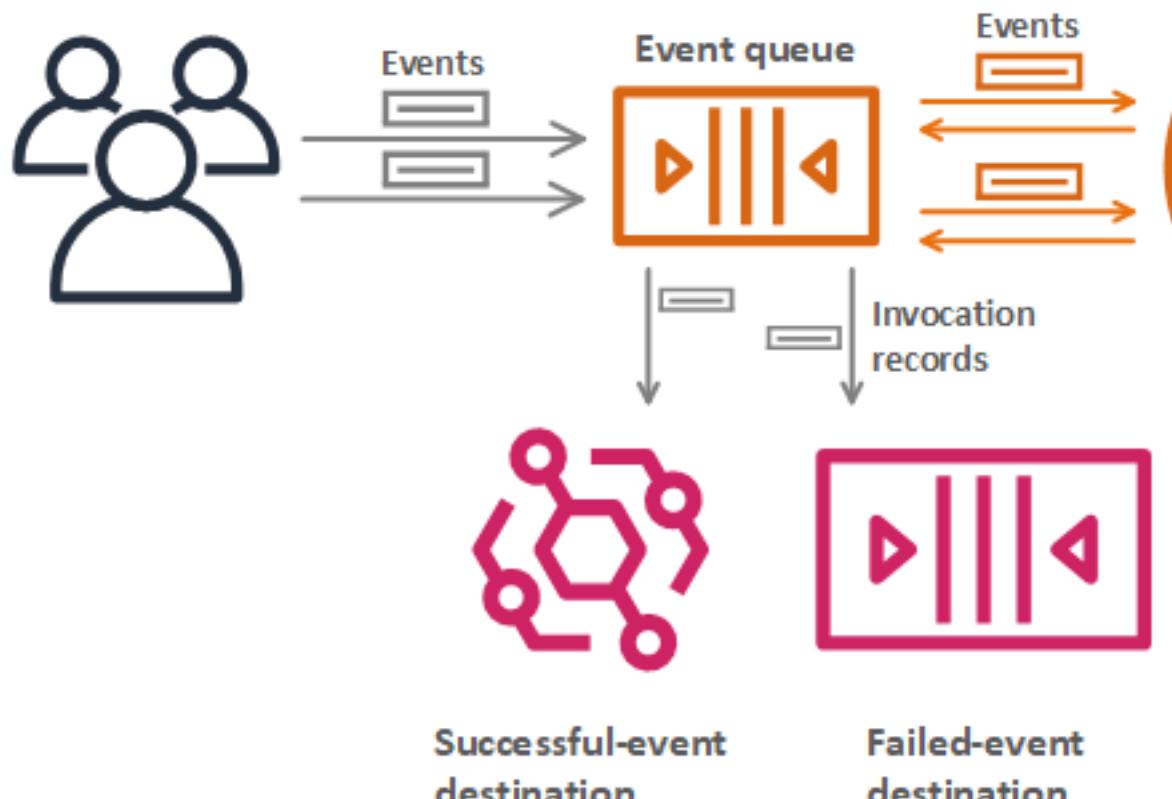
Configuration des destinations pour les appels asynchrones

Pour envoyer des enregistrements d'appel asynchrone à un autre service, ajoutez une destination à votre fonction. Vous pouvez configurer des destinations distinctes pour les événements dont le traitement échoue et ceux dont le traitement aboutit. Comme les paramètres de gestion des erreurs, vous pouvez configurer des destinations au niveau d'une fonction, d'une version ou d'un alias.

L'exemple suivant montre une fonction qui traite les appels asynchrones. Lorsque la fonction renvoie une réponse de réussite ou se termine sans générer d'erreur, Lambda envoie un enregistrement de l'appel à

un bus d'événements EventBridge. Lorsqu'un événement échoue pour toutes les tentatives de traitement, Lambda envoie un enregistrement d'appel à une file d'attente Amazon SQS.

Destinations for Asynchronous Invocation



Pour envoyer des événements à une destination, votre fonction a besoin d'autorisations supplémentaires. Ajoutez une stratégie avec les autorisations requises au [rôle d'exécution \(p. 33\)](#) de votre fonction. Chaque service de destination nécessite une autorisation différente, comme suit :

- Amazon SQS – [sns:SendMessage](#)
- Amazon SNS – [sns:Publish](#)
- Lambda – [lambda:InvokeFunction](#)
- EventBridge – [events:PutEvents](#)

Ajoutez des destinations à votre fonction dans la console Lambda du concepteur de fonctions.

Pour configurer une destination pour des enregistrements d'appel asynchrone

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.

3. Sous Designer (Concepteur), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Asynchronous invocation (Appel asynchrone).
5. Pour Condition choisissez l'une des options suivantes :
 - On failure (En cas d'échec) – Envoyer un enregistrement lorsque l'événement échoue lors de toutes les tentatives de traitement ou dépasse l'âge maximal.
 - On success (En cas de succès) – Envoyer un enregistrement lorsque la fonction traite avec succès un appel asynchrone.
6. Pour Type de destination, choisissez le type de ressource qui reçoit l'enregistrement d'appel.
7. Pour Destination, choisissez une ressource.
8. Choisissez Enregistrer.

Lorsqu'un appel correspond à la condition, Lambda envoie un document JSON avec des détails sur l'appel à la destination. L'exemple suivant illustre un enregistrement d'appel pour un événement dont le traitement a échoué à trois reprises en raison d'une erreur de fonction.

Example enregistrement d'appel

```
{  
    "version": "1.0",  
    "timestamp": "2019-11-14T18:16:05.568Z",  
    "requestContext": {  
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",  
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:  
$LATEST",  
        "condition": "RetriesExhausted",  
        "approximateInvokeCount": 3  
    },  
    "requestPayload": {  
        "ORDER_IDS": [  
            "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",  
            "637de236-e7b2-464e-xmpl-baf57f86bb53",  
            "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"  
        ]  
    },  
    "responseContext": {  
        "statusCode": 200,  
        "executedVersion": "$LATEST",  
        "functionError": "Unhandled"  
    },  
    "responsePayload": {  
        "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited  
before completing request"  
    }  
}
```

L'enregistrement d'appel contient des détails sur l'événement, la réponse et la raison pour laquelle l'enregistrement a été envoyé.

API de configuration d'appel asynchrone

Pour gérer les paramètres d'appel asynchrone avec l'interface de ligne de commande AWS ou AWS SDK, utilisez les opérations d'API suivantes.

- [PutFunctionEventInvokeConfig](#)
- [GetFunctionEventInvokeConfig](#)

- [UpdateFunctionEventInvokeConfig](#)
- [ListFunctionEventInvokeConfigs](#)
- [DeleteFunctionEventInvokeConfig](#)

Pour configurer l'appel asynchrone avec l'interface de ligne de commande AWS, utilisez la commande `put-function-event-invoke-config`. L'exemple suivant montre comment configurer une fonction avec un âge d'événement maximal de 1 heure et aucune nouvelle tentative.

```
$ aws lambda put-function-event-invoke-config --function-name error \
--maximum-event-age-in-seconds 3600 --maximum-retry-attempts 0
{
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

La commande `put-function-event-invoke-config` remplace toute configuration existante sur la fonction, la version ou l'alias. Pour configurer une option sans en réinitialiser d'autres, utilisez `update-function-event-invoke-config`. L'exemple suivant configure Lambda pour envoyer un enregistrement à une file d'attente SQS nommée destination lorsqu'un événement ne peut pas être traité.

```
$ aws lambda update-function-event-invoke-config --function-name error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'
{
    "LastModified": 1573687896.493,
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {
            "Destination": "arn:aws:sqs:us-east-2:123456789012:destination"
        }
    }
}
```

Files d'attente de lettres mortes de fonction AWS Lambda

Comme alternative à une [destination réservée aux échecs \(p. 96\)](#), vous pouvez configurer votre fonction avec une file d'attente de lettres mortes pour enregistrer les événements ignorés en vue d'un traitement ultérieur. Une file d'attente de lettres mortes agit de la même manière qu'une destination réservée aux échecs en ce sens qu'elle est utilisée lorsqu'un événement échoue à toutes les tentatives de traitement ou expire sans être traité. Toutefois, une file d'attente de lettres mortes fait partie d'une configuration spécifique à la version de la fonction. Elle est donc verrouillée lorsque vous publiez une version. Les destinations réservées aux échecs prennent également en charge des cibles supplémentaires et incluent des détails sur la réponse de la fonction dans l'enregistrement d'appel.

Si vous n'avez pas de file d'attente ou de rubrique, créez-en une. Choisissez le type de cible qui correspond à votre cas d'utilisation.

- **File d'attente Amazon SQS** – Une file d'attente retient les événements qui ont échoué jusqu'à ce qu'ils soient récupérés. Vous pouvez récupérer les événements manuellement, ou [configurer Lambda pour lire à partir de la file d'attente \(p. 285\)](#) et appeler une fonction.

Créez une file d'attente dans la [console Amazon SQS](#).

- **Rubrique Amazon SNS** – Une rubrique relaie les événements qui ont échoué vers une ou plusieurs destinations. Vous pouvez configurer une rubrique pour envoyer des événements à une adresse électronique, une fonction Lambda ou un point de terminaison HTTP.

Créez une rubrique dans la [console Amazon SNS](#).

Pour envoyer des événements à une file d'attente ou une rubrique, votre fonction a besoin d'autorisations supplémentaires. Ajoutez une stratégie avec les autorisations requises au [rôle d'exécution \(p. 33\)](#) de votre fonction.

- Amazon SQS – [sq:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Si la file d'attente ou la rubrique cible est chiffrée avec une clé gérée par le client, le rôle d'exécution doit également être un utilisateur dans la [stratégie basée sur les ressources](#) de la clé.

Après avoir créé la cible et mis à jour votre rôle d'exécution de la fonction, ajoutez la file d'attente de lettres mortes à votre fonction. Vous pouvez configurer plusieurs fonctions pour envoyer des événements à la même cible.

Configuration d'une file d'attente de lettres mortes

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Asynchronous invocation (Appel asynchrone), choisissez Edit (Modifier).
4. Définissez DLQ resource (Ressource DLQ) sur Amazon SQS ou Amazon SNS.
5. Choisissez la file d'attente ou la rubrique cible.
6. Choisissez Save.

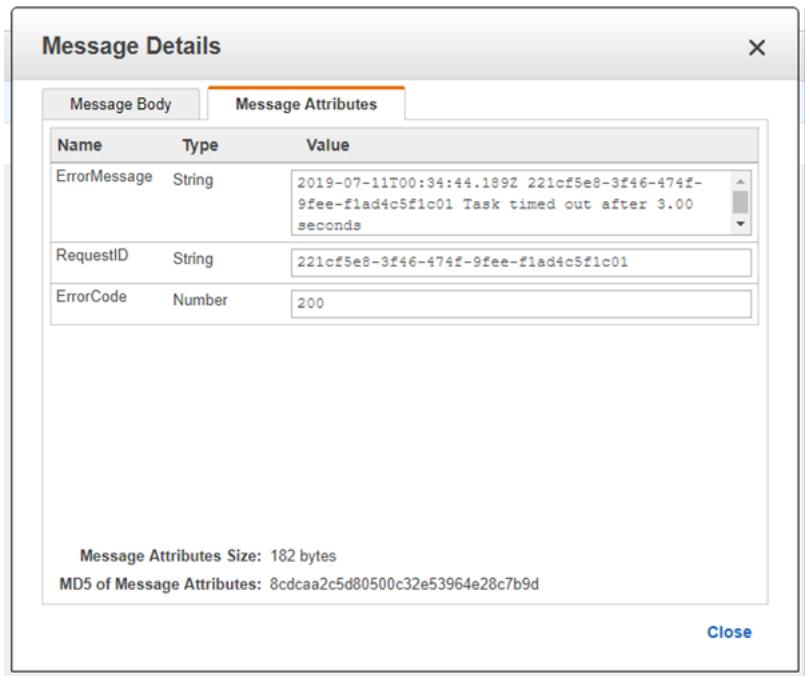
Pour configurer une file d'attente de lettres mortes avec la AWS CLI, utilisez la commande `update-function-configuration`.

```
$ aws lambda update-function-configuration --function-name my-function \
--dead-letter-config TargetArn=arn:aws:sns:us-east-2:123456789012:my-topic
```

Lambda envoie l'événement à la file d'attente de lettres mortes en l'état, avec des informations supplémentaires dans les attributs. Vous pouvez utiliser ces informations pour identifier l'erreur que la fonction a renvoyée ou établir une corrélation entre l'événement et des journaux ou une trace AWS X-Ray.

Attributs de message de file d'attente de lettres mortes

- RequestID (Chaîne) – ID de la demande d'appel. Les ID de demande apparaissent dans les journaux de la fonction. Vous pouvez également utiliser le kit de développement logiciel X-Ray pour enregistrer l'ID de demande sur un attribut dans le suivi. Vous pouvez ensuite rechercher des traces par ID de demande dans la console X-Ray. Consultez l'[exemple de processeur d'erreur \(p. 308\)](#).
- ErrorCode (Nombre) – Le code d'état HTTP.
- ErrorMessage (Chaîne) – Le premier Ko du message d'erreur.



Si Lambda ne peut pas envoyer un message à la file d'attente de lettres mortes, il supprime l'événement et émet la métrique [DeadLetterErrors](#) (p. 445). Cela peut se produire en raison d'un manque d'autorisations ou si la taille totale du message est supérieure à la limite de la file d'attente cible ou rubrique. 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.

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 sur la fonction Lambda. Pour plus d'informations, consultez [Utilisation de AWS Lambda avec Amazon SQS](#) (p. 285).

Mappages de source d'événement AWS Lambda

Un mappage de source d'événement est une ressource AWS Lambda qui lit à partir d'une source d'événement et appelle une fonction Lambda. Vous pouvez utiliser des mappages de source d'événement pour traiter des éléments à partir d'un flux ou d'une file d'attente dans des services qui n'appellent pas les fonctions Lambda directement. Lambda fournit des mappages de source d'événement pour les services suivants.

Services à partir desquels Lambda lit les événements

- [Amazon Kinesis](#) (p. 239)
- [Amazon DynamoDB](#) (p. 208)
- [Amazon Simple Queue Service](#) (p. 285)

Un mappage de source d'événement utilise les autorisations du rôle d'exécution (p. 33) de la fonction pour lire et gérer des éléments dans la source de l'événement. Les autorisations, la structure des événements, les paramètres et le comportement d'interrogation varient selon la source de l'événement. Pour en savoir plus, consultez la rubrique liée pour le service que vous utilisez en tant que source d'événement.

Pour gérer les mappages de source d'événement avec l'AWS CLI ou les kits SDK AWS, utilisez les actions d'API suivantes :

- [CreateEventSourceMapping \(p. 498\)](#)
- [ListEventSourceMappings \(p. 575\)](#)
- [GetEventSourceMapping \(p. 534\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)
- [DeleteEventSourceMapping \(p. 515\)](#)

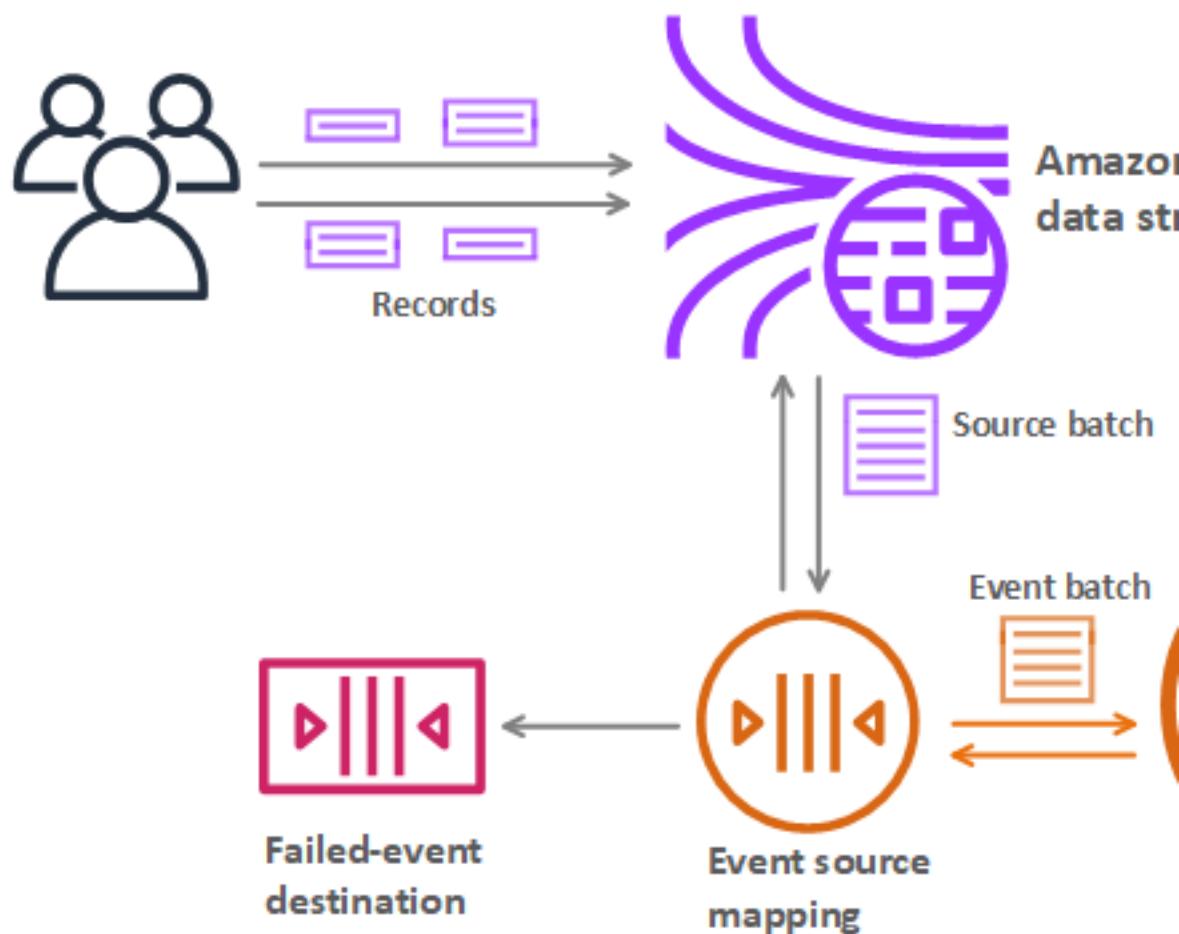
L'exemple suivant utilise l'AWS CLI pour mapper une fonction nommée `my-function` à un flux DynamoDB spécifié par son Amazon Resource Name (ARN), avec une taille de lot égale à 500.

```
$ aws lambda create-event-source-mapping --function-name my-function --batch-size 500 --starting-position LATEST \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/stream/2019-06-10T19:26:16.525
{
    "UUID": "14e0db71-5d35-4eb5-b481-8945cf9d10c2",
    "BatchSize": 500,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/stream/2019-06-10T19:26:16.525",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1560209851.963,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "User action",
    "DestinationConfig": {},
    "MaximumRecordAgeInSeconds": 604800,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 10000
}
```

Les mappages de source d'événement lisent les éléments d'un flux ou d'une file d'attente par lots. Ils incluent plusieurs éléments dans l'événement que votre fonction reçoit. Vous pouvez configurer la taille du lot envoyé à votre fonction par le mappage de source d'événement, jusqu'à un maximum qui varie en fonction du service. Le nombre d'éléments dans l'événement peut être inférieur à la taille de lot s'il y a pas suffisamment d'éléments disponibles, ou si le lot est trop volumineuse pour envoyer un événement et doit être divisé.

L'exemple suivant montre un mappage de source d'événement qui effectue une lecture à partir d'un flux Kinesis. Si un lot d'événements échoue à toutes les tentatives de traitement, le mappage de source d'événement envoie les détails du lot à une file d'attente SQS.

Event Source Mapping with Kinesis Stream



Le lot d'événements correspond à l'événement que Lambda envoie à la fonction. Il s'agit d'un lot d'enregistrements ou de messages compilés à partir des éléments que le mappage de source d'événement lit à partir d'un flux ou d'une file d'attente. La taille du lot et les autres paramètres ne s'appliquent qu'au lot d'événements.

Pour les flux, un mappage de source d'événement crée un itérateur pour chaque partition dans le flux et traite les éléments dans chaque partition dans l'ordre. Vous pouvez configurer le mappage de source d'événement pour lire uniquement les nouveaux éléments qui apparaissent dans le flux, ou pour démarrer avec des éléments plus anciens. Les éléments traités ne sont pas supprimés du flux et peuvent être traitées par d'autres fonctions ou d'autres consommateurs.

Par défaut, si votre fonction renvoie une erreur, l'ensemble du lot est retraité jusqu'à ce que la fonction aboutisse ou que les éléments du lot expirent. Pour s'assurer d'un traitement dans l'ordre, le traitement dans la partition concernée est mis en pause jusqu'à la résolution de l'erreur. Vous pouvez configurer le mappage de source d'événement pour ignorer les anciens événements, limiter le nombre de tentatives

ou traiter plusieurs lots en parallèle. Si vous traitez plusieurs lots en parallèle, le traitement dans l'ordre est toujours garanti pour chaque clé de partition, mais plusieurs clés de partition figurant dans la même partition sont traitées simultanément.

Vous pouvez également configurer le mappage de source d'événement pour envoyer un enregistrement d'appel à un autre service lorsqu'il rejette un lot d'événements. Lambda prend en charge les [destinations](#) (p. 96) suivantes pour les mappages de source d'événement.

- Amazon SQS – File d'attente SQS.
- Amazon SNS – Rubrique SNS.

L'enregistrement d'appel contient des détails sur le lot d'événements ayant échoué au format JSON.

L'exemple suivant illustre un enregistrement d'appel pour un flux Kinesis.

Example Enregistrement d'appel

```
{  
    "requestContext": {  
        "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",  
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
        "condition": "RetryAttemptsExhausted",  
        "approximateInvokeCount": 1  
    },  
    "responseContext": {  
        "statusCode": 200,  
        "executedVersion": "$LATEST",  
        "functionError": "Unhandled"  
    },  
    "version": "1.0",  
    "timestamp": "2019-11-14T00:38:06.021Z",  
    "KinesisBatchInfo": {  
        "shardId": "shardId-000000000001",  
        "startSequenceNumber": "49601189658422359378836298521827638475320189012309704722",  
        "endSequenceNumber": "49601189658422359378836298522902373528957594348623495186",  
        "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",  
        "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",  
        "batchSize": 500,  
        "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"  
    }  
}
```

Lambda prend également en charge le traitement dans l'ordre pour les [files d'attente FIFO](#) (p. 285), en effectuant un dimensionnement en fonction du nombre de groupes de messages actifs. Pour les files d'attente standard, les éléments ne sont pas nécessairement traités dans l'ordre. Lambda s'adapte pour traiter une file d'attente standard le plus rapidement possible. Les lots qui ont échoué sont renvoyés dans la file d'attente comme éléments individuels et peuvent être traités dans un autre regroupement que le lot original. Parfois, le mappage de source d'événement peut recevoir deux fois le même élément depuis la file d'attente, même sans aucune erreur de la fonction. Lambda supprime les éléments de la file d'attente lorsqu'ils sont traités avec succès. Vous pouvez configurer la file d'attente source pour envoyer des éléments vers une file d'attente de lettres mortes s'ils ne peuvent pas être traités.

Pour en savoir plus sur les services qui appellent les fonctions Lambda directement, consultez [Utilisation de AWS Lambda avec d'autres services](#) (p. 155).

Surveillance de l'état d'une fonction avec l'API Lambda

Lorsque vous créez ou mettez à jour une fonction, Lambda provisionne les ressources de calcul et de mise en réseau qui lui permettent de l'exécuter. Dans la plupart des cas, ce processus est très rapide, et votre fonction est prête à être appelée ou modifiée immédiatement.

Si vous configurez votre fonction pour vous connecter à un cloud privé virtuel (VPC), le processus peut prendre plus de temps. Lorsque vous connectez pour la première fois une fonction à un VPC, Lambda provisionne des interfaces réseau, ce qui prend environ une minute. Pour communiquer l'état actuel de votre fonction, Lambda inclut des champs supplémentaires dans le document de [configuration de fonction \(p. 673\)](#) qui est renvoyé par plusieurs actions d'API Lambda.

Lorsque vous créez une fonction, celle-ci est initialement à l'état Pending. Lorsque la fonction est prête à être appelée, l'état passe de Pending à Active. Lorsque l'état est Pending, les appels et autres actions d'API qui agissent sur la fonction renvoient une erreur. Si vous créez une automatisation autour de la création et de la mise à jour de fonctions, attendez que la fonction devienne active avant d'effectuer des actions supplémentaires qui agissent sur la fonction.

Vous pouvez utiliser l'API Lambda pour obtenir des informations sur l'état d'une fonction. Les informations d'état sont incluses dans le document [FunctionConfiguration \(p. 673\)](#) renvoyé par plusieurs actions d'API. Pour afficher l'état de la fonction avec l'AWS CLI, utilisez la commande `get-function-configuration`.

```
$ aws lambda get-function-configuration --function-name my-function
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs12.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "TracingConfig": {
        "Mode": "Active"
    },
    "State": "Pending",
    "StateReason": "The function is being created.",
    "StateReasonCode": "Creating",
    ...
}
```

`StateReason` et `StateReasonCode` contiennent des informations supplémentaires sur l'état lorsque celui-ci est différent de Active. Les opérations suivantes échouent lorsque la création de fonction est en attente :

- [Invoke \(p. 565\)](#)
- [UpdateFunctionCode \(p. 636\)](#)
- [UpdateFunctionConfiguration \(p. 643\)](#)
- [PublishVersion \(p. 601\)](#)

Lorsque vous mettez à jour la configuration d'une fonction, la mise à jour peut déclencher une opération asynchrone pour provisionner des ressources. Pendant que cette opération est en cours, vous pouvez appeler la fonction, mais d'autres opérations sur la fonction échouent. Les appels qui se produisent pendant la mise à jour sont exécutés par rapport à la configuration précédente. L'état de la fonction est Active, mais son `LastUpdateStatus` est InProgress.

Example Configuration de fonction – Connexion à un VPC

```
{
```

```
"FunctionName": "my-function",
"FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
"Runtime": "nodejs12.x",
"VpcConfig": {
    "SubnetIds": [
        "subnet-071f712345678e7c8",
        "subnet-07fd123456788a036",
        "subnet-0804f77612345cacf"
    ],
    "SecurityGroupIds": [
        "sg-085912345678492fb"
    ],
    "VpcId": "vpc-08e1234569e011e83"
},
"State": "Active",
"LastUpdateStatus": "InProgress",
...
}
```

Les opérations suivantes échouent lorsqu'une mise à jour asynchrone est en cours :

- [UpdateFunctionCode \(p. 636\)](#)
- [UpdateFunctionConfiguration \(p. 643\)](#)
- [PublishVersion \(p. 601\)](#)

D'autres opérations, y compris l'appel, fonctionnent pendant que les mises à jour sont en cours.

Par exemple, lorsque vous connectez votre fonction à un cloud privé virtuel (VPC), Lambda provisionne une interface réseau Elastic pour chaque sous-réseau. Ce processus peut laisser votre fonction dans un état en attente pendant une minute environ. Lambda récupère également les interfaces réseau qui ne sont pas utilisées, plaçant votre fonction dans un état `Inactive`. Lorsque la fonction est inactive, un appel la fait passer à l'état `Pending` pendant que l'accès réseau est restauré. L'appel qui déclenche la restauration ainsi que les autres appels émis pendant que l'opération est en attente échouent avec le code `ResourceNotReadyException`.

Si Lambda rencontre une erreur lors de la restauration de l'interface réseau d'une fonction, la fonction revient à l'état `Inactive`. L'appel suivant peut déclencher une autre tentative. Pour certaines erreurs de configuration, Lambda attend au moins 5 minutes avant de tenter de créer une autre interface réseau. Ces erreurs ont les valeurs `LastUpdateStatusReasonCode` suivantes :

- `InsufficientRolePermission` – Le rôle n'existe pas ou il lui manque des autorisations.
- `SubnetOutOfRangeAddresses` – Toutes les adresses IP d'un sous-réseau sont en cours d'utilisation.

Pour plus d'informations sur le fonctionnement des états avec la connectivité VPC, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC \(p. 81\)](#).

Dimensionnement d'une fonction AWS Lambda

La première fois que vous appelez votre fonction, AWS Lambda crée une instance de cette dernière et exécute sa méthode de gestionnaire pour traiter l'événement. Lorsque la fonction renvoie une réponse, elle reste active et attend de traiter des événements supplémentaires. Si vousappelez à nouveau la fonction pendant le traitement du premier événement, Lambda initialise une autre instance, et les deux événements sont traités simultanément. Avec l'entrée de plusieurs événements, Lambda les achemine vers des instances disponibles et crée de nouvelles instances en fonction des besoins. Lorsque le nombre de demandes diminue, Lambda arrête les instances non utilisées, afin de libérer la capacité de dimensionnement pour d'autres fonctions.

La simultanéité de votre fonction est le nombre d'instances répondant à des demandes à un moment donné. Pour un premier pic de trafic, la simultanéité cumulative de votre fonction dans une même région peut atteindre un niveau initial de 500 à 3 000, en fonction de la région.

Limites de simultanéité en rafale

- 3 000 – USA Ouest (Oregon), USA Est (Virginie du Nord), Europe (Irlande)
- 1 000 – Asie-Pacifique (Tokyo), Europe (Francfort)
- 500 – Autres régions.

Après le lancement initial en mode rafale, la simultanéité de votre fonction peut évoluer chaque minute avec jusqu'à 500 instances supplémentaires. Cela se poursuit jusqu'à ce que le nombre d'instances soit suffisant pour répondre à toutes les demandes ou lorsqu'une limite de simultanéité soit atteinte. Lorsque l'entrée des demandes est plus rapide que la capacité de dimensionnement de votre fonction ou lorsque votre fonction atteint la simultanéité maximale, des demandes supplémentaires échouent avec un code de blocage (code d'état 429).

L'exemple suivant montre une fonction qui traite un pic de trafic. Le dimensionnement de la fonction évolue à mesure que les appels augmentent de façon exponentielle. Elle lance une nouvelle instance pour toute demande qui ne peut pas être acheminée vers une instance disponible. Lorsque la limite de simultanéité en rafale est atteinte, la fonction commence une mise à l'échelle linéaire. Si cette simultanéité n'est pas suffisante pour répondre à toutes les demandes, les demandes supplémentaires sont soumises à une limitation et doivent faire l'objet d'une nouvelle tentative.

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

Légende

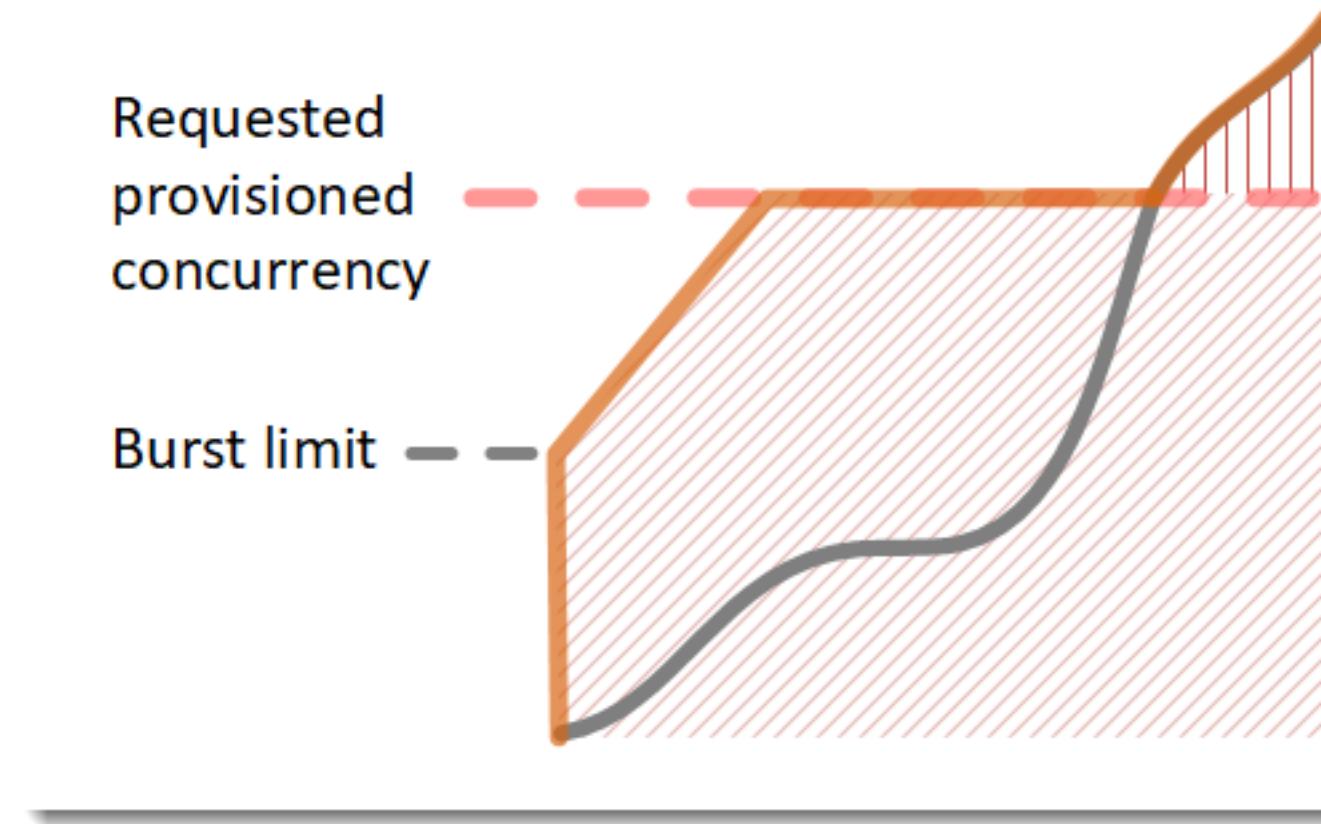
-  Instances de la fonction
-  Demandes ouvertes
-  Limitation possible

La fonction continue à évoluer jusqu'à ce que la limite de simultanéité du compte pour la région de la fonction soit atteinte. La fonction parvient à absorber la demande, les demandes diminuent et les instances inutilisées de la fonction sont arrêtées après un délai d'inactivité. Les instances inutilisées sont gelées en attente de demandes et n'entraînent aucun frais.

La limite de concurrence régionale commence à 1,000. Vous pouvez augmenter la limite en envoyant une demande dans la [console du Centre de support](#). Pour allouer la capacité par fonction, vous pouvez configurer les fonctions avec la [simultanéité réservée \(p. 61\)](#). La simultanéité réservée crée un pool qui ne peut être utilisé que par sa fonction et empêche sa fonction d'utiliser la simultanéité non réservée.

Lorsque votre fonction évolue, la première demande à laquelle chaque instance répond est affectée par le temps nécessaire pour charger et initialiser votre code. Si votre [code d'initialisation \(p. 19\)](#) prend beaucoup de temps, l'impact sur la moyenne et les centiles de la latence peut être significatif. Pour permettre à votre fonction d'évoluer sans fluctuations de latence, utilisez la [simultanéité allouée \(p. 61\)](#). L'exemple suivant montre une fonction avec simultanéité allouée qui traite un pic de trafic.

Function Scaling with Provisioned Concurrency



Légende

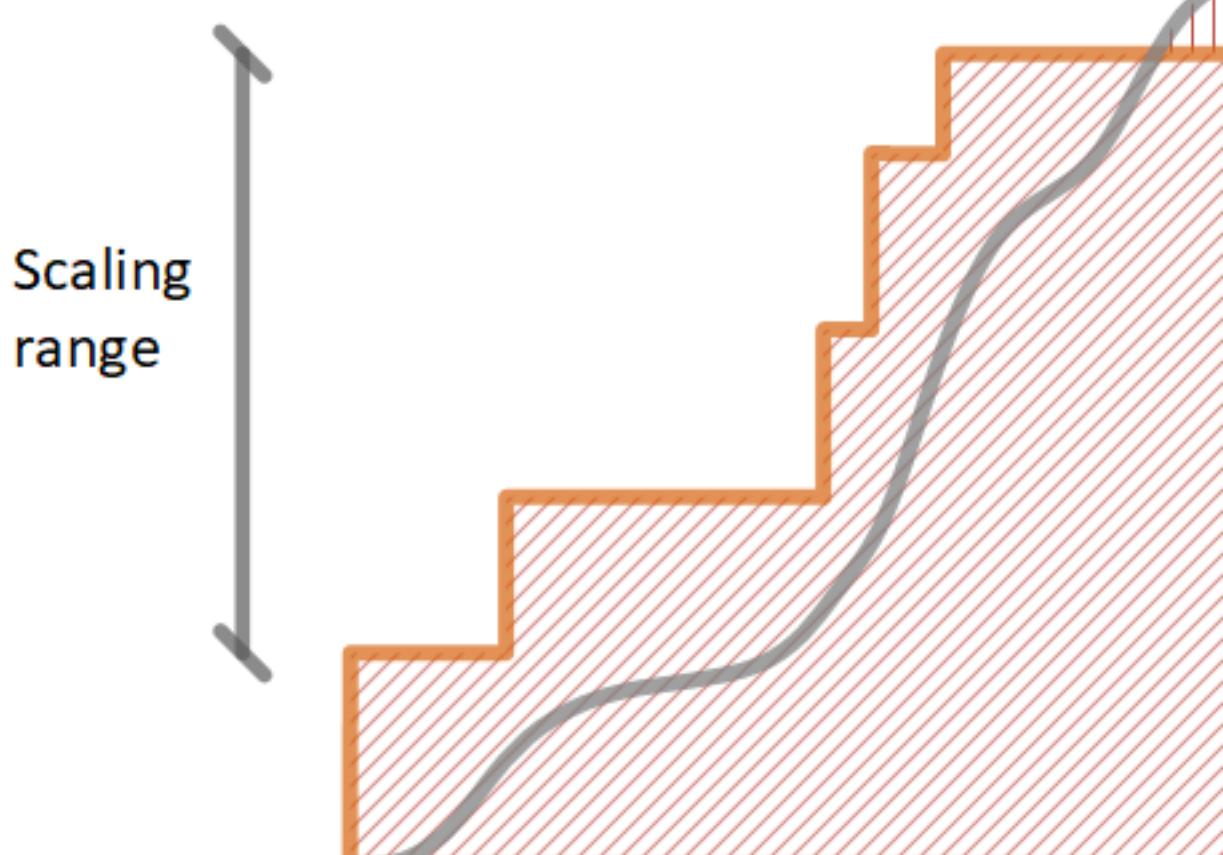
- Instances de la fonction
- Demandes ouvertes
- Simultanéité allouée
- Simultanéité standard

Lors de l'allocation de la simultanéité provisionnée, la fonction évolue avec le même comportement de rafale que la simultanéité standard. Après l'allocation, la simultanéité provisionnée répond aux demandes entrantes avec une latence très faible. Lorsque l'ensemble de la simultanéité allouée est utilisée, la fonction évolue normalement pour traiter les demandes supplémentaires.

Application Auto Scaling va encore plus loin en assurant le dimensionnement automatique pour la simultanéité provisionnée. Avec Application Auto Scaling, vous pouvez créer une stratégie de dimensionnement de suivi cible qui ajuste automatiquement les niveaux de simultanéité provisionnés, en fonction de la métrique d'utilisation émise par Lambda. [Utilisez l'API Application Auto Scaling \(p. 67\)](#) pour enregistrer un alias en tant que cible évolutive et créer une stratégie de dimensionnement.

Dans l'exemple suivant, une fonction évolue entre une valeur minimale et une valeur maximale de simultanéité allouée en fonction du niveau d'utilisation. Lorsque le nombre de demandes ouvertes augmente, Application Auto Scaling augmente la simultanéité allouée par paliers importants jusqu'à ce qu'elle atteigne la valeur maximale configurée. La fonction continue d'évoluer selon une simultanéité standard jusqu'à ce que l'utilisation commence à baisser. Lorsque l'utilisation est faible dans la durée, Application Auto Scaling diminue régulièrement la simultanéité allouée par paliers plus petits.

Autoscaling with Provisioned Concurrency



Légende

- Instances de la fonction
- Demandes ouvertes
- Simultanéité allouée
- Simultanéité standard

Lorsque vousappelez votrefonction defaçon asynchrone, avec unmappage de source d'événement ou avec unautre service AWS, lecomportement du dimensionnement varie. Par exemple, les mappages de source d'événement qui lisent àpartird'un flux sont limitées au nombre de partitions dans le flux. La capacité de dimensionnement inutilisée par une source d'événement peut être utilisée par d'autres clients et sources d'événements. Pour plus d'informations, consultez les rubriques suivantes.

- [Appel asynchrone \(p. 93\)](#)
- [Mappages de source d'événement AWS Lambda \(p. 101\)](#)
- [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#)
- [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#)

Vous pouvez surveiller les niveaux de simultanéité dans votre compte avec les métriques suivantes :

Métriques de simultanéité

- `ConcurrentExecutions`
- `UnreservedConcurrentExecutions`
- `ProvisionedConcurrentExecutions`
- `ProvisionedConcurrencyInvocations`
- `ProvisionedConcurrencySpilloverInvocations`
- `ProvisionedConcurrencyUtilization`

Pour de plus amples informations, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

Gestion des erreurs et tentatives automatiques dans AWS Lambda

Lorsque vousappelez unefonction, deux types d'erreur peuvent se produire. Les erreurs d'invocation surviennent lorsque la demande d'invocation est rejetée avant sa réception par votrefonction. Les erreurs de fonction se produisent lorsque lecode de votrefonction ou [l'exécution \(p. 122\)](#) renvoie une erreur. En fonction du type d'erreur, du type d'appel et du client ou du service qui appelle lafonction, le comportement de tentative et lastratégie de gestion des erreurs varient.

Desproblèmes avec la demande, l'appelant ou lecompte peuvent entraîner deserreurs d'appel. Les erreurs d'appel incluent un type d'erreur et un code d'état dans la réponse qui indiquent la cause de l'erreur.

Erreurs d'appel courantes

- Demande – L'événement demande est trop grand ou n'est pas un JSON valable, lafonction n'existe pas ou une valeur de paramètre est de type incorrect.

- Appelant – L'utilisateur ou le service n'a pas l'autorisation d'appeler la fonction.
- Compte – Le nombre maximum d'instances de fonctions est déjà en cours d'exécution ou les demandes sont réalisées trop rapidement.

Les clients comme la AWS CLI et le kit SDK AWS réessayeront les expirations du client, les erreurs de blocage (429) et les autres erreurs qui ne sont pas causées par une demande erronée (série 500). Pour obtenir la liste complète des erreurs d'appel, consultez [Invoke \(p. 565\)](#).

Les erreurs de fonction se produisent lorsque le code de votre fonction ou l'exécution qu'elle utilise renvoie une erreur.

Erreurs de fonction courantes

- Fonction – Le code de votre fonction lance une exception ou renvoie un objet d'erreur.
- Runtime – Le runtime a mis fin à votre fonction parce qu'elle arrivait à expiration, après avoir détecté une erreur de syntaxe ou parce qu'il n'a pas réussi à rassembler l'objet de réponse dans JSON. La fonction est sortie avec un code d'erreur.

Contrairement aux erreurs d'appel, les erreurs de fonction ne provoquent le renvoi par Lambda d'un code d'état de la série 400 ou 500. Si la fonction renvoie une erreur, Lambda l'indique en incluant un en-tête nommé `X-Amz-Function-Error`, et une réponse au format JSON avec le message d'erreur et d'autres détails. Pour obtenir des exemples d'erreurs de fonction dans chaque langage, consultez les rubriques suivantes.

- [Erreurs de fonction AWS Lambda dans Node.js \(p. 322\)](#)
- [Erreurs de fonction AWS Lambda dans Python \(p. 341\)](#)
- [Erreurs de fonctions AWS Lambda en Ruby \(p. 356\)](#)
- [Erreurs de fonction AWS Lambda dans Java \(p. 384\)](#)
- [Erreurs de fonction AWS Lambda dans Go \(p. 408\)](#)
- [Erreurs de fonction AWS Lambda en C# \(p. 428\)](#)
- [Erreurs de fonction AWS Lambda dans PowerShell \(p. 443\)](#)

Lorsque vous appelez une fonction directement, vous déterminez la stratégie de gestion des erreurs. Vous pouvez réessayer, envoyez l'événement à une file d'attente pour débogage ou ignorer l'erreur. Le code de votre fonction peut s'exécuter entièrement, partiellement ou pas du tout. Si vous recommencez, assurez-vous que le code de votre fonction peut gérer le même événement plusieurs fois, sans provoquer des transactions en double ou d'autres effets secondaires.

Lorsque vousappelez une fonction indirectement, vous devez connaître le comportement de la nouvelle tentative de l'appelant et tout service que la demande rencontre sur le chemin. Cela inclut les scénarios suivants.

- Appel asynchrone – Lambda réessaie deux fois les erreurs de fonction. Si la fonction ne dispose pas de la capacité suffisante pour gérer toutes les demandes entrantes, des événements peuvent attendre dans la file d'attente pendant des heures ou des jours avant d'être envoyés à la fonction. Vous pouvez configurer une file d'attente de lettres mortes sur la fonction pour capturer les événements qui n'ont pas été traitées avec succès. Pour en savoir plus, consultez [Appel asynchrone \(p. 93\)](#).
- Mappages de source d'événement – Les mappages de source d'événement qui lisent à partir des flux réessaient la totalité du lot d'événements. Les erreurs répétées bloquent le traitement de la partition concernée jusqu'à la résolution de l'erreur ou l'expiration des éléments. Pour détecter des partitions bloquées, vous pouvez surveiller la métrique [Âge de l'itérateur \(p. 445\)](#).

Pour les mappages de source d'événement qui lisent à partir d'une file d'attente, vous devez déterminer l'intervalle entre les nouvelles tentatives et la destination pour les événements ayant échoué, en

configurant le délai de visibilité et la stratégie de redirection dans la file d'attente source. Pour en savoir plus, consultez [Mappages de source d'événement AWS Lambda \(p. 101\)](#), ainsi que les rubriques spécifiques au service dans [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

- Services AWS – Les services AWS peuvent appeler votre fonction de façon [synchrone \(p. 92\)](#) ou [asynchrone](#). Dans le cas des appels synchrones, le service décide s'il convient d'effectuer une nouvelle tentative. Les services tels que API Gateway et Elastic Load Balancing, avec requêtes proxy d'un utilisateur ou client en amont, peuvent également choisir de relayer la réponse d'erreur au demandeur.

Pour un appel asynchrone, le comportement est le même que lors de l'appel de la fonction de façon asynchrone. Pour en savoir plus, consultez les rubriques spécifiques aux services dans [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#) et la documentation concernant le service appelant.

- Autres comptes et clients – Lorsque vous accordez l'accès à d'autres comptes, vous pouvez utiliser des [stratégies basées sur les ressources \(p. 36\)](#) pour restreindre les services ou les ressources qu'ils peuvent configurer pour appeler votre fonction. Pour éviter la surcharge de votre fonction, envisagez de placer une couche d'API devant votre fonction avec [Amazon API Gateway \(p. 157\)](#).

Pour vous aider à gérer les erreurs dans les applications Lambda, Lambda s'intègre à des services tels que Amazon CloudWatch et AWS X-Ray. Vous pouvez utiliser une combinaison de journaux, de mesures, d'alarmes et de suivi, pour détecter rapidement et d'identifier les problèmes dans le code de votre fonction, de l'API ou d'autres ressources qui prennent en charge votre application. Pour plus d'informations, consultez [Surveillance et dépannage des applications Lambda \(p. 444\)](#).

Pour un exemple d'application qui utilise un abonnement CloudWatch Logs, un suivi X-Ray et une fonction Lambda pour détecter et traiter les erreurs, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Appel de fonctions Lambda avec AWS Mobile SDK pour Android

Vous pouvez appeler une fonction Lambda à partir d'une application mobile. Insérez de la logique métier dans les fonctions afin de séparer leur cycle de vie de développement de celui des clients frontaux, ce qui rend le développement et la maintenance des applications mobiles moins complexes. Avec Mobile SDK pour Android, vous [utilisez Amazon Cognito pour authentifier les utilisateurs et autoriser les demandes \(p. 114\)](#).

Lorsque vous appelez une fonction à partir d'une application mobile, vous choisissez la structure de l'événement, le [type d'appel \(p. 91\)](#) et le modèle d'autorisation. Vous pouvez utiliser des [alias \(p. 72\)](#) pour activer les mises à jour continues de votre code de fonction, mais dans le cas contraire, la fonction et l'application sont étroitement liées. Au fur et à mesure que vous ajoutez d'autres fonctions, vous pouvez créer une couche d'API afin de dissocier le code de votre fonction de vos clients frontaux et améliorer les performances.

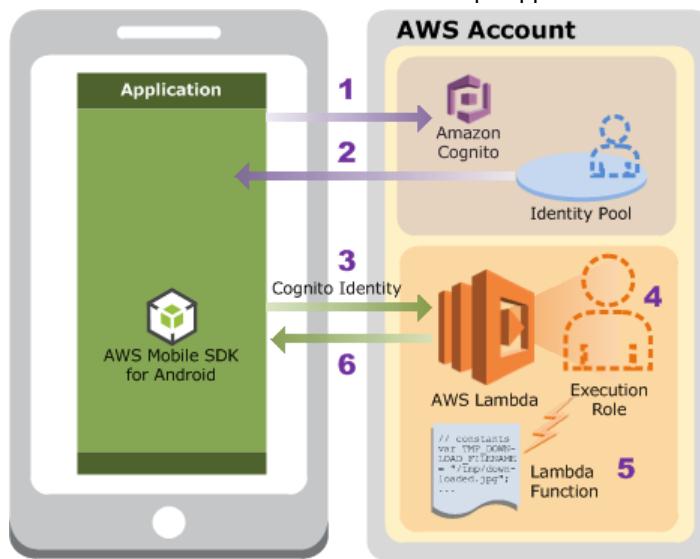
Pour créer une API web complète en fonctionnalités pour vos applications Web et mobiles, utilisez Amazon API Gateway. Avec API Gateway, vous pouvez ajouter des mécanismes d'autorisation personnalisés, limiter les demandes et mettre en cache les résultats pour l'ensemble de vos fonctions. Pour de plus amples informations, veuillez consulter [Utilisation de AWS Lambda avec Amazon API Gateway \(p. 157\)](#).

Rubriques

- [Didacticiel : Utilisation de AWS Lambda avec Mobile SDK pour Android \(p. 114\)](#)
- [Exemple de code de fonction \(p. 119\)](#)

Didacticiel : Utilisation de AWS Lambda avec Mobile SDK pour Android

Dans ce didacticiel, vous allez créer une simple application mobile Android qui utilise Amazon Cognito pour obtenir des informations d'identification et qui appelle une fonction Lambda.



L'application mobile récupère les informations d'identification AWS à partir d'un pool d'identités Amazon Cognito et les utilise pour appeler une fonction Lambda avec un événement qui contient les données de la demande. La fonction traite la demande et renvoie une réponse à l'élément frontal.

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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 \(p. 119\)](#).

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 nodejs12.x \  
--role arn:aws:iam::123456789012:role/lambda-android-role
```

Tester 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 outputfile.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 Lambda 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 {

    /**
     * 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.

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 \(p. 120\)](#)
- [Java \(p. 120\)](#)

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);
}
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

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 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);  
}  
}
```

Dépendances

- `aws-lambda-java-core`

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

Environnements d'exécution 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 sous-jacent fournit des bibliothèques et des [variables d'environnement \(p. 55\)](#) supplémentaires auxquelles vous pouvez accéder depuis le code de votre fonction.

Amazon Linux

- Image – [amzn-ami-hvm-2018.03.0.20181129-x86_64-gp2](#)
- Noyau Linux – 4.14.171-105.231.amzn1.x86_64

Amazon Linux 2

- Image – Personnalisé
- Noyau Linux – 4.14.165-102.205.amzn2.x86_64

Lorsque votre fonction est appelée, Lambda tente de réutiliser l'environnement d'exécution à partir d'un appel antérieur, 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 \(p. 124\)](#) 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 \(p. 124\)](#) lorsque la version arrive en fin de vie.

Environnements d'exécution Node.js

| Nom | Identifiant | Kit AWS SDK for JavaScript | Système d'exploitation |
|------------|-------------------------|----------------------------|------------------------|
| Node.js 12 | <code>nodejs12.x</code> | 2.631.0 | Amazon Linux 2 |
| Node.js 10 | <code>nodejs10.x</code> | 2.631.0 | Amazon Linux 2 |

Environnements d'exécution Python

| Nom | Identifiant | Kit SDK AWS pour Python | Système d'exploitation |
|------------|------------------------|-----------------------------------|------------------------|
| Python 3.8 | <code>python3.8</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux 2 |
| Python 3.7 | <code>python3.7</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |
| Python 3.6 | <code>python3.6</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |

| Nom | Identifiant | Kit SDK AWS pour Python | Système d'exploitation |
|------------|------------------------|-----------------------------------|------------------------|
| Python 2.7 | <code>python2.7</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |

Environnements d'exécution Ruby

| Nom | Identifiant | Kit AWS SDK for Ruby | Système d'exploitation |
|----------|----------------------|----------------------|------------------------|
| Ruby 2.7 | <code>ruby2.7</code> | 3.0.1 | Amazon Linux 2 |
| Ruby 2.5 | <code>ruby2.5</code> | 3.0.1 | Amazon Linux |

Environnements d'exécution Java

| Nom | Identificateur | JDK | Système d'exploitation |
|---------|---------------------|--------------------|------------------------|
| Java 11 | <code>java11</code> | amazon-corretto-11 | Amazon Linux 2 |
| Java 8 | <code>java8</code> | java-1.8.0-openjdk | Amazon Linux |

Environnements d'exécution Go

| Nom | Identifiant | Système d'exploitation |
|--------|--------------------|------------------------|
| Go 1.x | <code>go1.x</code> | Amazon Linux |

Environnements d'exécution .NET

| Nom | Identifiant | Système d'exploitation |
|---------------|----------------------------|------------------------|
| .NET Core 3.1 | <code>dotnetcore3.1</code> | Amazon Linux 2 |
| .NET Core 2.1 | <code>dotnetcore2.1</code> | Amazon Linux |

Pour utiliser d'autres langages dans Lambda, vous pouvez implémenter un [runtime personnalisé \(p. 126\)](#). L'environnement d'exécution de Lambda fournit une [interface de runtime \(p. 128\)](#) 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 \(p. 76\)](#).

Environnement d'exécution personnalisé

| Nom | Identifiant | Système d'exploitation |
|-------------------------|-----------------------|------------------------|
| Exécution personnalisée | <code>provided</code> | Amazon Linux |

Rubriques

- [Contexte d'exécution AWS Lambda \(p. 124\)](#)
- [Stratégie de prise en charge des environnements d'exécution \(p. 124\)](#)
- [Environnements d'exécution AWS Lambda personnalisés \(p. 126\)](#)

- [Interface de runtime AWS Lambda \(p. 128\)](#)
- [Didacticiel – Publication d'un runtime personnalisé \(p. 130\)](#)

Contexte d'exécution 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 :

- Les objets déclarés en dehors de la méthode de gestionnaire de la fonction restent initialisés, ce qui fournit 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 512 MB d'espace disque supplémentaire dans le répertoire `/tmp`. Le contenu du répertoire est préservé lorsque le contexte d'exécution est gelé, fourissant 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 \(p. 30\)](#).
- 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 des environnements d'exécution

[Les exécutions AWS Lambda \(p. 122\)](#) combinent un système d'exploitation, un langage de programmation et des bibliothèques de logiciels qui font l'objet d'une maintenance et de mises à jour de

sécurité. Lorsqu'un composant d'une exécution n'est plus pris en charge pour les mises à jour de sécurité, Lambda rend obsolète l'exécution.

L'obsolescence se produit en deux phases. Pendant la première phase, vous ne pouvez plus créer de fonctions qui utilisent l'exécution obsolète. Pendant au moins 30 jours, vous pouvez continuer à mettre à jour les fonctions existantes qui utilisent l'exécution obsolète. Après cette période, la création et les mises à jour de la fonction sont définitivement désactivées. Toutefois, la fonction reste disponible pour traiter les événements d'appels.

Note

Python 2.7 est en fin de vie depuis le 1er janvier 2020. Cependant, l'environnement d'exécution Python 2.7 est toujours pris en charge et son obsolescence n'est pas planifiée pour le moment. Pour de plus amples informations, veuillez consulter l'article de blog [Continued support for Python 2.7 on AWS Lambda](#).

Les environnements d'exécution suivants sont devenus obsolètes :

Environnements d'exécution obsolètes

| Nom | Identifiant | Système d'exploitation | Date d'achèvement de l'obsolescence |
|------------------|----------------|------------------------|-------------------------------------|
| .NET Core 1.0 | dotnetcore1.0 | Amazon Linux | 30 juillet 2019 |
| .NET Core 2.0 | dotnetcore2.0 | Amazon Linux | 30 mai 2019 |
| Node.js 0.10 | nodejs | Amazon Linux | 31 octobre 2016 |
| Node.js 4.3 | nodejs4.3 | Amazon Linux | 6 mars 2020 |
| Node.js 4.3 edge | nodejs4.3-edge | Amazon Linux | 30 avril 2019 |
| Node.js 6.10 | nodejs6.10 | Amazon Linux | 12 août 2019 |
| Node.js 8.10 | nodejs8.10 | Amazon Linux | 6 mars 2020 |

Dans la plupart des cas, la date de fin de vie d'une version du langage ou système d'exploitation est connue largement à l'avance. Si vous avez des fonctions qui s'exécutent sur un runtime et qui deviendront obsolètes au cours des 60 prochains jours, Lambda vous informe par e-mail que vous devez vous préparer en migrant votre fonction vers une exécution prise en charge. Dans certains cas, tels que les problèmes de sécurité qui nécessitent une mise à jour irréversible ou un logiciel qui ne prend pas en charge le LTS (long-term support), une notification préalable n'est pas toujours possible.

Stratégies de prise en charge des langages et des infrastructures

- Node.js – github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java – www.oracle.com et aws.amazon.com/corretto
- Go – golang.org
- .NET Core – dotnet.microsoft.com

Une fois qu'une exécution est obsolète, Lambda peut la supprimer complètement à tout moment en désactivant l'appel. Les exécutions obsolètes ne sont éligibles ni pour les mises à jour de sécurité ni pour l'assistance technique. Avant la mise hors service d'une exécution, Lambda envoie des notifications supplémentaires aux clients concernés. Pour l'instant, aucune mise hors service d'exécution n'est prévue.

Environnements d'exécution 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 \(p. 122\)](#). 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 \(p. 130\)](#). 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 \(p. 126\)](#)
- [Création d'un runtime personnalisé \(p. 126\)](#)

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 \(p. 76\)](#).

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\ncd $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 \(p. 124\)](#) 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.

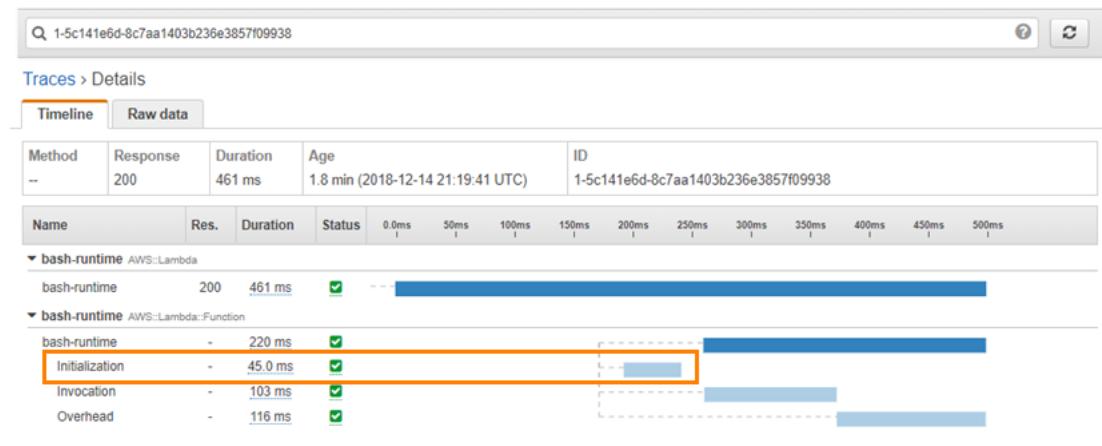
Pour obtenir une liste complète des variables disponibles, consultez [Variables d'environnement d'exécution \(p. 57\)](#).

- 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 \(p. 130\)](#) et quittez immédiatement.

L'initialisation est comptabilisée dans le délai d'attente et la durée d'exécution facturés. Lorsqu'une exécution déclenche l'initialisation d'une nouvelle instance de votre fonction, vous pouvez voir le temps d'initialisation dans les journaux et [le suivi AWS X-Ray \(p. 296\)](#).

Example log

```
REPORT RequestId: f8ac1208... Init Duration: 48.26 ms Duration: 237.17 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 26 MB
```



Pendant son exécution, le runtime utilise l'[interface de runtime Lambda \(p. 128\)](#) 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. Dans votre code d'exécution, effectuez les étapes suivantes dans l'ordre.

Traitement des tâches

- Obtenir un événement – Appelez l'API [prochain appel \(p. 128\)](#) 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` localement avec la même valeur. Le kit X-Ray SDK utilise cette valeur pour relier des données de suivi entre les services.
- 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 \(p. 129\)](#) pour afficher la réponse du gestionnaire.
- Gérer les erreurs – Si une erreur se produit,appelez l'API [erreur d'appel \(p. 129\)](#).
- 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 de procédure, consultez [Didacticiel – Publication d'un runtime personnalisé \(p. 130\)](#).

Interface de runtime AWS Lambda

AWS Lambda fournit une API HTTP pour les [runtimes personnalisés \(p. 126\)](#) 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 \(p. 122\)](#).

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 Demande

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Ressources

- [Appel suivant \(p. 128\)](#)
- [Réponse d'appel \(p. 129\)](#)
- [Erreur d'appel \(p. 129\)](#)
- [Erreur d'initialisation \(p. 130\)](#)

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.

Appelez `/runtime/invocation/next` pour obtenir l'événement d'appel et transmettez-le au gestionnaire de fonctions pour le traitement. Ne définissez pas un délai d'expiration sur l'appel `GET`. Entre le moment où Lambda amorce le runtime et celui où le runtime a un événement à renvoyer, le processus d'exécution peut être bloqué pour plusieurs secondes.

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 `$_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

Envie 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" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

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 d'initialisation

```
ERROR={"\\"errorMessage\"\": \\"Failed to load function.\\"", \\"errorType\"\":  
    \\"InvalidFunctionException\\\"}"  
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR"  
--header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

```
this is output
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Vous avez besoin d'un rôle IAM pour créer une fonction Lambda. Ce rôle doit avoir l'autorisation d'envoyer des journaux à CloudWatch Logs et d'accéder aux services AWS utilisés par votre fonction. Si vous n'avez pas de rôle pour le développement de fonctions, créez-en un.

Pour créer un rôle d'exécution

1. Ouvrez la page [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.

Création d'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="$(mktemp)"
    # Get an event. The HTTP request will block until one is received
    EVENT_DATA=$(curl -sS -L "$HEADERS" -X GET "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

    # Extract request ID by scraping response headers received above
    REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d '[:space:]' | cut -d: -f2)

    # Execute the handler function from the script
```

```
RESPONSE=$(($echo "$_HANDLER" | cut -d. -f2) "$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 \(p. 126\)](#).

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 file:///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",  
    "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+D3H1QbDARUVPrDwlC7bssPxyS0qt1QZqusE=", 
        "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.

Mettre à jour 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"}'
```

Mettre à jour le 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
fileb:/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
```

Partager 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. Ce 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. [AWS SAM \(p. 28\)](#) étend AWS CloudFormation avec une syntaxe simplifiée axée sur le développement d'applications Lambda.

L'[AWS CLI \(p. 28\)](#) et l'[Interface de ligne de commande SAM \(p. 29\)](#) 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 \(p. 136\)](#)
- [Création d'une application avec distribution continue dans la console Lambda \(p. 139\)](#)
- [Déploiements propagés pour les fonctions Lambda \(p. 148\)](#)
- [Types d'applications Lambda courantes et cas d'utilisation \(p. 149\)](#)
- [Bonnes pratiques d'utilisation des fonctions AWS Lambda \(p. 151\)](#)

Gestion des applications dans la console AWS Lambda

La console AWS Lambda vous permet de surveiller et de gérer vos [applications Lambda \(p. 136\)](#). 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. Ouvrez la [page Applications](#) de la console Lambda.

2. 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. Ouvrez la [page Applications](#) de la console Lambda.
2. 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",

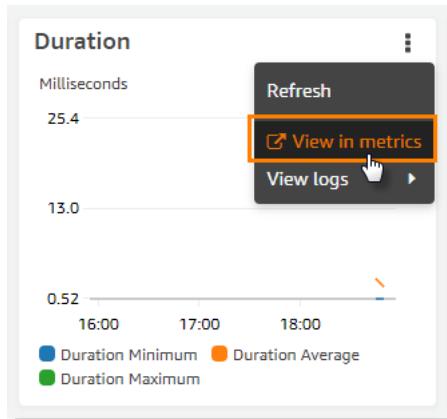
```

```
"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. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez une application qui possède le tableau de bord standard.
3. Choisissez Surveillance.
4. Sur un widget quelconque, choisissez Afficher dans les métriques dans le menu déroulant.



5. 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.

Création d'une application avec distribution continue dans la console Lambda

Vous pouvez utiliser la console Lambda pour créer une application avec un pipeline de distribution continue intégré. Avec une distribution continue, chaque modification que vous transmettez à votre référentiel de contrôle source déclenche un pipeline qui génère et déploie automatiquement votre application. La console Lambda fournit des projets de démarrage pour les types d'applications courants avec un exemple de code Node.js et des modèles qui créent des ressources de support.

Dans ce didacticiel, vous allez créer les ressources suivantes.

- Application – Une fonction Node.js Lambda, la spécification de génération et un modèle Modèle d'application sans serveur AWS (AWS SAM).
- Pipeline – Un pipeline AWS CodePipeline reliant les autres ressources pour permettre une distribution continue.
- Référentiel – Un référentiel Git dans AWS CodeCommit. Lorsque vous envoyez une modification, le pipeline copie le code source dans un compartiment Amazon S3 et le transmet au projet de génération.
- Déclencheur – Règle Amazon CloudWatch Events qui surveille la branche principale du référentiel et déclenche le pipeline.
- Projet de génération – Une génération AWS CodeBuild qui obtient le code source du pipeline et conditionne l'application en package. Le code source inclut une spécification de génération avec des commandes qui installent des dépendances et préparent le modèle d'application à déployer.
- Configuration du déploiement – L'étape de déploiement du pipeline définit un ensemble d'actions qui prennent le modèle AWS SAM traité à partir de la sortie de génération et déploient la nouvelle version avec AWS CloudFormation.
- Compartiment – Un compartiment Amazon Simple Storage Service (Amazon S3) pour le stockage d'artefacts de déploiement.
- Rôles – Les étapes source, génération et déploiement du pipeline ont des rôles IAM qui leur permettent de gérer les ressources AWS. La fonction de l'application a un [rôle d'exécution \(p. 33\)](#) qui lui permet de télécharger des journaux et qui peut être étendue pour accéder à d'autres services.

Vos ressources d'application et de pipeline sont définies dans des modèles AWS CloudFormation que vous pouvez personnaliser et étendre. Votre référentiel d'applications inclut un modèle que vous pouvez modifier pour ajouter des tables Amazon DynamoDB, une API Amazon API Gateway et d'autres ressources d'application. Le pipeline de distribution continue est défini dans un modèle distinct en dehors du contrôle de la source et possède sa propre pile.

Le pipeline mappe une seule branche dans un référentiel vers une seule pile d'applications. Vous pouvez créer d'autres pipelines pour ajouter des environnements pour d'autres branches dans le même référentiel. Vous pouvez également ajouter des étapes à votre pipeline pour le test, l'étape intermédiaire et des approbations manuelles. Pour plus d'informations sur AWS CodePipeline, consultez [Qu'est-ce que AWS CodePipeline ?](#).

Sections

- [Prérequis \(p. 140\)](#)
- [Création d'une application \(p. 140\)](#)
- [Appel de la fonction \(p. 141\)](#)
- [Ajouter une ressource AWS \(p. 142\)](#)
- [Mettre à jour la limite des autorisations \(p. 144\)](#)
- [Mettre à jour le code de la fonction \(p. 144\)](#)

- [Étapes suivantes \(p. 146\)](#)
- [Dépannage \(p. 146\)](#)
- [Nettoyage \(p. 147\)](#)

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Ce didacticiel utilise CodeCommit pour le contrôle de la source. Pour configurer votre ordinateur local pour qu'il puisse accéder au code d'application et le mettre à jour, consultez [Configuration](#) dans le AWS CodeCommit Guide de l'utilisateur.

Création d'une application

Choisissez une application dans la console Lambda. Dans Lambda, une application est une pile AWS CloudFormation avec une fonction Lambda et un certain nombre de ressources associées. Dans ce didacticiel, vous créez une application qui présente une fonction et son rôle d'exécution.

Pour créer une application

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez Create application.
3. Choisissez Créer à partir de zéro.
4. Configurez les paramètres de l'application.
 - Nom de l'application – **my-app**.
 - Description de l'application – **my application**.
 - Environnement d'exécution – Node.js 10.x.
 - Service de contrôle de la source – CodeCommit.
 - Nom du référentiel – **my-app-repo**.
 - Autorisations – Create roles and permissions boundary (Créer des rôles et des limites d'autorisations).
5. Sélectionnez Créer.

Lambda crée le pipeline et les ressources associées et valide l'exemple de code d'application dans le référentiel Git. Au fur et à mesure que les ressources sont créées, elles apparaissent sur la page d'aperçu.

| Logical ID | Physical ID | Type | Last modified |
|---------------------------|---|-----------------------|----------------|
| CodeCommitRepo | b4976f9b-9399-45f1-bd21-e7a9a315981d | CodeCommit Repository | 9 seconds ago |
| PermissionsBoundaryPolicy | arn:aws:iam::123456789012:policy/my-app-us-east-2-PermissionsBoundary | IAM ManagedPolicy | 13 seconds ago |
| S3Bucket | aws-us-east-2-123456789012-my-app-pipe | S3 Bucket | 13 seconds ago |

La pile d'infrastructure contient le référentiel, le projet de génération et d'autres ressources qui se combinent pour former un pipeline de livraison continue. Lorsque le déploiement de cette pile est terminé, elle déploie à son tour la pile d'applications qui contient la fonction et le rôle d'exécution. Il s'agit des ressources d'application qui apparaissent sous Ressources.

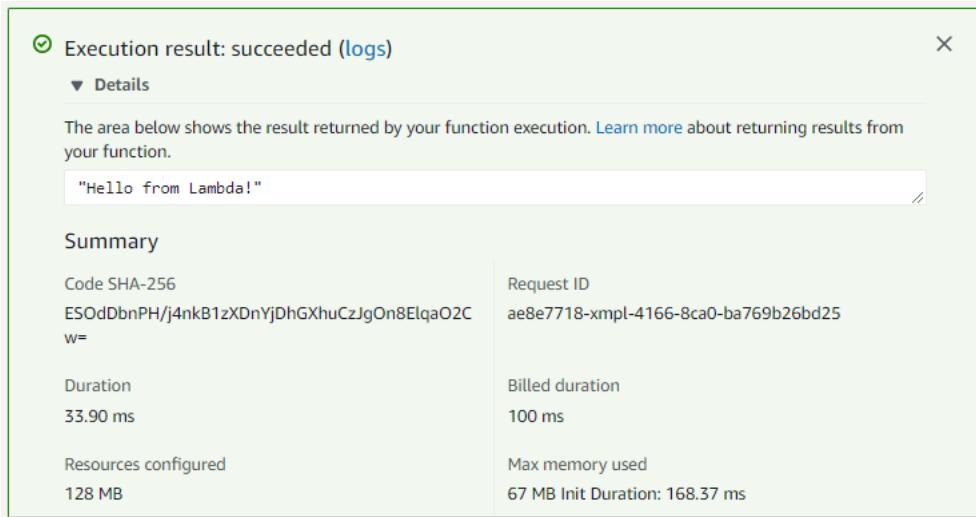
Appel de la fonction

Lorsque le processus de déploiement est terminé, appelez la fonction à partir de la console Lambda.

Appeler la fonction de l'application

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez my-app.
3. Sous Resouces (Ressources), choisissez helloFromLambdaFunction.
4. Sélectionnez Test.
5. Configurez un événement de test.
 - Nom de l'événement – **event**
 - Corps de texte – **{ }**
6. Sélectionnez Create.
7. Sélectionnez Test.

La console Lambda exécute votre fonction et affiche le résultat. Développez la section Details (Détails) sous le résultat pour afficher les détails de sortie et d'exécution.



Ajouter une ressource AWS

À l'étape précédente, la console Lambda a créé un référentiel Git qui contient du code de fonction, un modèle et une spécification de génération. Vous pouvez ajouter des ressources à votre application en modifiant le modèle et en transmettant les modifications au référentiel. Pour obtenir une copie de l'application sur votre ordinateur local, clonez le référentiel.

Cloner le référentiel de projet

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez my-app.
3. Choisissez Code.
4. Sous Repository details (Détails du référentiel), copiez l'URI du référentiel HTTP ou SSH, selon le mode d'authentification que vous avez défini au cours de la [configuration \(p. 140\)](#).
5. Pour cloner le référentiel, utilisez la commande `git clone`.

```
~$ git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/my-app-repo
```

Pour ajouter une table DynamoDB à l'application, définissez une ressource `AWS::Serverless::SimpleTable` dans le modèle.

Ajouter une table DynamoDB

1. Ouvrez `template.yml` dans un éditeur de texte.
2. Ajoutez une ressource de table, une variable d'environnement qui transmet le nom de la table à la fonction et une stratégie d'autorisations qui permet à la gérer.

Example template.yml - ressources

```
...
Ressources:
  ddbTable:
    Type: AWS::Serverless::SimpleTable
    Properties:
```

```

PrimaryKey:
  Name: id
  Type: String
ProvisionedThroughput:
  ReadCapacityUnits: 1
  WriteCapacityUnits: 1
helloFromLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ./
    Handler: src/handlers/hello-from-lambda.helloFromLambdaHandler
    Runtime: nodejs10.x
    MemorySize: 128
    Timeout: 60
    Description: A Lambda function that returns a static string.
Environment:
  Variables:
    DDB_TABLE: !Ref ddbTable
Policies:
  - DynamoDBCrudPolicy:
      TableName: !Ref ddbTable
  - AWSLambdaBasicExecutionRole

```

3. Valider et transmettre le changement.

```

~/my-app-repo$ git commit -am "Add DynamoDB table"
~/my-app-repo$ git push

```

Lorsque vous transmettez une modification, elle déclenche le pipeline de l'application. Utilisez l'onglet Deployments (Déploiements) de l'écran de l'application pour effectuer le suivi de la modification au fur et à mesure qu'elle circule dans le pipeline. Une fois le déploiement terminé, passez à l'étape suivante.

| Deployment | Resource type | Last updated time | Status |
|---------------|--------------------|-------------------|--|
| 6 minutes ago | Lambda application | 5 minutes ago | ✓ Update complete |
| 2 hours ago | Lambda application | 2 hours ago | ✓ Create complete |

Mettre à jour la limite des autorisations

L'exemple d'application applique une limite d'autorisations au rôle d'exécution de sa fonction. La limite des autorisations limite celles que vous pouvez ajouter au rôle de la fonction. Sans la limite, les utilisateurs disposant d'un accès en écriture au référentiel de projet pourraient modifier le modèle de projet, pour donner à la fonction l'autorisation d'accéder aux ressources et services en dehors de la portée de l'exemple d'application.

Pour que la fonction utilise l'autorisation DynamoDB que vous avez ajoutée à son rôle d'exécution à l'étape précédente, vous devez étendre la limite des autorisations pour permettre des autorisations supplémentaires. La console Lambda détecte les ressources qui ne se trouvent pas dans la limite des autorisations et fournit une stratégie mise à jour que vous pouvez utiliser pour la mettre à jour.

Mettre à jour la limite des autorisations de l'application

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez votre application.
3. Sous Resources (Ressources), choisissez Edit permissions boundary (Modifier la limite des autorisations).
4. Suivez les instructions affichées pour mettre à jour la limite afin d'autoriser l'accès à la nouvelle table.

Pour plus d'informations sur les limites d'autorisations, consultez [Utilisation des limites d'autorisations pour les applications AWS Lambda \(p. 49\)](#).

Mettre à jour le code de la fonction

Ensuite, mettez à jour le code de fonction pour utiliser la table. Le code suivant utilise la table DynamoDB pour effectuer le suivi du nombre d'appels traités par chaque instance de la fonction. Il utilise l'ID du flux de journal comme identifiant unique pour l'instance de fonction.

Mettre à jour le code de la fonction.

1. Ajoutez un nouveau gestionnaire nommé `index.js` dans le dossier `src/handlers` avec le contenu suivant.

Example `src/handlers/index.js`

```
const dynamodb = require('aws-sdk/clients/dynamodb');
const docClient = new dynamodb.DocumentClient();

exports.handler = async (event, context) => {
    const message = 'Hello from Lambda!';
    const tableName = process.env.DDB_TABLE;
    const logStreamName = context.logStreamName;
    var params = {
        TableName : tableName,
        Key: { id : logStreamName },
        UpdateExpression: 'set invocations = if_not_exists(invocations, :start)
+ :inc',
        ExpressionAttributeValues: {
            ':start': 0,
            ':inc': 1
        },
        ReturnValues: 'ALL_NEW'
    };
    await docClient.update(params).promise();
```

```
const response = {
  body: JSON.stringify(message)
};
console.log(`body: ${response.body}`);
return response;
}
```

- Ouvrez le modèle d'application et modifiez la valeur du gestionnaire en `src/handlers/index.handler`.

Example template.yml

```
...
helloFromLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ./
    Handler: src/handlers/index.handler
    Runtime: nodejs10.x
```

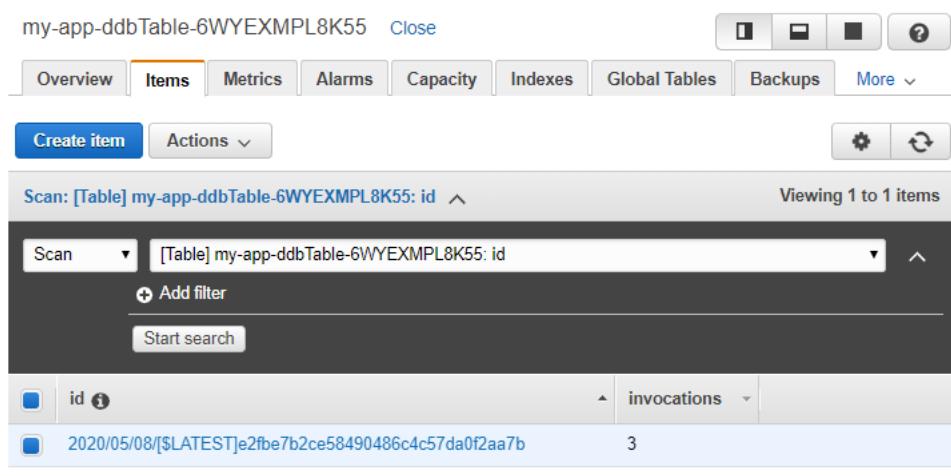
- Valider et transmettre le changement.

```
~/my-app-repo$ git add . && git commit -m "Use DynamoDB table"
~/my-app-repo$ git push
```

Une fois le changement de code déployé, appelez la fonction plusieurs fois pour mettre à jour la table DynamoDB.

Pour afficher la table DynamoDB

- Ouvrez la [page Tables de la console DynamoDB](#).
- Choisissez la table qui commence par my-app.
- Choisissez Items (Éléments).
- Choisissez Start search (Lancer la recherche).



Lambda crée des instances supplémentaires de votre fonction pour gérer plusieurs appels simultanés. Chaque flux de journal du groupe de journaux CloudWatch Logs correspond à une instance de fonction. Une nouvelle instance de fonction est également créée lorsque vous modifiez le code ou la configuration

de votre fonction. Pour plus d'informations sur le dimensionnement, consultez [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#).

Étapes suivantes

Le modèle AWS CloudFormation qui définit vos ressources d'application utilise transformation Modèle d'application sans serveur AWS pour simplifier la syntaxe des définitions de ressources et automatiser le téléchargement du package de déploiement et d'autres artefacts. AWS SAM fournit également une interface de ligne de commande (CLI AWS SAM), qui possède les mêmes fonctionnalités de packaging et de déploiement que l'AWS CLI, avec des fonctionnalités supplémentaires spécifiques aux applications Lambda. Utilisez l'interface de ligne de commande AWS SAM pour tester votre application localement dans un conteneur Docker qui émule l'environnement d'exécution Lambda.

- [Installation de l'interface de ligne de commande AWS SAM](#)
- [Test et débogage d'applications sans serveur](#)

AWS Cloud9 fournit un environnement de développement en ligne qui inclut Node.js, l'interface de ligne de commande AWS SAM et Docker. Avec AWS Cloud9, vous pouvez commencer à développer rapidement et accéder à votre environnement de développement à partir de n'importe quel ordinateur. Pour plus d'informations, consultez [Mise en route](#) dans le AWS Cloud9 Guide de l'utilisateur.

Pour le développement local, les kits d'outils AWS pour les environnements de développement intégrés (IDE) vous permettent de tester et de déboguer les fonctions avant de les transférer vers votre référentiel.

- [AWS Toolkit pour JetBrains](#) – Plugin pour les IDE PyCharm (Python) et IntelliJ (Java).
- [AWS Toolkit for Eclipse](#) – Plugin pour Eclipse IDE (plusieurs langues).
- [AWS Toolkit pour Visual Studio Code](#) – Plugin pour Visual Studio Code IDE (plusieurs langues).
- [AWS Toolkit for Visual Studio](#) – Plugin pour Visual Studio IDE (plusieurs langues).

Dépannage

Au fur et à mesure que vous développez votre application, vous rencontrerez probablement les types d'erreurs suivants.

- Erreurs de génération – Problèmes qui se produisent pendant la phase de génération, y compris les erreurs de compilation, de test et de packaging.
- Erreurs de déploiement – Problèmes qui se produisent lorsque AWS CloudFormation n'est pas en mesure de mettre à jour la pile d'applications. Il s'agit notamment d'erreurs d'autorisations, de limites de compte, de problèmes de service ou d'erreurs de modèle.
- Erreurs d'appel – Erreurs renvoyées par le code ou l'exécution d'une fonction.

Pour les erreurs de génération et de déploiement, vous pouvez en identifier la cause dans la console Lambda.

Résoudre les erreurs d'application

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez une application.
3. Choisissez Déploiements.
4. Pour afficher le pipeline de l'application, choisissez Deployment pipeline (Pipeline de déploiement).
5. Identifiez l'action qui a rencontré une erreur.
6. Pour afficher l'erreur dans le contexte, choisissez Détails (Détails).

Pour les erreurs de déploiement qui se produisent lors de l'action ExécuteChangeSet, le pipeline est lié à une liste d'événements de pile dans la console AWS CloudFormation. Recherchez un événement présentant l'état UPDATE_FAILED. Étant donné que AWS CloudFormation est restauré après une erreur, l'événement pertinent se trouve sous plusieurs autres événements de la liste. Si AWS CloudFormation n'a pu créer aucun jeu de modifications, l'erreur s'affiche sous Change sets (Jeux de modifications) au lieu de sous Events (Événements).

Une cause fréquente d'erreurs de déploiement et d'appel est le manque d'autorisations dans un ou plusieurs rôles. Pour les déploiements, le rôle du pipeline (`CloudFormationRole`) est équivalent aux [autorisations utilisateur \(p. 41\)](#) que vous utiliseriez pour mettre directement à jour une pile AWS CloudFormation. Si vous ajoutez des ressources à votre application ou activez des fonctionnalités Lambda nécessitant des autorisations utilisateur, le rôle de déploiement est utilisé. Vous trouverez un lien vers le rôle de déploiement sous Infrastructure dans la présentation de l'application.

Si votre fonction accède à d'autres services ou ressources AWS, ou si vous activez des fonctionnalités qui requièrent des autorisations supplémentaires, le [rôle d'exécution \(p. 33\)](#) de la fonction est utilisé. Tous les rôles d'exécution créés dans votre modèle d'application sont également soumis à la limite des autorisations de l'application. Cette limite exige votre accord explicite pour l'accès à des services et ressources supplémentaires dans IAM, après avoir ajouté des autorisations au rôle d'exécution dans le modèle.

Par exemple, pour [connecter une fonction à un cloud privé virtuel \(p. 81\)](#) (VPC), vous avez besoin d'autorisations utilisateur pour décrire les ressources VPC. Le rôle d'exécution a besoin d'une autorisation pour gérer les interfaces réseau. Les étapes suivantes sont requises.

1. Ajoutez les autorisations utilisateur requises au rôle de déploiement dans IAM.
2. Ajoutez les autorisations du rôle d'exécution à la limite des autorisations dans IAM.
3. Ajoutez les autorisations du rôle d'exécution au rôle d'exécution dans le modèle d'application.
4. Valider et transmettez pour déployer le rôle d'exécution mis à jour.

Après avoir corrigé les erreurs d'autorisations, choisissez Release change (Modification de version) dans la vue d'ensemble du pipeline pour exécuter à nouveau la génération et le déploiement.

Nettoyage

Vous pouvez continuer à modifier et utiliser l'exemple pour développer votre propre application. Si vous avez terminé d'utiliser l'exemple, supprimez l'application pour éviter de payer le pipeline, le référentiel et le stockage.

Pour supprimer l'application

1. Ouvrez la [console AWS CloudFormation](#).
2. Supprimez la pile d'applications – my-app.
3. Ouvrez la [console Amazon S3](#).
4. Supprimez le compartiment d'artefact – **aws-us-east-2-123456789012-my-app-pipe**.
5. Retournez à la console AWS CloudFormation et supprimez la pile d'infrastructure – serverlessrepo-my-app-toolchain.

Les journaux de fonctions ne sont pas associés à la pile d'applications ou d'infrastructure dans AWS CloudFormation. Supprimez le groupe de journaux séparément dans la console CloudWatch Logs.

Pour supprimer le groupe de journaux

1. Ouvrez la [page Groupes de journaux](#) de la console Amazon CloudWatch.

2. Choisissez le groupe de journaux de la fonction (/aws/lambda/my-app-helloFromLambdaFunction-**YV1VXMPK7QK**).
3. Choisissez Actions, puis Supprimer le groupe de journaux.
4. Sélectionnez Oui, supprimer.

Déploiements propagés pour les fonctions Lambda

Utilisez des déploiements propagés pour contrôler les risques associés à l'introduction de nouvelles versions de votre fonction Lambda. Dans un déploiement propagé, le système déploie automatiquement la nouvelle version de la fonction et envoie progressivement un volume croissant de trafic vers la nouvelle version. La quantité de trafic et le taux d'augmentation sont des paramètres que vous pouvez configurer.

Vous configurez un déploiement propagé via AWS CodeDeploy et AWS SAM. CodeDeploy est un service qui automatisse les déploiements d'application vers des plates-formes informatiques Amazon telles que Amazon EC2 et AWS Lambda. Pour plus d'informations, consultez [Qu'est-ce que CodeDeploy ?](#). En utilisant CodeDeploy pour déployer votre fonction Lambda, vous pouvez facilement surveiller l'état du déploiement et lancer une restauration si vous détectez des problèmes.

AWS SAM est une infrastructure open source qui permet de créer des applications sans serveur. Vous créez un modèle AWS SAM (au format YAML) pour spécifier la configuration des composants requis pour le déploiement propagé. AWS SAM utilise le modèle pour créer et configurer les composants. Pour plus d'informations, consultez [Qu'est-ce que le modèle d'application sans serveur AWS ?](#).

Dans un déploiement propagé, AWS SAM effectue les tâches suivantes :

- Il configure votre fonction Lambda et crée un alias.
La configuration de routage d'alias est la capacité sous-jacente qui implémente le déploiement propagé.
- Elle crée une application CodeDeploy et un groupe de déploiement.
Le groupe de déploiement gère le déploiement propagé et la restauration (si nécessaire).
- Il détecte le moment où vous créez une nouvelle version de votre fonction Lambda.
- Il déclenche CodeDeploy pour démarrer le déploiement de la nouvelle version.

Exemple de modèle AWS SAM Lambda

L'exemple suivant présente un [modèle AWS SAM](#) pour un déploiement propagé simple.

```
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A sample SAM template for deploying Lambda functions.

Resources:
# Details about the myDateTimeFunction Lambda function
myDateTimeFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: myDateTimeFunction.handler
    Runtime: nodejs12.x
# Creates an alias named "live" for the function, and automatically publishes when you
update the function.
  AutoPublishAlias: live
  DeploymentPreference:
# Specifies the deployment configuration
    Type: Linear10PercentEvery2Minutes
```

Ce modèle définit une fonction Lambda nommée `myDateTimeFunction` avec les propriétés suivantes.

AutoPublishAlias

La propriété `AutoPublishAlias` crée un alias nommé `live`. De plus, l'infrastructure AWS SAM détecte automatiquement le moment où vous enregistrez du code nouveau pour la fonction. L'infrastructure publie ensuite une nouvelle version de fonction et met à jour l'alias `live` de sorte qu'il pointe vers la nouvelle version.

DeploymentPreference

La propriété `DeploymentPreference` détermine la vitesse à laquelle l'application CodeDeploy déplace le trafic de la version d'origine de la fonction Lambda vers la nouvelle version. La valeur `Linear10PercentEvery2Minutes` déplace 10 % du trafic toutes les deux minutes vers la nouvelle version.

Pour obtenir la liste des configurations de déploiement prédéfinies, consultez [Configurations de déploiement](#).

Pour accéder à un didacticiel détaillé sur la manière d'utiliser CodeDeploy avec des fonctions Lambda, consultez [Déploiement d'une fonction Lambda mise à jour avec CodeDeploy](#).

Types d'applications Lambda courantes et cas d'utilisation

Lors de la création d'applications sur AWS Lambda, les composants de base sont les fonctions Lambda et les déclencheurs d'événements. Un déclencheur est le service ou l'application AWS qui appelle une fonction, et une fonction Lambda est le code et l'exécution qui traitent les événements. Pour bien comprendre cette distinction, prenez en compte les scénarios suivants :

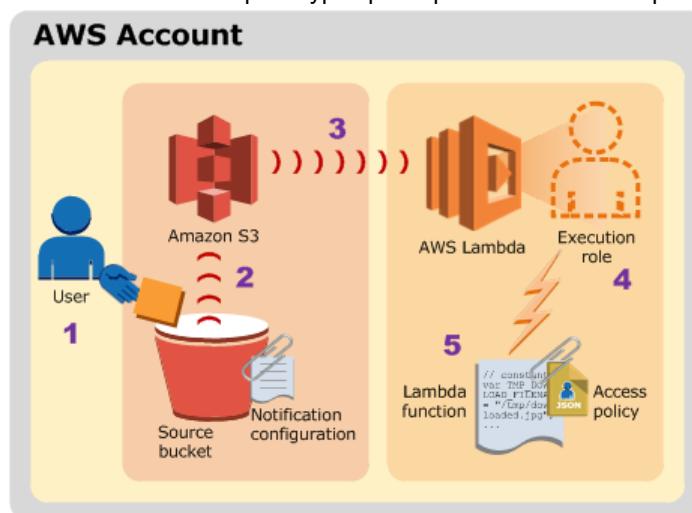
- Traitement des fichiers – Supposons que votre application permette de partager des photos. Les personnes l'utilisent pour importer des photos que l'application stocke dans un compartiment Amazon S3. L'application crée ensuite une miniature des photos de chaque utilisateur et les affiche sur leur page 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.

AWS Lambda prend en charge de nombreux services AWS en tant que sources d'événements. Pour plus d'informations, consultez [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#). 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.

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 \(p. 33\)](#).
4. AWS Lambda exécute la fonction Lambda en spécifiant l'événement comme paramètre.

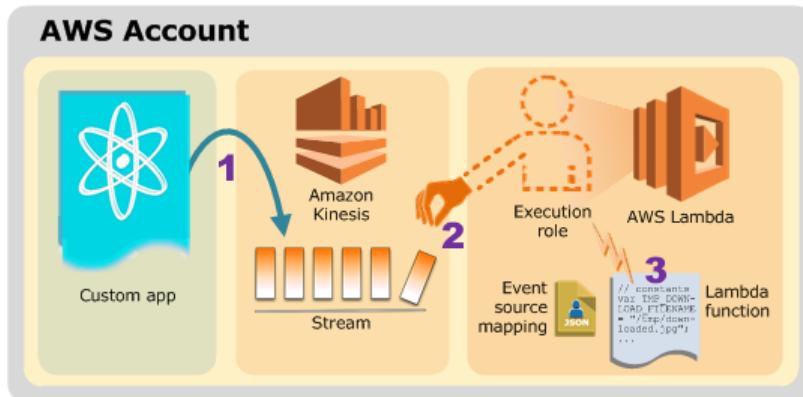
Vous configurez Amazon S3 pour appeler votre fonction en tant qu'action de notification de compartiment. Pour accorder à Amazon S3 l'autorisation d'appeler la fonction, mettez à jour la [stratégie de fonction basée sur les ressources \(p. 36\)](#).

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.

- [CreateEventSourceMapping \(p. 498\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)

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.

Lorsque vous utilisez les sources d'événements basées sur les flux, vous créez des mappages de sources d'événements dans AWS Lambda. Lambda lit les éléments depuis le flux et invoque la fonction de façon synchrone. Vous n'avez pas besoin d'accorder à Lambda l'autorisation d'appeler la fonction. En revanche, une autorisation est nécessaire pour accéder au flux en lecture.

Bonnes pratiques d'utilisation des fonctions AWS Lambda

Les bonnes pratiques suivantes sont recommandées pour l'utilisation d'AWS Lambda :

Rubriques

- [Code de fonction \(p. 151\)](#)
- [Configuration de fonctions \(p. 152\)](#)
- [Alarmes et métriques \(p. 153\)](#)
- [Appels d'événements de flux \(p. 153\)](#)

Code de fonction

- Séparez le gestionnaire Lambda 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. Initialisez les clients SDK et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le répertoire `/tmp`. Les appels ultérieurs traités par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser du temps et des coûts d'exécution.

Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas le contexte d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.

- Utilisez des [variables d'environnement \(p. 55\)](#) 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 [Environnements d'exécution AWS Lambda \(p. 122\)](#)). 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 disposer du contrôle total des dépendances que votre fonction utilise, empaquetez 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`. Pour obtenir des instructions, veuillez consulter [Package de déploiement AWS Lambda en Java \(p. 366\)](#).
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage du [contexte d'exécution \(p. 124\)](#). Par exemple, préférez les infrastructures 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 en savoir plus, consultez la page [Autorisations AWS Lambda \(p. 32\)](#).
- Familiarisez-vous avec [Limites AWS Lambda \(p. 30\)](#). 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 \(p. 504\)](#) et [UpdateFunctionConfiguration \(p. 643\)](#).
 - 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 [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#) 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 \(p. 499\)](#) 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.

Par défaut, Lambda appelle votre fonction dès que des enregistrements sont disponibles dans le flux. Si le lot qu'il lit depuis le flux ne comporte qu'un enregistrement, Lambda n'envoie qu'un enregistrement à la fonction. Pour éviter d'appeler la fonction avec un petit nombre d'enregistrements, vous pouvez demander à la source d'événement de les mettre en mémoire tampon pendant 5 minutes maximum en

configurant une fenêtre de lot. Avant d'appeler la fonction, Lambda continue de lire les enregistrements du flux jusqu'à avoir collecté un lot complet ou jusqu'à l'expiration de la fenêtre de lot.

- 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).

Utilisation de AWS Lambda avec d'autres services

AWS Lambda s'intègre à d'autres services AWS pour appeler des fonctions. Vous pouvez configurer les déclencheurs pour appeler une fonction en réponse aux événements du cycle de vie des ressources, répondre aux demandes HTTP entrantes, consommer les événements d'une file d'attente ou [s'exécuter selon une planification](#) (p. 188).

Chaque service qui s'intègre à Lambda envoie des données à votre fonction dans JSON en tant qu'événement. La structure du document d'événement est différente pour chaque type d'événement, et contient des données relatives à la ressource ou à la demande qui a déclenché la fonction. Les exécutions de Lambda convertissent l'événement en un objet et le transmettent à votre fonction.

L'exemple suivant illustre un événement test à partir d'une [Equilibreur de charge d'application](#) (p. 235) qui représente une demande GET pour /lambda?query=1234ABCD.

Example événement d'une Equilibreur de charge d'application

```
{
  "requestContext": {
    "elb": {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambdac-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
    }
  },
  "httpMethod": "GET",
  "path": "/lambda",
  "queryStringParameters": {
    "query": "1234ABCD"
  },
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "accept-encoding": "gzip",
    "accept-language": "en-US,en;q=0.9",
    "connection": "keep-alive",
    "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
    "upgrade-insecure-requests": "1",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http",
    "x-imforwards": "20"
  },
  "body": "",
  "isBase64Encoded": false
}
```

Note

L'exécution de Lambda convertit le document d'événement en objet et le transmet à votre [gestionnaire de fonctions](#) (p. 17). Pour les langages compilés, Lambda fournit des définitions pour les types d'événements dans une bibliothèque. Pour plus d'informations, consultez les rubriques suivantes.

- [Création de fonctions Lambda avec Java](#) (p. 363)

- [Création de fonctions Lambda avec Go \(p. 398\)](#)
- [Création de fonctions Lambda avec C# \(p. 413\)](#)
- [Création de fonctions Lambda avec PowerShell \(p. 435\)](#)

Pour les services qui génèrent une file d'attente ou un flux de données, vous créez un [mappage de sources d'événements \(p. 101\)](#) dans Lambda et accordez à Lambda l'autorisation d'accéder à l'autre service dans le [rôle d'exécution \(p. 33\)](#). Lambda lit les données à partir de l'autre service, crée un événement et appelle votre fonction.

Services à partir desquels Lambda lit les événements

- [Amazon Kinesis \(p. 239\)](#)
- [Amazon DynamoDB \(p. 208\)](#)
- [Amazon Simple Queue Service \(p. 285\)](#)

Les autres services appellent votre fonction directement. Vous accordez à l'autre service l'autorisation dans la [stratégie de la fonction basée sur les ressources \(p. 36\)](#) et configurez l'autre service pour générer des événements et appeler votre fonction. Selon le service, l'appel peut être synchrone ou asynchrone. Pour un appel synchrone, l'autre service attend la réponse de votre fonction et peut [réessayer en cas d'erreurs \(p. 111\)](#).

Services qui appellent les fonctions Lambda de façon synchrone

- [Elastic Load Balancing \(Equilibreur de charge d'application\) \(p. 235\)](#)
- [Amazon Cognito \(p. 206\)](#)
- [Amazon Lex \(p. 255\)](#)
- [Amazon Alexa \(p. 157\)](#)
- [Amazon API Gateway \(p. 157\)](#)
- [Amazon CloudFront \(Lambda@Edge\) \(p. 197\)](#)
- [Amazon Kinesis Data Firehose \(p. 239\)](#)
- [AWS Step Functions](#)
- [Lot Amazon Simple Storage Service \(p. 276\)](#)

Pour l'appel asynchrone, Lambda place l'événement dans une file d'attente avant de le transmettre à votre fonction. L'autre service obtient une réponse de réussite dès que l'événement est mis en file d'attente et n'est pas conscient de ce qui se passe par la suite. Si une erreur se produit, Lambda gère les [nouvelles tentatives \(p. 111\)](#) et peut envoyer les événements ayant échoué dans une [file d'attente de lettres mortes \(p. 99\)](#) que vous configurez.

Services qui appellent les fonctions Lambda de façon asynchrone

- [Amazon Simple Storage Service \(p. 262\)](#)
- [Amazon Simple Notification Service \(p. 280\)](#)
- [Amazon Simple Email Service \(p. 278\)](#)
- [AWS CloudFormation \(p. 195\)](#)
- [Amazon CloudWatch Logs \(p. 194\)](#)
- [Amazon CloudWatch Events \(p. 188\)](#)
- [AWS CodeCommit \(p. 198\)](#)
- [AWS Config \(p. 207\)](#)
- [AWS IoT \(p. 236\)](#)
- [Événements AWS IoT \(p. 237\)](#)

- [AWS CodePipeline \(p. 199\)](#)

Consultez les rubriques de cette section pour plus de détails sur chaque service, ainsi que des exemples d'événements que vous pouvez utiliser pour tester votre fonction.

Utilisation de AWS Lambda avec 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.

Le kit Alexa Skills est disponible sur GitHub.

- [Kit de développement logiciel \(SDK\) Alexa Skills Kit pour Node.js](#)
- [Kit de développement logiciel \(SDK\) Alexa Skills Kit pour Java](#)

Example Événement Alexa Smart Home

```
{  
  "header": {  
    "payloadVersion": "1",  
    "namespace": "Control",  
    "name": "SwitchOnOffRequest"  
  },  
  "payload": {  
    "switchControlAction": "TURN_ON",  
    "appliance": {  
      "additionalApplianceDetails": {  
        "key2": "value2",  
        "key1": "value1"  
      },  
      "applianceId": "sampleId"  
    },  
    "accessToken": "sampleAccessToken"  
  }  
}
```

Pour plus d'informations, consultez [Mise en route avec le kit Alexa Skills](#).

Utilisation de AWS Lambda avec Amazon API Gateway

Vous pouvez créer une API web avec un point de terminaison HTTP pour votre fonction Lambda à l'aide de Amazon API Gateway. API Gateway fournit des outils pour créer et documenter des API web qui acheminent les requêtes HTTP vers des fonctions Lambda. Vous pouvez sécuriser l'accès à votre API avec des contrôles d'authentification et d'autorisation. Vos API peuvent servir le trafic sur Internet ou peuvent être accessibles uniquement au sein de votre VPC.

Pour ajouter un point de terminaison public à votre fonction Lambda

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.

3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).
4. Choisissez API Gateway.
5. Pour API, choisissez Créer une API.
6. Pour Sécurité, choisissez Ouvrir.
7. Choisissez Add (Ajouter).

Avec le déclencheur API Gateway sélectionné dans le concepteur, choisissez le point de terminaison pour appeler la fonction avec API Gateway.

The screenshot shows the AWS Lambda Designer interface. At the top, there's a header with the AWS logo and 'Lambda'. Below it, a sidebar on the left lists 'Triggers' and 'Functions'. In the main area, a 'Designer' tab is selected. A 'Triggers' card shows a single entry: 'nodejs-apig' with an 'Enabled' status and a 'Delete' button. Below this, an 'APIs' card shows a single entry: 'nodejs-apig' with an 'arn:aws:execute-api:us-east-2:123456789012:5hqaxmpl4f/*/GET/' ARN, an 'Enabled' status, and a 'Delete' button. The 'API endpoint' field contains the URL <https://5hqaxmpl4f.execute-api.us-east-2.amazonaws.com/api/>, which is highlighted with a red box and has a hand cursor icon over it. Below the URL, it says 'API type: rest' and 'Authorization: NONE'.

Les API API Gateway comprennent des étapes, des ressources, des méthodes et des intégrations. L'étape et la ressource déterminent le chemin du point de terminaison :

Format du chemin d'accès de l'API

- `/prod/` – L'étape `prod` et la ressource racine.
- `/prod/user` – L'étape `prod` et la ressource `user`.
- `/dev/{proxy+}` – N'importe quel routage à l'étape `dev`.
- `/` – (API HTTP) Étape par défaut et ressource racine.

Une intégration Lambda mappe un chemin d'accès et une combinaison de méthode HTTP à une fonction Lambda. Vous pouvez configurer API Gateway pour transmettre le corps de la requête HTTP tel quel (intégration personnalisée) ou pour encapsuler le corps de la requête dans un document qui inclut toutes les informations de demande, y compris les en-têtes, la ressource, le chemin d'accès et la méthode.

Amazon API Gateway appelle votre fonction de [manière synchrone \(p. 92\)](#) avec un événement qui contient une représentation JSON de la requête HTTP. Pour une intégration personnalisée, l'événement est le corps de la requête. Pour une intégration par proxy, l'événement a une structure définie. L'exemple suivant montre un événement proxy à partir d'une API API Gateway REST.

Example `event.json` API Gateway Événement de proxy (API REST)

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "GET",
  "requestContext": {
    "resourcePath": "/",
    "httpMethod": "GET",
    "path": "/Prod/",
    ...
  }
}
```

```

},
"headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "accept-encoding": "gzip, deflate, br",
    "Host": "70ixmpl4fl.execute-api.us-east-2.amazonaws.com",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-5e66d96f-7491f09xmpl79d18acf3d050",
    ...
},
"multiValueHeaders": {
    "accept": [
        "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
    ],
    "accept-encoding": [
        "gzip, deflate, br"
    ],
    ...
},
"queryStringParameters": null,
"multiValueQueryStringParameters": null,
"pathParameters": null,
"stageVariables": null,
"body": null,
"isBase64Encoded": false
}

```

Cet exemple montre un événement pour une requête GET au chemin racine de l'étape Production d'une API REST. La forme et le contenu de l'événement varient selon le [type d'API \(p. 162\)](#) et la configuration.

API Gateway attend une réponse de votre fonction et relaie le résultat à l'appelant. Pour une intégration personnalisée, vous définissez une réponse d'intégration et une réponse de méthode pour convertir la sortie de la fonction en réponse HTTP. Pour une intégration par proxy, la fonction doit répondre avec une représentation de la réponse dans un format spécifique.

L'exemple suivant montre un objet de réponse d'une fonction Node.js. L'objet de réponse représente une réponse HTTP réussie qui contient un document JSON.

Example [index.js](#) – Objet de réponse d'intégration de proxy (Node.js)

```

var response = {
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "isBase64Encoded": false,
    "multiValueHeaders": {
        "X-Custom-Header": ["My value", "My other value"]
    },
    "body": "{\n    \"TotalCodeSize\": 104330022,\n    \"FunctionCount\": 26\n}"
}

```

L'exécution Lambda sérialise l'objet de réponse dans JSON et l'envoie à l'API. L'API analyse la réponse et l'utilise pour créer une réponse HTTP, qu'elle envoie ensuite au client qui a fait la demande d'origine.

Example Réponse HTTP

```

< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 55
< Connection: keep-alive

```

```
< x-amzn-RequestId: 32998fea-xmpl-4268-8c72-16138d629356
< X-Custom-Header: My value
< X-Custom-Header: My other value
< X-Amzn-Trace-Id: Root=1-5e6aa925-ccecxmplbae116148e52f036
<
{
    "TotalCodeSize": 104330022,
    "FunctionCount": 26
}
```

Les ressources de votre API définissent une ou plusieurs méthodes, telles que GET ou POST. Les méthodes ont une intégration qui achemine les requêtes vers une fonction Lambda ou un autre type d'intégration. Vous pouvez définir chaque ressource et méthode individuellement, ou utiliser des types de ressource et de méthode spéciaux pour correspondre à toutes les demandes adaptées à un modèle. Une ressource proxy attrape tous les chemins sous une ressource. La méthode ANY attrape toutes les méthodes HTTP.

Sections

- [Autorisations \(p. 160\)](#)
- [Gestion des erreurs avec une API API Gateway \(p. 161\)](#)
- [Choix d'un type d'API \(p. 162\)](#)
- [Exemples d'applications \(p. 164\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec Amazon API Gateway \(p. 164\)](#)
- [Exemple de code de fonction \(p. 173\)](#)
- [Création d'un microservice simple avec Lambda et API Gateway \(p. 175\)](#)
- [Modèle AWS SAM pour une application API Gateway \(p. 177\)](#)

Autorisations

Pour appeler la fonction, Amazon API Gateway a besoin de l'autorisation de la [stratégie basée sur les ressources \(p. 36\)](#) de la fonction. Vous pouvez accorder l'autorisation d'appel à toute une API ou accorder un accès limité à une étape, à une ressource ou à une méthode.

Lorsque vous ajoutez une API à votre fonction à l'aide de la console Lambda, en utilisant la console API Gateway ou un modèle AWS SAM, la stratégie basée sur les ressources de la fonction est mise à jour automatiquement. L'exemple suivant montre une stratégie de fonction avec une instruction qui a été ajoutée par un modèle AWS SAM.

Example stratégie de fonction

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLEXE2F",
            "Effect": "Allow",
            "Principal": {
                "Service": "apigateway.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:nodejs-apig-
function-1G3MXMPLXVXYI",
            "Condition": {
                "ArnLike": {
                    "AWS:SourceArn": "arn:aws:execute-api:us-east-2:123456789012:ktyvxmpls1/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
```

```
        }  
    ]  
}
```

Confirmez la stratégie de fonction (p. 36) dans l'onglet Autorisations de la console Lambda.

Vous pouvez gérer manuellement les autorisations de stratégie de fonction à l'aide des opérations d'API suivantes :

- [AddPermission \(p. 490\)](#)
- [RemovePermission \(p. 620\)](#)
- [GetPolicy \(p. 560\)](#)

Pour accorder l'autorisation d'appel à une API existante, utilisez la commande `add-permission`.

```
$ aws lambda add-permission --function-name my-function \  
--statement-id apigateway-get --action lambda:InvokeFunction \  
--principal apigateway.amazonaws.com \  
--source-arn "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET/" \  
{  
    "Statement": "{\"Sid\": \"apigateway-test-2\", \"Effect\": \"Allow\", \"Principal\": \"  
    {\"Service\": \"apigateway.amazonaws.com\"}, \"Action\": \"lambda:InvokeFunction\", \"Resource\": \"  
    \": \"arn:aws:lambda:us-east-2:123456789012:function:my-function\", \"Condition\": {\"ArnLike\": \"  
    \": {\"AWS:SourceArn\": \"arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET\"}}\"}  
}
```

Note

Si votre fonction et l'API se trouvent dans des régions différentes, l'identificateur de région dans l'ARN source doit correspondre à la région de la fonction, et pas de la région de l'API. Lorsque API Gateway appelle une fonction, il utilise un ARN de ressource basé sur l'ARN de l'API, mais modifié pour correspondre à la région de la fonction.

L'ARN source dans cet exemple accorde l'autorisation à une intégration sur la méthode GET de la ressource racine au cours de l'étape par défaut d'une API, avec l'ID `mnh1xmpli7`. Vous pouvez utiliser un astérisque dans l'ARN source pour accorder des autorisations à plusieurs étapes, méthodes ou ressources.

Modèles de ressources

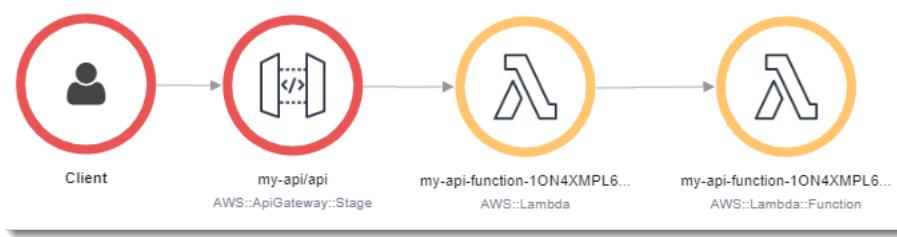
- `mnh1xmpli7/*/GET/*` – Méthode GET sur toutes les ressources à toutes les étapes.
- `mnh1xmpli7/prod/ANY/user` – TOUTE méthode sur la ressource `user` au cours de l'étape `prod`.
- `mnh1xmpli7/*/*/*` – Toute méthode sur toutes les ressources à toutes les étapes.

Pour de plus amples informations sur l'affichage de la stratégie et la suppression des instructions, veuillez consulter [Nettoyage des stratégies basées sur les ressources \(p. 40\)](#).

Gestion des erreurs avec une API API Gateway

API Gateway traite toutes les erreurs d'appel et de fonction comme des erreurs internes. Si l'API Lambda rejette la demande d'appel, API Gateway renvoie un code d'erreur 500. Si la fonction s'exécute mais renvoie une erreur, ou renvoie une réponse au mauvais format, API Gateway renvoie un 502. Dans les deux cas, le corps de la réponse de API Gateway est `{"message": "Internal server error"}`.

L'exemple suivant montre une carte de trace X-Ray pour une demande qui a entraîné une erreur de fonction et un 502 issu de API Gateway. Le client reçoit le message d'erreur générique.

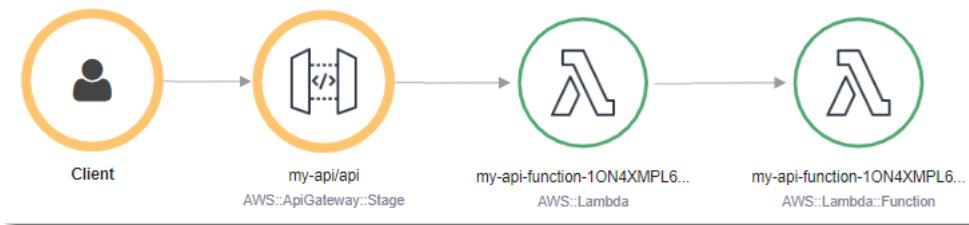


Pour personnaliser la réponse d'erreur, vous devez attraper les erreurs dans votre code et formater une réponse dans le format requis.

Example [index.js](#) – Erreur de formatage

```
var formatError = function(error){  
    var response = {  
        "statusCode": error.statusCode,  
        "headers": {  
            "Content-Type": "text/plain",  
            "x-amzn-ErrorType": error.code  
        },  
        "isBase64Encoded": false,  
        "body": error.code + ":" + error.message  
    }  
    return response  
}
```

API Gateway convertit cette réponse en une erreur HTTP avec un code d'état et un corps personnalisés. Dans la carte de trace, le nœud de fonction est vert car il a géré l'erreur.



Choix d'un type d'API

API Gateway prend en charge trois types d'API qui appellent des fonctions Lambda :

- API HTTP – Une API RESTful légère à faible latence.
- API REST – Une API RESTful personnalisable et riche en fonctionnalités.
- API WebSocket – API web qui gère des connexions persistantes avec les clients pour la communication duplex intégral.

Les API HTTP et les API REST sont toutes deux des API RESTful qui traitent les requêtes HTTP et les réponses renvoyées. Les API HTTP sont plus récentes et sont construites avec l'API API Gateway version 2. Les fonctionnalités suivantes sont nouvelles pour les API HTTP :

Fonctionnalités de l'API HTTP

- Déploiements automatiques – Lorsque vous modifiez des routages ou des intégrations, les modifications se déploient automatiquement sur des étapes pour lesquelles le déploiement automatique est activé.

- Étape par défaut – Vous pouvez créer une étape par défaut (`$default`) pour servir les requêtes au chemin d'accès racine de l'URL de votre API. Pour les étapes nommées, vous devez inclure le nom de l'étape au début du chemin d'accès.
- Configuration CORS – Vous pouvez configurer votre API pour ajouter des en-têtes CORS aux réponses sortantes, au lieu de les ajouter manuellement dans votre code de fonction.

Les API REST sont les API RESTful classiques prises en charge par API Gateway depuis leur lancement. Les API REST disposent actuellement d'un plus grand nombre de fonctionnalités de personnalisation, d'intégration et de gestion.

Fonctionnalités de l'API REST

- Types d'intégration – Les API REST prennent en charge les intégrations Lambda personnalisées. Avec une intégration personnalisée, vous pouvez envoyer uniquement le corps de la requête à la fonction, ou appliquer un modèle de transformation au corps de la requête avant de l'envoyer à la fonction.
- Contrôle d'accès – Les API REST prennent en charge plus d'options pour l'authentification et l'autorisation.
- Surveillance et traçage – Les API REST prennent en charge le traçage AWS X-Ray et des options de journalisation supplémentaires.

Pour une comparaison détaillée, consultez [Choix entre les API HTTP et les API REST](#) dans le Guide du développeur API Gateway.

Les API WebSocket utilisent également l'API API Gateway version 2 et prennent en charge un ensemble de fonctionnalités similaire. Utilisez une API WebSocket pour les applications qui bénéficient d'une connexion persistante entre le client et l'API. Les API WebSocket fournissent une communication duplex intégral, ce qui signifie que le client et l'API peuvent envoyer des messages en continu sans attendre de réponse.

Les API HTTP prennent en charge un format d'événement simplifié (version 2.0). L'exemple suivant montre un événement provenant d'une API HTTP.

Example `event-v2.json` – Événement de proxy API Gateway (API HTTP)

```
{  
    "version": "2.0",  
    "routeKey": "ANY /nodejs-apig-function-1G3XmplZxVXYI",  
    "rawPath": "/default/nodejs-apig-function-1G3XmplZxVXYI",  
    "rawQueryString": "",  
    "cookies": [  
        "s_fid=7AABXmpl1AFD9BBF-0643Xmpl09956DE2",  
        "regStatus=pre-register"  
    ],  
    "headers": {  
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",  
        "accept-encoding": "gzip, deflate, br",  
        ...  
    },  
    "requestContext": {  
        "accountId": "123456789012",  
        "apiId": "r3pmxmplak",  
        "domainName": "r3pmxmplak.execute-api.us-east-2.amazonaws.com",  
        "domainPrefix": "r3pmxmplak",  
        "http": {  
            "method": "GET",  
            "path": "/default/nodejs-apig-function-1G3XmplZxVXYI",  
            "protocol": "HTTP/1.1",  
            "stage": "prod"  
        }  
    }  
}
```

```
        "sourceIp": "205.255.255.176",
        "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "requestId": "JKJaXmPLvHcEsha=",
    "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
    "stage": "default",
    "time": "10/Mar/2020:05:16:23 +0000",
    "timeEpoch": 1583817383220
},
"isBase64Encoded": true
}
```

Pour plus d'informations, consultez [Intégrations AWS Lambda](#) dans le Guide du développeur API Gateway.

Exemples d'applications

Le référentiel GitHub de ce guide fournit l'exemple d'application suivant pour API Gateway.

- [API Gateway avec Node.js](#) – Fonction avec un modèle AWS SAM qui crée une API REST dont le suivi AWS X-Ray est activé. Il inclut des scripts pour le déploiement, l'appel de la fonction, le test de l'API et le nettoyage.

Lambda fournit également des [plans \(p. 27\)](#) et des [modèles \(p. 28\)](#) que vous pouvez utiliser pour créer une application API Gateway dans la console Lambda.

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": "Lambda API Gateway"
        }
    }
}
```

```
        "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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution](#) (p. 33) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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 API Gateway 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 de code dans d'autres langages, veuillez consulter [Exemple de code de fonction \(p. 173\)](#).

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 nodejs12.x \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

Tester 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=bs8fgo6bp0
$ 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": "iuig5w",
  "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": "20vgzsz",
```

```
    "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 une autorisation 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.

Note

Si votre fonction et l'API se trouvent dans des régions différentes, l'identificateur de région dans l'ARN source doit correspondre à la région de la fonction, et pas de la région de 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 Créer.

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`). Étant 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 `testInvokeMethod` 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 testInvokeMethod --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 \(p. 173\)](#)
- [Python 3 \(p. 174\)](#)
- [Go \(p. 175\)](#)

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}');
}
};
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

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
    '''
    #print("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)')
```

Comptez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

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
}
```

Générez l'exécutable avec `go build` et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Go \(p. 398\)](#).

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 \(p. 217\)](#)

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 trouverez ci-dessous un exemple de modèle AWS SAM pour l'application Lambda issue de ce [tutoriel \(p. 164\)](#). Copiez le texte ci-dessous dans un fichier et enregistrez-le en regard du package ZIP que vous avez créé précédemment. 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: nodejs12.x
      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 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.

Example Journal CloudTrail

```
{
  "Records": [
    {
      "eventVersion": "1.02",
```

```

"userIdentity": {
    "type": "Root",
    "principalId": "123456789012",
    "arn": "arn:aws:iam::123456789012:root",
    "accountId": "123456789012",
    "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-east-2",
"sourceIPAddress": "205.251.233.176",
"userAgent": "console.amazonaws.com",
"requestParameters": {
    "name": "dropmeplease"
},
"responseElements": {
    "topicArn": "arn:aws:sns:us-east-2:123456789012:exampletopic"
},
"requestID": "3fdb7834-9079-557e-8ef2-350abc03536b",
"eventID": "17b46459-dada-4278-b8e2-5a4ca9ff1a9c",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
},
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2015-01-24T22:41:54Z"
            }
        }
    },
    "eventTime": "2015-01-24T23:27:02Z",
    "eventSource": "sns.amazonaws.com",
    "eventName": "GetTopicAttributes",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "205.251.233.176",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "topicArn": "arn:aws:sns:us-east-2:123456789012:exampletopic"
    },
    "responseElements": null,
    "requestID": "4a0388f7-a0af-5df9-9587-c5c98c29cbec",
    "eventID": "ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
]
}

```

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 \(p. 262\)](#).

Rubriques

- [Journalisation des appels d'API AWS Lambda avec AWS CloudTrail \(p. 179\)](#)
- [Didacticiel : Déclenchement d'une fonction Lambda avec des événements AWS CloudTrail \(p. 182\)](#)
- [Exemple de code de fonction \(p. 186\)](#)

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 de plus amples informations, veuillez consulter [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 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 \(p. 490\)](#)
- [CreateEventSourceMapping \(p. 498\)](#)
- [CreateFunction \(p. 504\)](#)

(Le paramètre `ZipFile` ne figure pas dans les journaux CloudTrail pour `CreateFunction`.)

- [DeleteEventSourceMapping \(p. 515\)](#)
 - [DeleteFunction \(p. 519\)](#)
 - [GetEventSourceMapping \(p. 534\)](#)
 - [GetFunction \(p. 538\)](#)
 - [GetFunctionConfiguration \(p. 543\)](#)
 - [GetPolicy \(p. 560\)](#)
 - [ListEventSourceMappings \(p. 575\)](#)
 - [ListFunctions \(p. 581\)](#)
 - [RemovePermission \(p. 620\)](#)
 - [UpdateEventSourceMapping \(p. 630\)](#)
 - [UpdateFunctionCode \(p. 636\)](#)
- (Le paramètre `ZipFile` ne figure pas dans les journaux CloudTrail pour `UpdateFunctionCode`.)
- [UpdateFunctionConfiguration \(p. 643\)](#)

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 la [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`.

```
{  
  "Records": [  
    {  
      "eventVersion": "1.03",  
      "userIdentity": {  
        "type": "IAMUser",  
        "principal": "arn:aws:iam::123456789012:root",  
        "arn": "arn:aws:iam::123456789012:user/root",  
        "accountId": "123456789012",  
        "sessionContext": {  
          "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
          "sessionIssuer": {  
            "arn": "arn:aws:sts::123456789012:assumed-role/LambdaTestRole/CloudTrail",  
            "type": "AWS",  
            "principal": "lambda.amazonaws.com",  
            "accountId": "123456789012",  
            "ARN": "arn:aws:sts::123456789012:assumed-role/LambdaTestRole/CloudTrail",  
            "sessionName": "CloudTrailAssumeRoleSession",  
            "sessionDuration": 3600  
          },  
          "sessionToken": "QWERTYUIOPASDFGHJKLZXCVBNM1234567890",  
          "expiration": "2023-01-12T12:00:00Z",  
          "startTime": "2023-01-12T11:59:59Z"  
        }  
      }  
    }  
  ]  
}
```

```

"principalId": "A1B2C3D4E5F6G7EXAMPLE",
"arn": "arn:aws:iam::999999999999:user/myUserName",
"accountId": "999999999999",
"accessKeyId": "AKIAIOSFODNN7EXAMPLE",
"userName": "myUserName"
},
"eventTime": "2015-03-18T19:03:36Z",
"eventSource": "lambda.amazonaws.com",
"eventName": "GetFunction",
"awsRegion": "us-east-1",
"sourceIPAddress": "127.0.0.1",
"userAgent": "Python-httplib2/0.8 (gzip)",
"errorCode": "AccessDenied",
"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 plus d'informations sur cette option, consultez [Journalisation d'événements de données pour les journaux de suivi](#).

Didacticiel : Déclenchement d'une fonction Lambda avec des événements AWS CloudTrail

Vous pouvez configurer Amazon S3 pour publier des événements dans AWS Lambda lorsque AWS CloudTrail stocke des journaux d'appels d'API. La fonction Lambda peut lire l'objet de journal et traiter les enregistrements d'accès consignés par CloudTrail.

Utilisez les instructions suivantes pour créer une fonction Lambda qui vous informe d'un appel d'API spécifique effectué dans votre compte. La fonction traite les événements de notification à partir de Amazon S3, lit les journaux à partir d'un compartiment et publie des alertes à travers une rubrique Amazon SNS. Dans le cadre de ce didacticiel, vous créez ce qui suit :

- Un journal de suivi CloudTrail et un compartiment S3 dans lequel enregistrer les journaux.
- Une rubrique Amazon SNS pour publier des notifications d'alerte.
- Un rôle d'utilisateur IAM avec des autorisations pour lire des éléments à partir d'un compartiment S3 et écrire des journaux dans Amazon CloudWatch.
- Une fonction Lambda qui traite les journaux CloudTrail et envoie une notification chaque fois qu'une rubrique Amazon SNS est créée.

Exigences

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Avant de commencer, assurez-vous de disposer des outils suivants :

- [Node.js 8 avec npm](#).
- Shell Bash. Pour Linux et macOS, inclus par défaut. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et de Bash.
- L'interface de ligne de commande AWS.

Étape 1 : Création d'un journal de suivi dans CloudTrail

Lorsque vous créez un journal de suivi, CloudTrail enregistre les appels d'API dans des fichiers journaux et les stocke dans Amazon S3. Un journal CloudTrail est un tableau d'événements non triés au format JSON. Pour chaque appel à une action d'API prise en charge, CloudTrail enregistre des informations sur la demande et l'entité qui l'a exécutée. Les événements de journaux incluent le nom de l'action, les paramètres, les valeurs de réponse et des détails sur le demandeur.

Pour créer un journal d'activité

1. Ouvrez la page [Trails \(Suivis\) de la console CloudTrail](#).
2. Choisissez Create trail (Créer un journal de suivi).
3. Pour Trail name (Nom du suivi), saisissez un nom.
4. Pour S3 bucket (Compartiment S3), saisissez un nom.
5. Sélectionnez Create.
6. Enregistrez l'Amazon Resource Name (ARN) du compartiment pour l'ajouter au rôle d'exécution IAM que vous créerez ultérieurement.

Étape 2 : Création d'une rubrique Amazon SNS

Créez une rubrique Amazon SNS pour envoyer une notification lorsque de nouveaux événements d'objet se sont produits.

Pour créer une rubrique

1. Accédez à la page [Topics \(Rubriques\)](#) de la console Amazon SNS.
2. Choisissez Create topic.
3. Pour Topic name (Nom de la rubrique), saisissez un nom.
4. Choisissez Create topic.
5. Enregistrez l'ARN de la rubrique. Vous en aurez besoin lorsque vous créerez le rôle d'exécution IAM et la fonction Lambda.

Étape 3 : Création d'un rôle d'exécution IAM

Un [rôle d'exécution \(p. 33\)](#) donne à votre fonction l'autorisation d'accéder aux ressources AWS. Créez un rôle d'exécution qui accorde à la fonction l'autorisation d'accéder à CloudWatch Logs, Amazon S3 et Amazon SNS.

Pour créer un rôle d'exécution

1. Accédez à la page [Roles \(Rôles\)](#) de la console IAM.
2. Sélectionnez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
 - Pour Trusted entity (Entité de confiance), choisissezLambda.
 - Pour Role name (Nom du rôle), saisissez **lambda-cloudtrail-role**.
 - Pour Permissions (Autorisations), créez une stratégie personnalisée avec les instructions suivantes. Remplacez les valeurs mises en surbrillance par les noms de votre compartiment et de votre rubrique.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:*"  
            ],  
            "Resource": "arn:aws:logs:***:***"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::my-bucket/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:us-west-2:123456789012:my-topic"  
        }  
    ]  
}
```

```
    ]  
}
```

4. Enregistrez l'ARN du rôle. Vous en aurez besoin lors de la création de la fonction Lambda.

Étape 4 : Création de la fonction Lambda

La Lambda fonction suivante traite CloudTrail les journaux et envoie une notification via Amazon SNS lorsqu'une nouvelle Amazon SNS rubrique est créée.

Pour créer la fonction

1. Créez un dossier et attribuez-lui un nom qui indique qu'il s'agit de votre fonction Lambda (par exemple, *lambda-cloudtrail*).
2. Dans le dossier, créez un fichier appelé `index.js`.
3. Collez le code suivant dans `index.js`. Remplacez l'ARN de la rubrique Amazon SNS par l'ARN Amazon S3 créé par lors de la création de la rubrique Amazon SNS.

```
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-east-2';  
var SNS_TOPIC_ARN        = 'arn:aws:sns:us-west-2:123456789012:my-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");
});
};
```

4. Dans le dossier `lambda-cloudtrail` exécutez le script suivant. Il crée un fichier `package-lock.json` et un dossier `node_modules`, qui gèrent toutes les dépendances.

```
$ npm install --async
```

5. Exécutez le script suivant pour créer un package de déploiement.

```
$ zip -r function.zip .
```

6. Créez une fonction Lambda nommée CloudTrailEventProcessing avec la commande `create-function` en exécutant le script suivant. Effectuez les remplacements indiqués.

```
$ aws lambda create-function --function-name CloudTrailEventProcessing \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x --timeout
10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-cloudtrail-role
```

Étape 5 :Ajout d'autorisations à la stratégie de la fonction Lambda

La stratégie de ressources de la fonction Lambda a besoin d'autorisations pour permettre à Amazon S3 d'appeler la fonction.

Accorder à Amazon S3 des autorisations pour appeler la fonction

1. Exécutez la commande suivante `add-permission`. Remplacez l'ARN et l'ID de compte par les vôtres.

```
$ aws lambda add-permission --function-name CloudTrailEventProcessing \
--statement-id Id-1 --action "lambda:InvokeFunction" --principal s3.amazonaws.com \
--source-arn arn:aws:s3:::my-bucket \
--source-account 123456789012
```

Cette commande accorde au service principal Amazon S3 (`s3.amazonaws.com`) les autorisations nécessaires pour effectuer l'action `lambda:InvokeFunction`. Les autorisations d'appel sont accordées à Amazon S3 uniquement lorsque les conditions suivantes sont réunies :

- CloudTrail stocke un objet de journal dans le compartiment spécifié.
 - Le compartiment appartient au compte AWS spécifié. 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 le compte AWS spécifique peut appeler votre fonction Lambda.
2. Pour afficher la stratégie d'accès de la fonction Lambda, exécutez la commande `get-policy` suivante et remplacez le nom de la fonction.

```
$ aws lambda get-policy --function-name function-name
```

Étape 6 : Configuration des notifications sur un compartiment Amazon S3

Pour demander à que Amazon S3 publie des événements de création d'objet dans Lambda, ajoutez une configuration de notification au compartiment S3. Dans cette configuration, spécifiez les éléments suivants :

- Type d'événement – Tous les types d'événement qui créent des objets.
- Fonction Lambda – La fonction Lambda que vous souhaitez que Amazon S3 appelle.

Pour configurer des notifications

1. Ouvrez la [console Amazon S3](#).
2. Choisissez le compartiment source.
3. Choisissez Propriétés.
4. Sous Events (Événements), configurez une notification avec les paramètres suivants :
 - Name (Nom) – **lambda-trigger**
 - Events (Événements) – **All object create events**
 - Send to (Envoyer à) – **Lambda function**
 - Lambda – **CloudTrailEventProcessing**

Quand CloudTrail stocke les journaux dans le compartiment, Amazon S3 envoie un événement à la fonction. L'événement fournit des informations, y compris le nom du compartiment et le nom de clé de l'objet de journal créé par CloudTrail.

Exemple de code de fonction

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js \(p. 187\)](#)

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 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");  
});  
};
```

Comprimez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

Utilisation de AWS Lambda avec 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](#).

CloudWatch Events appelle votre fonction de manière asynchrone avec un document d'événement qui enveloppe l'événement de sa source. L'exemple suivant montre un événement issu d'un instantané de base de données dans Amazon Relational Database Service.

Example CloudWatch Events event

```
{  
  "version": "0",  
  "id": "fe8d3c65-xmpl-c5c3-2c87-81584709a377",  
  "detail-type": "RDS DB Instance Event",  
  "source": "aws.rds",  
  "account": "123456789012",  
  "time": "2020-04-28T07:20:20Z",  
  "region": "us-east-2",  
  "resources": [  
    "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1"  
,  
  ],  
  "detail": {  
    "EventCategories": [  
      "backup"  
    ],  
    "SourceType": "DB_INSTANCE",  
    "SourceArn": "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1",  
    "Date": "2020-04-28T07:20:20.112Z",  
    "Message": "Finished DB Instance backup",  
    "SourceIdentifier": "rdz6xmpliljlb1"
```

```
}
```

Vous pouvez également 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.

Example Événement de message CloudWatch Events

```
{
  "account": "123456789012",
  "region": "us-east-2",
  "detail": {},
  "detail-type": "Scheduled Event",
  "source": "aws.events",
  "time": "2019-03-01T01:23:45Z",
  "id": "cdc73f9d-aea9-11e3-9d5a-835b769c0d9c",
  "resources": [
    "arn:aws:events:us-east-1:123456789012:rule/my-schedule"
  ]
}
```

Pour configurer CloudWatch Events pour appeler votre fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisir une fonction
3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).
4. Définissez le type de déclencheur sur CloudWatch Events/EventBridge.
5. Pour Rule (Règle), choisissez Create a new rule (Créer une règle).
6. Configurez les options restantes et choisissez Ajouter.

Pour de plus amples informations sur la planification des expressions, veuillez consulter [Expressions de planification à l'aide de rate ou de cron \(p. 193\)](#).

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.

Rubriques

- [Didacticiel : Utilisation d'AWS Lambda avec les événements planifiés \(p. 189\)](#)
- [Modèle AWS SAM pour une application CloudWatch Events \(p. 192\)](#)
- [Expressions de planification à l'aide de rate ou de cron \(p. 193\)](#)

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Créer 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 de plus amples informations sur la planification des expressions, veuillez consulter [Expressions de planification à l'aide de rate ou de cron \(p. 193\)](#).

Tester la fonction Lambda

Testez la fonction avec un exemple d'événement fourni par la console Lambda.

1. Ouvrez la [page Fonctions](#) de la console Lambda.
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 (Amazon SNS) 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 Alarmes.
3. Choisissez Créer une alarme.
4. Choisissez Alarmes.
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 du graphique de prévisualisation.

- 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 qui déclenchera l'alarme.

Pour déclencher une alarme

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez lambda-canary.
3. Sous Variables d'environnement, choisissez Modifier.
4. Définir attendue sur **404**.
5. Choisissez Enregistrer.

Attendez une minute, puis recherchez un message d'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 de ce [tutoriel \(p. 189\)](#). 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.

Expressions de planification à l'aide de rate ou de cron

AWS Lambda prend en charge les expressions cron et rate standard pour les fréquences allant jusqu'à une fois par minute. Les expressions rate de CloudWatch Events ont le format suivant.

```
rate(Value Unit)
```

Où **Value** est un nombre entier positif et **Unit** des minutes, heures ou jours. 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)`).

Exemples d'expressions rate

| Fréquence | Expression |
|----------------------|------------------------------|
| Toutes les 5 minutes | <code>rate(5 minutes)</code> |
| Toutes les heures | <code>rate(1 hour)</code> |
| Tous les sept jours | <code>rate(7 days)</code> |

Les expressions cron ont le format ci-dessous.

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Exemples d'expressions cron

| Fréquence | Expression |
|---|---|
| 10 h 15 (UTC) tous les jours | <code>cron(15 10 * * ? *)</code> |
| 18 h 00 du lundi au vendredi | <code>cron(0 18 ? * MON-FRI *)</code> |
| 8 h 00 le premier jour du mois | <code>cron(0 8 1 * ? *)</code> |
| Toutes les 10 minutes les jours de semaine | <code>cron(0/10 * ? * MON-FRI *)</code> |
| Toutes les 5 minutes entre 8 h 00 et 17 h 55 les jours de semaine | <code>cron(0/5 8-17 ? * MON-FRI *)</code> |

| Fréquence | Expression |
|--|---------------------|
| 9 h 00 le premier lundi de chaque mois | cron(0 9 ? * 2#1 *) |

Notez bien ce qui suit :

- Si vous utilisez la console Lambda, n'incluez pas le préfixe cron à votre expression.
- L'une des valeurs pour le jour du mois ou le jour de la semaine doit être un point d'interrogation (?).

Pour plus d'informations, consultez [Expressions de planification pour les règles](#) dans le Guide de l'utilisateur de CloudWatch Events.

Utilisation de AWS Lambda avec Amazon CloudWatch Logs

Vous pouvez utiliser une fonction Lambda pour surveiller et analyser les journaux à partir d'un flux de journaux Amazon CloudWatch Logs. Créez des [abonnements](#) pour un ou plusieurs flux de journaux afin d'appeler une fonction lorsque des journaux sont créés ou correspondent à un modèle facultatif. Utilisez la fonction pour envoyer une notification ou conserver le journal dans une base de données ou un emplacement de stockage.

CloudWatch Logs appelle votre fonction de façon asynchrone avec un événement qui contient les données de journal. La valeur du champ de données est un fichier ZIP codé en Base64.

Example Événement de message de l'Amazon CloudWatch Logs

```
{
  "awslogs": {
    "data": "ewogICAgIm1lc3NhZ2VUeXB1IjogIkRBVEFTUVTUOFHRSIsCiAgICAib3duZXIIoAiMTIzNDU2Nzg5MDEyIiwKICAgICJsb2dH"
  }
}
```

Une fois décodées et décompressées, les données du journal constituent un document JSON avec la structure suivante.

Example Données de message de l'Amazon CloudWatch Logs (décodées)

```
{
  "messageType": "DATA_MESSAGE",
  "owner": "123456789012",
  "logGroup": "/aws/lambda/echo-nodejs",
  "logStream": "2019/03/13/[$LATEST]94fa867e5374431291a7fc14e2f56ae7",
  "subscriptionFilters": [
    "LambdaStream_cloudwatchlogs-node"
  ],
  "logEvents": [
    {
      "id": "34622316099697884706540976068822859012661220141643892546",
      "timestamp": 1552518348220,
      "message": "REPORT RequestId: 6234bffe-149a-b642-81ff-2e8e376d8aff\\tDuration: 46.84 ms\\tBilled Duration: 100 ms \\tMemory Size: 192 MB\\tMax Memory Used: 72 MB\\t\\n"
    }
  ]
}
```

```
    ]  
}
```

Pour obtenir un exemple d'application qui utilise CloudWatch Logs en tant que déclencheur, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de AWS Lambda avec AWS CloudFormation

Dans un modèle AWS CloudFormation, vous pouvez spécifier une fonction Lambda comme cible pour une ressource personnalisée. Utilisez des ressources personnalisées pour traiter des paramètres, récupérer des valeurs de configuration ou appeler d'autres services AWS lors d'événements du cycle de vie de la pile.

L'exemple suivant appelle une fonction qui est définie ailleurs dans le modèle.

Example – Définition de ressources personnalisées

```
Resources:  
  primerinvoke:  
    Type: AWS::CloudFormation::CustomResource  
    Version: "1.0"  
    Properties:  
      ServiceToken: !GetAtt primer.Arn  
      FunctionName: !Ref randomerror
```

Le jeton de service est l'Amazon Resource Name (ARN) de la fonction appelée par AWS CloudFormation lorsque vous créez, mettez à jour ou supprimez la pile. Vous pouvez également inclure des propriétés supplémentaires comme `FunctionName`, transmises par AWS CloudFormation à votre fonction en l'état.

AWS CloudFormation appelle votre fonction Lambda [de façon asynchrone \(p. 93\)](#) avec un événement qui inclut une URL de rappel.

Example – Événement de message AWS CloudFormation

```
{  
  "RequestType": "Create",  
  "ServiceToken": "arn:aws:lambda:us-east-2:123456789012:function:lambda-error-processor-primer-14ROR2T3JKU66",  
  "ResponseURL": "https://cloudformation-custom-resource-response-useast2.s3-us-east-2.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-2%3A123456789012%3Astack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456%7Cprimerinvoke%7C5d478078-13e9-baf0-464a-7ef285ecc786?  
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1555451971&Signature=28UijsPe5I4dvukKQqM  
%2F9Rf1o4%3D",  
  "StackId": "arn:aws:cloudformation:us-east-2:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",  
  "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",  
  "LogicalResourceId": "primerinvoke",  
  "ResourceType": "AWS::CloudFormation::CustomResource",  
  "ResourceProperties": {  
    "ServiceToken": "arn:aws:lambda:us-east-2:123456789012:function:lambda-error-processor-primer-14ROR2T3JKU66",  
    "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"  
  }  
}
```

La fonction se charge de renvoyer à l'URL de rappel une réponse indiquant le succès ou l'échec. Pour obtenir la syntaxe de réponse complète, consultez [Objets de réponse des ressources personnalisées](#).

Example – Réponse de ressource personnalisée AWS CloudFormation

```
{
    "Status": "SUCCESS",
    "PhysicalResourceId": "2019/04/18/[ $LATEST ]b3d1bfc65f19ec610654e4d9b9de47a0",
    "StackId": "arn:aws:cloudformation:us-east-2:123456789012:stack/lambda-error-
processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
    "LogicalResourceId": "primerinvoke"
}
```

AWS CloudFormation fournit une bibliothèque appelée `cfn-response` qui traite l'envoi de la réponse. Si vous définissez votre fonction au sein d'un modèle, vous pouvez demander la bibliothèque par son nom. AWS CloudFormation ajoute alors la bibliothèque au package de déploiement qu'il crée pour la fonction.

L'exemple de fonction suivant appelle une deuxième fonction. Si l'appel aboutit, la fonction envoie une réponse de réussite à AWS CloudFormation et la mise à jour de la pile continue. Le modèle utilise le type de ressource [AWS::Serverless::Function](#) fourni par Modèle d'application sans serveur AWS.

Example `error-processor/template.yml` – Fonction de ressource personnalisée

```
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      InlineCode: |
        var aws = require('aws-sdk');
        var response = require('cfn-response');
        exports.handler = function(event, context) {
          // For Delete requests, immediately send a SUCCESS response.
          if (event.RequestType == "Delete") {
            response.send(event, context, "SUCCESS");
            return;
          }
          var responseStatus = "FAILED";
          var responseData = {};
          var functionName = event.ResourceProperties.FunctionName
          var lambda = new aws.Lambda();
          lambda.invoke({ FunctionName: functionName }, function(err, invokeResult) {
            if (err) {
              responseData = {Error: "Invoke call failed"};
              console.log(responseData.Error + ":\n", err);
            }
            else responseStatus = "SUCCESS";
            response.send(event, context, responseStatus, responseData);
          });
        };
        Description: Invoke a function to create a log stream.
        MemorySize: 128
        Timeout: 8
        Role: !GetAtt role.Arn
        Tracing: Active
```

Si la fonction appelée par la ressource personnalisée n'est pas définie dans un modèle, vous pouvez obtenir le code source pour `cfn-response` à partir de [Module cfn-response](#) dans le AWS CloudFormation Guide de l'utilisateur.

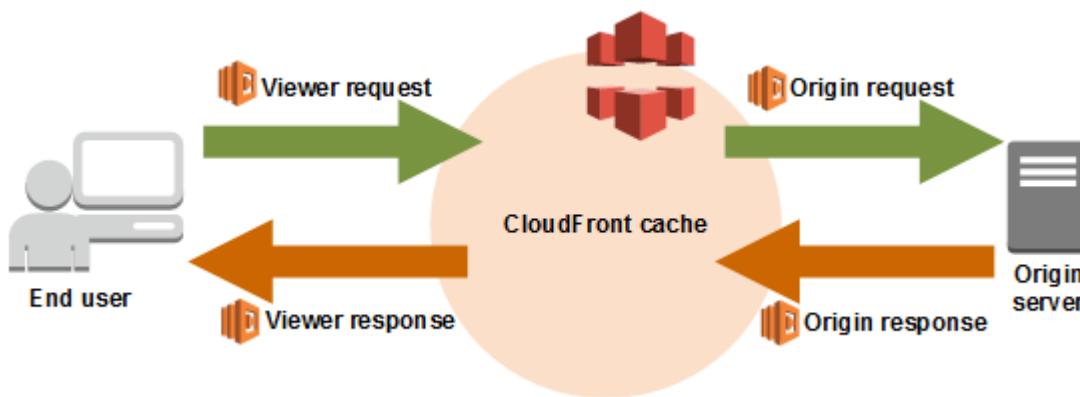
Pour obtenir un exemple d'application qui utilise une ressource personnalisée pour s'assurer que le groupe de journaux d'une fonction est créé avant la ressource qui en dépend, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Pour plus d'informations sur les ressources personnalisées, consultez [Ressources personnalisées](#) dans le AWS CloudFormation Guide de l'utilisateur.

Utilisation de AWS Lambda avec CloudFront Lambda@Edge

Lambda@Edge vous permet d'exécuter des fonctions Lambda Node.js et Python 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)



Note

Lambda @Edge prend en charge un ensemble limité d'environnements d'exécution et de fonctionnalités. Pour plus d'informations, consultez [Exigences et restrictions relatives aux fonctions Lambda](#) dans le manuel du développeur Amazon CloudFront.

Vous pouvez également générer des réponses pour les utilisateurs sans jamais envoyer de demande à l'origine.

Example Événement de message CloudFront

```
{  
  "Records": [  
    {  
      "cf": {  
        "config": {  
          "distributionId": "EDFDVBD6EXAMPLE"  
        },  
        "request": {  
          "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",  
          "hostHeader": "www.example.com",  
          "httpMethod": "GET",  
          "uri": "/image.png"  
        }  
      }  
    }  
  ]  
}
```

```
"method": "GET",
"uri": "/picture.jpg",
"headers": [
    "host": [
        {
            "key": "Host",
            "value": "d111111abcdef8.cloudfront.net"
        }
    ],
    "user-agent": [
        {
            "key": "User-Agent",
            "value": "curl/7.51.0"
        }
    ]
}
]
```

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](#).

Utilisation de AWS Lambda avec 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.

Example Événement de message AWS CodeCommit

```
{
    "Records": [
        {
            "awsRegion": "us-east-2",
            "codecommit": {

```

```

        "references": [
            {
                "commit": "5e493c6f3067653f3d04eca608b4901eb227078",
                "ref": "refs/heads/master"
            }
        ],
        "eventId": "31ade2c7-f889-47c5-a937-1cf99e2790e9",
        "eventName": "ReferenceChanges",
        "eventPartNumber": 1,
        "eventSource": "aws:codecommit",
        "eventSourceARN": "arn:aws:codecommit:us-east-2:123456789012:lambda-pipeline-
repo",
        "eventTime": "2019-03-12T20:58:25.400+0000",
        "eventTotalParts": 1,
        "eventTriggerConfigId": "0d17d6a4-efeb-46f3-b3ab-a63741badeb8",
        "eventTriggerName": "index.handler",
        "eventVersion": "1.0",
        "userIdentityARN": "arn:aws:iam::123456789012:user/intern"
    }
]
}

```

Pour plus d'informations, consultez [Gestion des déclencheurs pour un référentiel AWS CodeCommit](#).

Utilisation d'AWS Lambda avec AWS CodePipeline

AWS CodePipeline est un service qui vous permet de créer des pipelines de diffusion en continu pour les applications exécutées sur AWS. Vous pouvez créer un pipeline pour déployer votre application Lambda. Vous pouvez également créer un pipeline pour appeler une fonction Lambda afin d'effectuer une tâche lors de l'exécution du pipeline. Lorsque vous [créez une application Lambda \(p. 136\)](#) dans la console Lambda, Lambda crée un pipeline qui inclut les étapes liées à la source, à la génération et au déploiement.

CodePipeline appelle votre fonction de manière asynchrone avec un événement qui contient des détails sur la tâche. L'exemple suivant montre un événement d'un pipeline qui a appelé une fonction nommée `my-function`.

Example CodePipeline event

```
{
    "CodePipeline.job": {
        "id": "c0d76431-b0e7-xmpl-97e3-e8ee786eb6f6",
        "accountId": "123456789012",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "my-function",
                    "UserParameters": "{\"KEY\": \"VALUE\"}"
                }
            },
            "inputArtifacts": [
                {
                    "name": "my-pipeline-SourceArtifact",
                    "revision": "e0c7xmpl2308ca3071aa7bab414de234ab52eea",
                    "location": {
                        "type": "S3",
                        "s3Location": {
                            "bucketName": "aws-us-west-2-123456789012-my-pipeline",
                            "objectKey": "my-pipeline/test-api-2/TdOSFRV"
                        }
                    }
                }
            ]
        }
    }
}
```

```
        }
    ],
    "outputArtifacts": [
        {
            "name": "invokeOutput",
            "revision": null,
            "location": {
                "type": "S3",
                "s3Location": {
                    "bucketName": "aws-us-west-2-123456789012-my-pipeline",
                    "objectKey": "my-pipeline/invokeOutp/D0YHsJn"
                }
            }
        }
    ],
    "artifactCredentials": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "secretAccessKey": "6CGtmAa3lzWtV7a....",
        "sessionToken": "IQoJb3JpZ2luX2VjEA....",
        "expirationTime": 1575493418000
    }
}
}
```

Pour terminer la tâche, la fonction doit appeler l'API CodePipeline pour signaler la réussite ou l'échec. L'exemple de fonction Node.js suivant utilise l'opération `PutJobSuccessResult` pour signaler la réussite. Il obtient l'ID de tâche pour l'appel d'API à partir de l'objet événement.

Example index.js

```
var AWS = require('aws-sdk')
var codepipeline = new AWS.CodePipeline()

exports.handler = async (event) => {
    console.log(JSON.stringify(event, null, 2))
    var jobId = event["CodePipeline.job"].id
    var params = {
        jobId: jobId
    }
    return codepipeline.putJobSuccessResult(params).promise()
}
```

Pour l'appel asynchrone, Lambda met le message en file d'attente et fait une [nouvelle tentative \(p. 111\)](#) si votre fonction renvoie une erreur. Configurez votre fonction avec une [destination \(p. 96\)](#) pour conserver les événements que la fonction n'a pas pu traiter.

Pour plus d'informations sur la configuration d'un pipeline pour appeler une fonction Lambda, consultez [Appel d'une fonction AWS Lambda dans un pipeline](#) dans le AWS CodePipeline Guide de l'utilisateur.

Sections

- [Autorisations \(p. 200\)](#)
- [Création d'un pipeline de livraison continue pour une application Lambda avec AWS CodePipeline \(p. 201\)](#)

Autorisations

Pour appeler une fonction, un pipeline CodePipeline a besoin d'une autorisation pour utiliser les opérations d'API suivantes :

- [ListFunctions \(p. 581\)](#)
- [InvokeFunction \(p. 565\)](#)

Le [rôle de service de pipeline](#) par défaut inclut ces autorisations.

Pour terminer une tâche, la fonction a besoin des autorisations suivantes dans son [rôle d'exécution \(p. 33\)](#).

- `codepipeline:PutJobSuccessResult`
- `codepipeline:PutJobFailureResult`

Ces autorisations sont incluses dans la stratégie gérée [AWSCodePipelineCustomActionAccess](#).

Création d'un pipeline de livraison continue pour une application Lambda avec AWS CodePipeline

Vous pouvez utiliser AWS CodePipeline pour créer un pipeline de livraison continue pour votre application Lambda. CodePipeline combine des ressources de contrôle de code source, de génération et de déploiement afin de créer un pipeline qui s'exécute chaque fois que vous apportez une modification pour le code source de votre application.

Dans ce didacticiel, vous allez créer les ressources suivantes.

- Référentiel – Un référentiel Git dans AWS CodeCommit. Lorsque vous envoyez une modification, le pipeline copie le code source dans un compartiment Amazon S3 et le transmet au projet de génération.
- Projet de génération – Une génération AWS CodeBuild qui obtient le code source du pipeline et conditionne l'application en package. Le code source inclut une spécification de génération avec des commandes qui installent des dépendances et préparent un modèle Modèle d'application sans serveur AWS (AWS SAM) à déployer.
- Configuration de déploiement – L'étape de déploiement du pipeline définit un ensemble d'actions qui prennent le modèle AWS SAM à partir de la sortie de génération, créent un jeu de modifications dans AWS CloudFormation et exécutent le jeu de modifications pour mettre à jour la pile AWS CloudFormation de l'application.
- Pile AWS CloudFormation – L'étape de déploiement de pile utilise un modèle pour créer une pile dans AWS CloudFormation. Le modèle est un document au format YAML qui définit les ressources de l'application Lambda. L'application inclut une fonction Lambda et une API Amazon API Gateway qui l'appelle.
- Rôles – Le pipeline, la génération et le déploiement présentent chacun un rôle de service qui leur permet de gérer les ressources AWS. La console crée le pipeline et les rôles de génération lors de la création de ces ressources. Vous créez le rôle qui permet à AWS CloudFormation de gérer la pile d'applications.

Le pipeline mappe une seule branche dans un référentiel vers une seule pile AWS CloudFormation. Vous pouvez créer d'autres pipelines pour ajouter des environnements pour d'autres branches dans le même référentiel. Vous pouvez également ajouter des étapes à votre pipeline pour le test, l'étape intermédiaire et des approbations manuelles. Pour plus d'informations sur AWS CodePipeline, consultez [Qu'est-ce que AWS CodePipeline ?](#)

Pour une autre méthode de création d'un pipeline avec Modèle d'application sans serveur AWS et AWS CloudFormation, consultez [Automatissez vos déploiements d'applications sans serveur](#) sur la chaîne YouTube Amazon Web Services.

Sections

- [Prérequis \(p. 202\)](#)
- [Créer un rôle AWS CloudFormation \(p. 202\)](#)

- [Configurer un référentiel \(p. 203\)](#)
- [Crée un pipeline. \(p. 204\)](#)
- [Mettre à jour le rôle de l'étape de génération \(p. 205\)](#)
- [Réaliser l'étape de déploiement \(p. 205\)](#)
- [Tester l'application \(p. 206\)](#)

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Au cours de l'étape de génération, le script de génération charge des artefacts dans Amazon Simple Storage Service (Amazon S3). Vous pouvez utiliser un compartiment existant ou en créer un nouveau pour le pipeline. Utilisez l'AWS CLI pour créer un compartiment.

```
$ aws s3 mb s3://lambda-deployment-artifacts-123456789012
```

Créer un rôle AWS CloudFormation

Créez un rôle qui autorise AWS CloudFormation à accéder aux ressources AWS.

Pour créer un rôle AWS CloudFormation

1. Ouvrez la page [Rôles](#) dans la console IAM.
2. Choisissez [Créer un rôle](#).
3. Créez un rôle avec les propriétés suivantes :
 - Entité de confiance – AWS CloudFormation
 - Autorisations – AWSLambdaExecute
 - Nom du rôle – **cfn-lambda-pipeline**
4. Ouvrez le rôle. Sous l'onglet Permissions (Autorisations), choisissez Add inline policy (Ajouter une stratégie en ligne).
5. Dans [Créer une stratégie](#), cliquez sur l'onglet JSON et ajoutez la stratégie suivante.

```
{  
    "Statement": [  
        {
```

```
"Action": [
    "apigateway:*",
    "codedeploy:*",
    "lambda:*",
    "cloudformation:CreateChangeSet",
    "iam:GetRole",
    "iam:CreateRole",
    "iam:DeleteRole",
    "iam:PutRolePolicy",
    "iam:AttachRolePolicy",
    "iam:DeleteRolePolicy",
    "iam:DetachRolePolicy",
    "iam:PassRole",
    "s3:GetObject",
    "s3:GetObjectVersion",
    "s3:GetBucketVersioning"
],
"Resource": "*",
"Effect": "Allow"
},
],
"Version": "2012-10-17"
}
```

Configurer un référentiel

Créez un référentiel AWS CodeCommit pour stocker vos fichiers de projet. Pour plus d'informations, consultez [Configuration](#) dans le guide de l'utilisateur CodeCommit.

Pour créer un référentiel

1. Ouvrez la [console Outils de développement](#).
2. Dans Source, choisissez Référentiels.
3. Choisissez Create repository.
4. Suivez les instructions pour créer et cloner un référentiel nommé **lambda-pipeline-repo**.

Créez les fichiers suivants dans le dossier du référentiel.

Example index.js

Fonction Lambda qui renvoie l'heure actuelle.

```
var time = require('time');
exports.handler = (event, context, callback) => {
    var currentTime = new time.Date();
    currentTime.setTz("America/Los_Angeles");
    callback(null, {
        statusCode: '200',
        body: 'The time in Los Angeles is: ' + currentTime.toString(),
    });
};
```

Example template.yml

Modèle AWS SAM ([p. 28](#)) qui définit l'application.

```
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: nodejs10.x
      CodeUri: .
  Events:
    MyTimeApi:
      Type: Api
      Properties:
        Path: /TimeResource
        Method: GET
```

Example buildspec.yml

Une [spécification de génération AWS CodeBuild](#) qui installe les packages requis et charge le package de déploiement dans Amazon S3. Remplacez le texte en surbrillance par le nom de votre compartiment.

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 10
  build:
    commands:
      - npm install time
      - export BUCKET=lambda-deployment-artifacts-123456789012
      - aws cloudformation package --template-file template.yml --s3-bucket $BUCKET --
output-template-file outputtemplate.yml
artifacts:
  type: zip
  files:
    - template.yml
    - outputtemplate.yml
```

Validez et envoyez les fichiers vers CodeCommit.

```
~/lambda-pipeline-repo$ git add .
~/lambda-pipeline-repo$ git commit -m "project files"
~/lambda-pipeline-repo$ git push
```

Crée un pipeline.

Créez un pipeline qui déploie votre application. Le pipeline surveille votre référentiel pour identifier les modifications, exécute une génération AWS CodeBuild pour créer un package de déploiement et déploie l'application avec AWS CloudFormation. Pendant le processus de création du pipeline, vous créez également le projet de génération AWS CodeBuild.

Pour créer un pipeline

1. Ouvrez la [console Outils de développement](#).
2. Sous Pipeline, choisissez Pipelines.
3. Choisissez Create pipeline.
4. Configurez les paramètres du pipeline, puis choisissez Suivant.
 - Nom du pipeline – **lambda-pipeline**

- Rôle de service – Nouveau rôle de service
 - Magasin d'artefacts – Emplacement par défaut
5. Configurez les paramètres de l'étape source, puis choisissez Suivant.
- Fournisseur de la source – AWS CodeCommit
 - Nom du référentiel – lambda-pipeline-repo
 - Nom de la branche – master
 - Options de détection des modifications – Amazon CloudWatch Events
6. Dans Fournisseur de génération, choisissez AWS CodeBuild, puis Créez un projet.
7. Configurez les paramètres du projet de génération, puis choisissez Continuer vers CodePipeline.
- Nom du projet – **lambda-pipeline-build**
 - Système d'exploitation – Ubuntu
 - Runtime – Standard
 - Version du runtime – aws/codebuild/standard:2.0
 - Version d'image – Dernière
 - Nom du fichier buildspec – **buildspec.yml**
8. Choisissez Suivant.
9. Configurez les paramètres de l'étape de déploiement, puis choisissez Suivant.
- Fournisseur de déploiement – AWS CloudFormation
 - Mode d'action – Créez ou remplacer un jeu de modifications
 - Nom de la pile – lambda-pipeline-stack
 - Nom du jeu de modifications – lambda-pipeline-changeset
 - Template (Modèle) – **BuildArtifact::outputtemplate.yml**
 - Capacités – CAPABILITY_IAM, CAPABILITY_AUTO_EXPAND
 - Nom du rôle – cfn-lambda-pipeline
10. Choisissez Create pipeline.

Le pipeline échoue la première fois qu'il s'exécute, car il a besoin d'autorisations supplémentaires. Dans la section suivante, vous ajoutez des autorisations au rôle qui est généré pour votre étape de génération.

Mettre à jour le rôle de l'étape de génération

Au cours de l'étape de génération, AWS CodeBuild a besoin d'une autorisation pour charger la sortie de génération dans votre compartiment Amazon S3.

Pour mettre à jour le rôle

1. Ouvrez la page [Rôles](#) dans la console IAM.
2. Choisissez code-build-lambda-pipeline-service-role.
3. Choisissez Attach Policies (Attacher des stratégies).
4. Attachez AmazonS3FullAccess.

Réaliser l'étape de déploiement

L'étape de déploiement a une action qui crée un jeu de modifications pour la pile AWS CloudFormation qui gère votre application Lambda. Un jeu de modifications spécifie les modifications qui ont été apportées à la pile, telles que l'ajout de nouvelles ressources et la mise à jour des ressources existantes. Les ensembles

de modifications vous permettent de prévisualiser les modifications qui vont être apportées avant de les exécuter, et d'ajouter les étapes d'approbation. Ajoutez une seconde action qui exécute le jeu de modifications pour terminer le déploiement.

Mise à jour de l'étape de déploiement

1. Ouvrez votre pipeline dans la [console Outils de développement](#).
2. Choisissez Edit.
3. En regard de Déployer, choisissez Modifier l'étape.
4. Choisissez Ajouter un groupe d'actions.
5. Configurez les paramètres de l'étape de déploiement, puis choisissez Suivant.
 - Nom de l'action – **execute-changeset**
 - Fournisseur d'action – AWS CloudFormation
 - Artefacts d'entrée – BuildArtifact
 - Mode d'action – Exécuter un jeu de modifications
 - Nom de la pile – lambda-pipeline-stack
 - Nom du jeu de modifications – lambda-pipeline-changeset
6. Sélectionnez Done.
7. Choisissez Enregistrer.
8. Choisissez Publier une modification pour exécuter le pipeline.

Votre pipeline est prêt. Envoyez les modifications vers la branche maître pour déclencher un déploiement.

Tester l'application

L'application inclut une API API Gateway avec un point de terminaison public qui renvoie l'heure actuelle. Utilisez la console Lambda pour afficher l'application et accéder à l'API.

Pour tester l'application

1. Ouvrez la [page Applications](#) de la console Lambda.
2. Choisissez lambda-pipeline-stack.
3. Sous Resources (Ressources), développez ServerlessRestApi.
4. Choisissez Prod API endpoint (Point de terminaison d'API Prod).
5. Ajoutez **/TimeResource** à la fin de l'URL. Par exemple, <https://l1193nqxdjj.execute-api.us-east-2.amazonaws.com/Prod/TimeResource>.
6. Ouvrez l'URL.

L'API renvoie l'heure actuelle dans le format suivant.

```
The time in Los Angeles is: Thu Jun 27 2019 16:07:20 GMT-0700 (PDT)
```

Utilisation de AWS Lambda avec 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 [Introducing Amazon Cognito Events: Sync Triggers](#) dans le blog Mobile Development.

Example Événement de message 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
}
```

Vous configurez un mappage de source d'événement à l'aide de la configuration des abonnements aux événements Amazon Cognito. Pour plus d'informations 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.

Utilisation de AWS Lambda avec 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.

Example Événement de message 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",  
    "ruleParameters": {},  
    "resultToken": null  
}
```

```
    "configRuleName": "change-triggered-config-rule",
    "configRuleId": "config-rule-0123456",
    "accountId": "012345678912",
    "version": "1.0"
}
```

Pour plus d'informations, consultez [Évaluation des ressources avec les règles AWS Config](#).

Utilisation de AWS Lambda avec Amazon DynamoDB

Vous pouvez utiliser une fonction AWS Lambda pour traiter des enregistrements dans un [flux Amazon DynamoDB](#). 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 \(p. 92\)](#) 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.

Par défaut, Lambda appelle votre fonction dès que des enregistrements sont disponibles dans le flux. Si le lot qu'il lit depuis le flux ne comporte qu'un enregistrement, Lambda n'envoie qu'un enregistrement à la fonction. Pour éviter d'appeler la fonction avec un petit nombre d'enregistrements, vous pouvez demander à la source d'événement de les mettre en mémoire tampon pendant 5 minutes maximum en configurant une fenêtre de lot. Avant d'appeler la fonction, Lambda continue de lire les enregistrements du flux jusqu'à avoir collecté un lot complet ou jusqu'à l'expiration de la fenêtre de lot.

Si votre fonction renvoie une erreur, Lambda retente de traiter le lot jusqu'à ce que le traitement réussisse ou que les données expirent. Pour éviter les partitions bloquées, vous pouvez configurer le mappage de source d'événement pour réessayer avec une taille de lot plus petite, limiter le nombre de nouvelles tentatives ou supprimer les enregistrements qui sont trop vieux. Afin de conserver les événements supprimés, vous pouvez configurer le mappage de source d'événement pour envoyer les informations détaillées sur les lots ayant échoué à une file d'attente SQS ou à une rubrique SNS.

Vous pouvez également augmenter la simultanéité en traitant plusieurs lots à partir de chaque partition en parallèle. Lambda peut traiter jusqu'à 10 lots dans chaque partition simultanément. Si vous augmentez le nombre de lots simultanés par partition, Lambda garantit toujours le traitement dans l'ordre au niveau de la clé de partition.

Sections

- [Autorisations du rôle d'exécution \(p. 210\)](#)
- [Configuration d'un flux comme source d'événement \(p. 210\)](#)
- [API de mappage de la source d'événement \(p. 211\)](#)
- [Gestion des erreurs \(p. 213\)](#)
- [Métriques Amazon CloudWatch \(p. 214\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB \(p. 214\)](#)
- [Exemple de code de fonction \(p. 219\)](#)
- [Modèle AWS SAM pour une application DynamoDB \(p. 222\)](#)

Autorisations du rôle d'exécution

Lambda a besoin des autorisations suivantes pour gérer les ressources liées à votre flux DynamoDB. Ajoutez-les au rôle d'exécution de votre fonction.

- [dynamodb:DescribeStream](#)
- [dynamodb:GetRecords](#)
- [dynamodb:GetShardIterator](#)
- [dynamodb>ListStreams](#)

La stratégie gérée `AWSLambdaDynamoDBExecutionRole` inclut ces autorisations. Pour en savoir plus, consultez [Rôle d'exécution AWS Lambda \(p. 33\)](#).

Pour envoyer les enregistrements des lots ayant échoué à une file d'attente ou à une rubrique, votre fonction a besoin d'autorisations supplémentaires. Chaque service de destination nécessite une autorisation différente, comme suit :

- Amazon SQS – [sns:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Configuration d'un flux comme 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 pour 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. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).
4. Choisissez un type de déclencheur.
5. Configurez les options requises, puis choisissez Ajouter.

Lambda prend en charge les options suivantes pour les sources d'événements DynamoDB.

Options de source d'événement

- Table DynamoDB– La table DynamoDB à partir de laquelle lire les enregistrements.
- Taille de lot – Nombre d'enregistrements à envoyer à la fonction dans chaque lot, jusqu'à 1 000. 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 \(p. 30\)](#) pour un appel synchrone (6 MB).
- Batch window (Fenêtre de lot) – Spécifiez l'intervalle de temps maximal pour collecter des enregistrements avant d'appeler la fonction, en secondes.
- Starting position (Position de départ) : traitez uniquement les nouveaux enregistrements, ou tous enregistrement existants.
 - Dernier : 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.

- On-failure destination (Destination en cas d'échec) – File d'attente SQS ou rubrique SNS pour les enregistrements qui ne peuvent pas être traités. Lorsque Lambda supprime un lot d'enregistrements parce qu'il est trop vieux ou parce que toutes les nouvelles tentatives ont été épuisées, il envoie les détails sur ce lot à la file d'attente ou à la rubrique.
- Retry attempts (Nouvelles tentatives) – Nombre maximal de nouvelles tentatives effectuées par Lambda lorsqu'une fonction renvoie une erreur. Cela ne s'applique pas aux limitations ou erreurs de service où le lot n'a pas atteint la fonction.
- Maximum age of record (Âge maximal de l'enregistrement) – Âge maximal d'un enregistrement que Lambda envoie à votre fonction.
- Split batch on error (Fractionner le lot en cas d'erreur) – Lorsque la fonction renvoie une erreur, fractionnez le lot en deux avant de réessayer.
- Concurrent batches per shard (Lots simultanés par partition) – Traitez plusieurs lots simultanément à partir de la même partition.
- 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.

Note

DynamoDB facture les demandes de lecture effectuées par Lambda pour obtenir des enregistrements à partir du flux. Pour plus d'informations sur la tarification, consultez [Tarification Amazon DynamoDB](#).

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 gérer les mappages de source d'événement avec l'AWS CLI ou les kits SDK AWS, utilisez les actions d'API suivantes :

- [CreateEventSourceMapping \(p. 498\)](#)
- [ListEventSourceMappings \(p. 575\)](#)
- [GetEventSourceMapping \(p. 534\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)
- [DeleteEventSourceMapping \(p. 515\)](#)

L'exemple suivant utilise l'AWS CLI pour mapper une fonction nommée `my-function` à un flux DynamoDB spécifié par son Amazon Resource Name (ARN), avec une taille de lot égale à 500.

```
$ aws lambda create-event-source-mapping --function-name my-function --batch-size 500 --starting-position LATEST \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/stream/2019-06-10T19:26:16.525
{
    "UUID": "14e0db71-5d35-4eb5-b481-8945cf9d10c2",
    "BatchSize": 500,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/stream/2019-06-10T19:26:16.525",
```

```

    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1560209851.963,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "User action",
    "DestinationConfig": {},
    "MaximumRecordAgeInSeconds": 604800,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 10000
}

```

Configurez des options supplémentaires pour personnaliser la manière dont les lots sont traités et pour indiquer quand supprimer les enregistrements qui ne peuvent pas être traités. L'exemple suivant met à jour un mappage de source d'événement pour envoyer un enregistrement d'échec à une file d'attente SQS après deux nouvelles tentatives ou si les enregistrements datent de plus d'une heure.

```

$ aws lambda update-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 --maximum-record-age-in-seconds 3600
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-2:123456789012:dlq"}}'
{
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
    "BatchSize": 100,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1573243620.0,
    "LastProcessingResult": "PROBLEM: Function call failed",
    "State": "Updating",
    "StateTransitionReason": "User action",
    "DestinationConfig": {},
    "MaximumRecordAgeInSeconds": 604800,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 10000
}

```

Les paramètres mis à jour sont appliqués de façon asynchrone et ne sont pas reflétés dans la sortie tant que le processus n'est pas terminé. Utilisez la commande `get-event-source-mapping` pour afficher le statut actuel.

```

$ aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
{
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
    "BatchSize": 100,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1573244760.0,
    "LastProcessingResult": "PROBLEM: Function call failed",
    "State": "Enabled",
    "StateTransitionReason": "User action",
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "arn:aws:sqs:us-east-2:123456789012:dlq"
        }
    },
    "MaximumRecordAgeInSeconds": 3600,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 2
}

```

}

Pour traiter plusieurs lots simultanément, utilisez l'option `--parallelization-factor`.

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

Gestion des erreurs

Le mappage de source d'événement qui lit les enregistrements à partir de votre flux DynamoDB appelle votre fonction de façon synchrone et réessaie en cas d'erreur. Si la fonction est limitée ou si le service Lambda renvoie une erreur sans appeler la fonction, Lambda réessaie jusqu'à ce que les enregistrements expirent ou dépassent l'âge maximal que vous configurez dans le mappage de source d'événement.

Si la fonction reçoit les enregistrements mais renvoie une erreur, Lambda réessaie jusqu'à ce que les enregistrements du lot expirent, dépassent l'âge maximal ou atteignent la limite de nouvelles tentatives configurée. Pour les erreurs de fonction, vous pouvez également configurer le mappage de source d'événement pour fractionner un lot ayant échoué en deux lots. Le fait de réessayer avec des lots plus petits permet d'isoler les enregistrements incorrects et de contourner les problèmes de dépassement de délai. Le fractionnement d'un lot n'est pas comptabilisé dans la limite de nouvelles tentatives.

Si les mesures de gestion des erreurs échouent, Lambda supprime les enregistrements et continue de traiter les lots à partir du flux. Avec les paramètres par défaut, cela signifie qu'un enregistrement incorrect peut bloquer le traitement sur la partition concernée pendant jusqu'à one day. Pour éviter cela, configurez le mappage de source d'événement de votre fonction avec un nombre raisonnable de nouvelles tentatives et un âge maximal des enregistrements adapté à votre cas d'utilisation.

Pour conserver un enregistrement des lots supprimés, configurez une destination en cas d'échec. Lambda envoie un document à la rubrique ou à la file d'attente de cette destination avec des détails sur le lot.

Pour configurer une destination pour enregistrer les événements ayant échoué.

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Stream invocation (Appel de flux).
5. Pour Stream (Flux), choisissez un flux qui est mappé à la fonction.
6. Pour Type de destination, choisissez le type de ressource qui reçoit l'enregistrement d'appel.
7. Pour Destination, choisissez une ressource.
8. Choisissez Enregistrer.

L'exemple suivant illustre un enregistrement d'appel pour un flux DynamoDB.

Example Enregistrement d'appel

```
{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted",
    "approximateInvokeCount": 1
  },
  "responseContext": {
    "statusCode": 200,
```

```
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:13:49.717Z",
    "DDBStreamBatchInfo": {
        "shardId": "shardId-00000001573689847184-864758bb",
        "startSequenceNumber": "800000000003126276362",
        "endSequenceNumber": "800000000003126276362",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",
        "batchSize": 1,
        "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/
stream/2019-11-14T00:04:06.388"
    }
}
```

Vous pouvez utiliser ces informations pour récupérer les enregistrements concernés à partir du flux afin de résoudre les problèmes. Les enregistrements réels ne sont pas inclus. Vous devez donc traiter cet enregistrement et les récupérer à partir du flux avant qu'ils expirent et soient perdus.

Métriques 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 de plus amples informations, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution](#) (p. 33) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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. 219).

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 nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-dynamodb-role
```

Tester la fonction Lambda

Dans cette étape, vousappelez la fonction Lambda manuellement via la commande CLI `aws lambda 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": 26  
            }  
        }  
    ]  
}
```

```
        "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
outfile.txt
```

La fonction renvoie la chaîne `message` dans le corps de la réponse.

Vérifiez la sortie dans le fichier `outfile.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](#).

Ajouter 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` (indique qu'AWS Lambda est parvenu à lire les enregistrements à partir du flux et à appeler votre 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
```

Tester 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 dans 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 \(p. 219\)](#)
- [Java 11 \(p. 219\)](#)
- [C# \(p. 220\)](#)
- [Python 3 \(p. 221\)](#)
- [Go \(p. 221\)](#)

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");
};
```

Comptez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

Java 11

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.LambdaLogger;
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.

Dépendances

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

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());
    }
}
```

Remplacez le fichier `Program.cs` d'un projet .NET Core par l'exemple ci-dessus. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans C# \(p. 414\)](#).

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'])))
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

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())
            }
        }
    }
}
```

Générez l'exécutable avec `go build` et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Go \(p. 398\)](#).

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 \(p. 214\)](#). 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 Amazon EC2

Vous pouvez utiliser AWS Lambda pour traiter les événements du cycle de vie à partir de Amazon Elastic Compute Cloud et gérer les ressources Amazon EC2. Amazon EC2 envoie des événements à Amazon CloudWatch Events pour des événements du cycle de vie notamment lorsqu'une instance change d'état, lorsqu'un instantané de volume Amazon Elastic Block Store est terminé ou lorsqu'une instance Spot est planifiée pour être interrompue. Vous configurez CloudWatch Events pour transférer ces événements à une fonction Lambda pour leur traitement.

CloudWatch Events appelle votre fonction Lambda de manière asynchrone avec le document d'événement de Amazon EC2.

Example événement du cycle de vie d'une instance

```
{  
    "version": "0",  
    "id": "b6ba298a-7732-2226-xmpl-976312c1a050",  
    "detail-type": "EC2 Instance State-change Notification",  
    "source": "aws.ec2",  
    "account": "123456798012",  
    "time": "2019-10-02T17:59:30Z",  
    "region": "us-east-2",  
    "resources": [  
        "arn:aws:ec2:us-east-2:123456798012:instance/i-0c314xmplcd5b8173"  
    ],  
    "detail": {  
        "instance-id": "i-0c314xmplcd5b8173",  
        "state": "running"  
    }  
}
```

Pour plus d'informations sur la configuration des événements dans CloudWatch Events, veuillez consulter [Utilisation de AWS Lambda avec Amazon CloudWatch Events \(p. 188\)](#). Pour obtenir un exemple de fonction qui traite les notifications d'instantanés Amazon EBS, consultez [Amazon CloudWatch Events pour Amazon EBS](#) dans le Amazon EC2 Guide de l'utilisateur pour les instances Linux.

Vous pouvez également utiliser le kit SDK AWS pour gérer des instances et d'autres ressources avec l'API Amazon EC2. Pour obtenir un didacticiel avec un exemple d'application en C #, consultez [Didacticiel : Utiliser Kit AWS SDK pour .NET pour gérer des instances Spot Amazon EC2 \(p. 224\)](#).

Autorisations

Pour traiter les événements du cycle de vie à partir de Amazon EC2, CloudWatch Events a besoin de l'autorisation pour appeler votre fonction. Cette autorisation provient de la [stratégie basée sur les ressources \(p. 36\)](#) de la fonction. Si vous utilisez la console CloudWatch Events pour configurer un déclencheur d'événement, la console met à jour la stratégie basée sur les ressources en votre nom. Sinon, ajoutez une instruction comme suit :

Example déclaration de stratégie basée sur les ressources pour les notifications de cycle de vie Amazon EC2

```
{  
    "Sid": "ec2-events",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "events.amazonaws.com"  
    },  
    "Action": "lambda:InvokeFunction",  
    "Resource": "arn:aws:lambda:us-east-2:12456789012:function:my-function",  
    "Condition": {  
        "ArnLike": {  
            "AWS:SourceArn": "arn:aws:events:us-east-2:12456789012:rule/*"  
        }  
    }  
}
```

Pour ajouter une instruction, utilisez la commande add-permission AWS CLI.

```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
```

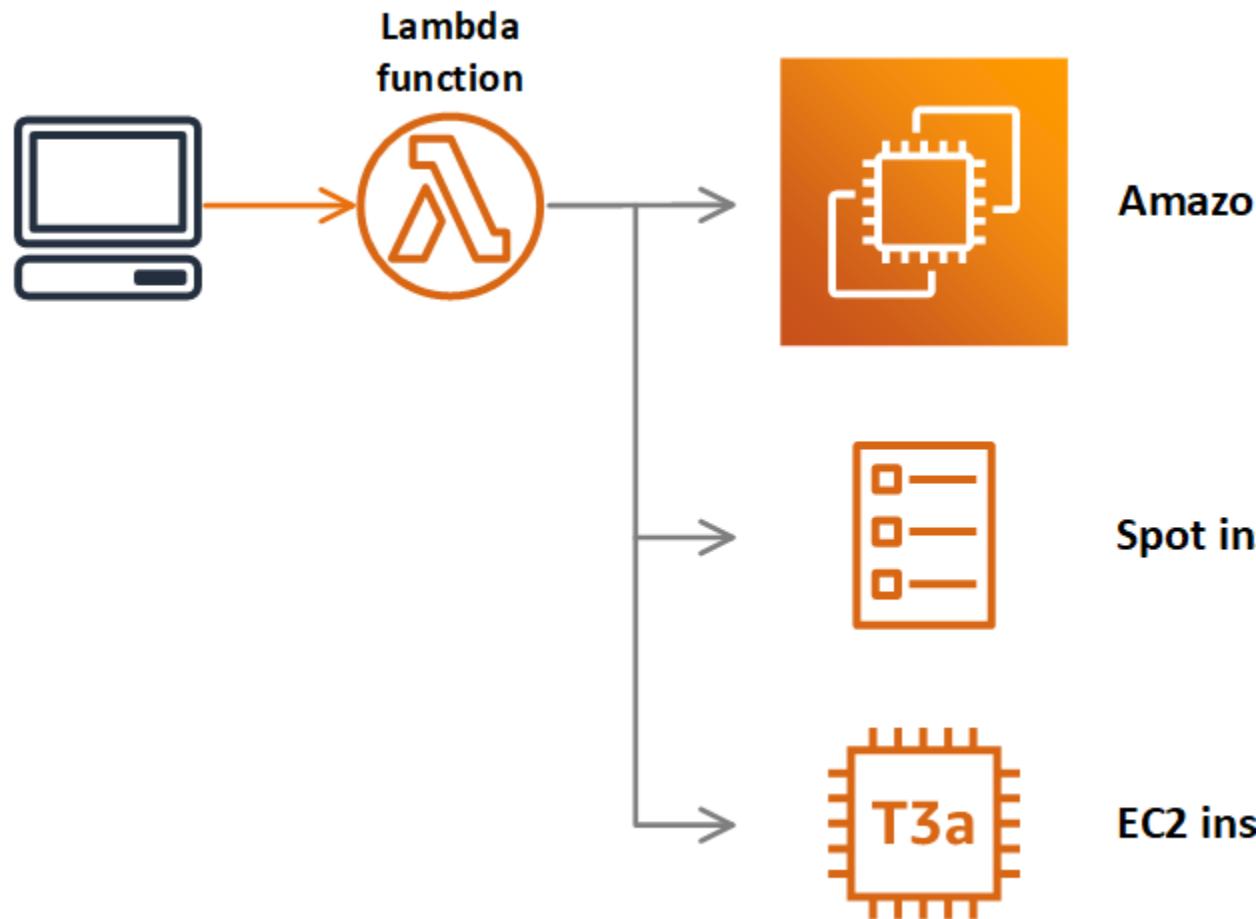
```
--principal events.amazonaws.com --function-name my-function --source-arn  
'arn:aws:events:us-east-2:12456789012:rule/*'
```

Si votre fonction utilise le kit SDK AWS pour gérer les ressources Amazon EC2, ajoutez des autorisations Amazon EC2 au rôle d'exécution (p. 33) de la fonction.

Didacticiel : Utiliser Kit AWS SDK pour .NET pour gérer des instances Spot Amazon EC2

Vous pouvez utiliser le Kit AWS SDK pour .NET pour gérer des instances Spot Amazon EC2 avec du code C#. Le kit SDK vous permet d'utiliser l'API Amazon EC2 pour créer des demandes d'instance Spot, déterminer le moment où la demande est exécutée, supprimer des demandes et identifier les instances créées.

Ce didacticiel fournit du code qui exécute ces tâches et un exemple d'application que vous pouvez exécuter localement ou sur AWS. Il inclut un exemple de projet que vous pouvez déployer dans l'environnement d'exécution AWS Lambda.NET Core 2.1.



Pour plus d'informations sur l'utilisation des instances Spot et les meilleures pratiques, consultez [Instances Spot](#) dans le guide de l'utilisateur Amazon EC2.

Prérequis

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Ce didacticiel utilise le code du référentiel GitHub du guide du développeur. Le référentiel contient également des scripts d'assistance et des fichiers de configuration nécessaires pour suivre ses procédures. Clonez le référentiel à l'adresse github.com/awsdocs/aws-lambda-developer-guide.

Pour utiliser l'exemple de code, vous avez besoin des outils suivants :

- AWS CLI – Pour déployer l'exemple d'application vers AWS, installez la [AWS CLI](#). La AWS CLI fournit également des informations d'identification pour l'exemple de code lorsque vous l'exécutez localement.
- Interface de ligne de commande .NET Core – Pour exécuter et tester le code localement, installez le [SDK .NET Core 2.1](#).
- Outil global .NET Core Lambda – Pour créer le package de déploiement pour Lambda, installez [l'outil global .NET Core](#) avec l'interface de ligne de commande .NET Core.

```
$ dotnet tool install -g Amazon.Lambda.Tools
```

Le code de ce didacticiel gère les demandes Spot qui lancent instances Amazon EC2. Pour exécuter le code localement, vous avez besoin d'informations d'identification SDK avec autorisation d'utiliser les API suivantes.

- `ec2:RequestSpotInstance`
- `ec2:GetSpotRequestState`
- `ec2:CancelSpotRequest`
- `ec2:TerminateInstances`

Pour exécuter l'exemple d'application dans AWS, vous avez besoin [d'une autorisation pour utiliser Lambda \(p. 32\)](#) et les services suivants.

- [AWS CloudFormation \(tarification\)](#)
- [Amazon Elastic Compute Cloud \(tarification\)](#)

Des frais standard s'appliquent pour chaque service.

Vérifier le code

Localisez l'exemple de projet dans le référentiel guide sous [sample-apps/ec2-spot](#). Ce répertoire contient le code de la fonction Lambda, les tests, les fichiers de projet, les scripts et un modèle AWS CloudFormation.

La Function classe inclut une méthode `FunctionHandler` qui appelle d'autres méthodes pour créer des requêtes Spot, vérifier leur état et nettoyer. Il crée un client Amazon EC2 avec le Kit AWS SDK pour .NET dans un constructeur statique pour lui permettre d'être utilisé dans toute la classe.

Example `Function.cs` – `FunctionHandler`

```
using Amazon.EC2;
...
public class Function
{
    private static AmazonEC2Client ec2Client;

    static Function()
    {
        AWSSDKHandler.RegisterXRayForAllServices();
        ec2Client = new AmazonEC2Client();
    }

    public async Task<string> FunctionHandler(Dictionary<string, string> input,
ILambdaContext context)
    {
        // More AMI IDs: aws.amazon.com/amazon-linux-2/release-notes/
// us-east-2 HVM EBS-Backed 64-bit Amazon Linux 2
        string ami = "ami-09d9edae5eb90d556";
        string sg = "default";
        InstanceType type = InstanceType.T3aNano;
        string price = "0.003";
        int count = 1;
        var requestSpotInstances = await RequestSpotInstance(ami, sg, type, price,
count);
        var spotRequestId =
requestSpotInstances.SpotInstanceRequests[0].SpotInstanceRequestId;
```

La méthode `RequestSpotInstance` crée une demande d'instance Spot.

Example `Function.cs` – `RequestSpotInstance`

```
using Amazon;
using Amazon.Util;
using Amazon.EC2;
using Amazon.EC2.Model;
...
public async Task<RequestSpotInstancesResponse> RequestSpotInstance(
    string amiId,
    string securityGroupName,
    InstanceType instanceType,
    string spotPrice,
    int instanceCount)
{
    var request = new RequestSpotInstancesRequest();

    var launchSpecification = new LaunchSpecification();
    launchSpecification.ImageId = amiId;
    launchSpecification.InstanceType = instanceType;
    launchSpecification.SecurityGroups.Add(securityGroupName);

    request.SpotPrice = spotPrice;
    request.InstanceCount = instanceCount;
    request.LaunchSpecification = launchSpecification;

    RequestSpotInstancesResponse response = await
ec2Client.RequestSpotInstancesAsync(request);
```

```
    return response;
}
...
```

Ensuite, il faut attendre que la demande Spot atteigne le statut Active avant de passer à la dernière étape. Pour déterminer l'état de votre demande Spot, utilisez la méthode [DescribeSpotInstanceRequests](#) pour obtenir l'état de l'ID de la demande Spot à surveiller.

```
public async Task<SpotInstanceRequest> GetSpotRequest(string spotRequestId)
{
    var request = new DescribeSpotInstanceRequestsRequest();
    request.SpotInstanceRequestIds.Add(spotRequestId);

    var describeResponse = await ec2Client.DescribeSpotInstanceRequestsAsync(request);

    return describeResponse.SpotInstanceRequests[0];
}
```

La dernière étape consiste à nettoyer les demandes et les instances. Il est important à la fois d'annuler toutes les demandes en cours et de résilier toutes les instances. Si vous annulez simplement vos demandes, cela ne résiliera pas vos instances, ce qui signifie que vous continuerez à être facturé pour elles. Lorsque vous résiliez vos instances, vos demandes Spot peuvent être annulées, mais dans certains cas, par exemple, si vous utilisez des demandes persistantes, la résiliation de vos instances ne sera pas suffisante pour empêcher votre demande d'être à nouveau satisfait. Par conséquent, annuler les demandes actives et résilier en même temps toutes les instances en cours d'exécution constitue une bonne pratique.

Utilisez la méthode [CancelSpotInstanceRequests](#) pour annuler une demande Spot. L'exemple suivant présente comment annuler une demande Spot.

```
public async Task CancelSpotRequest(string spotRequestId)
{
    Console.WriteLine("Canceling request " + spotRequestId);
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(spotRequestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}
```

Utilisez la méthode [TerminateInstances](#) pour résilier une instance.

```
public async Task TerminateSpotInstance(string instanceId)
{
    Console.WriteLine("Terminating instance " + instanceId);
    var terminateRequest = new TerminateInstancesRequest();
    terminateRequest.InstanceIds = new List<string>() { instanceId };
    try
    {
        var terminateResponse = await ec2Client.TerminateInstancesAsync(terminateRequest);
    }
    catch (AmazonEC2Exception ex)
    {
        // Check the ErrorCode to see if the instance does not exist.
        if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
        {
            Console.WriteLine("Instance {0} does not exist.", instanceId);
        }
        else
        {
            // The exception was thrown for another reason, so re-throw the exception.
        }
    }
}
```

```
        throw;
    }
}
```

Exécuter le code localement

Exécutez le code sur votre machine locale pour créer une demande d'instance Spot. Une fois la demande exécutée, le code supprime la demande et met fin à l'instance.

Pour exécuter le code de l'application

1. Accédez au répertoire `ec2Spot.Tests`.

```
$ cd test/ec2Spot.Tests
```

2. Utilisez l'interface de ligne de commande .NET pour exécuter les tests unitaires du projet.

```
test/ec2Spot.Tests$ dotnet test
Starting test execution, please wait...
sir-x5tgs5ij
open
open
open
open
open
active
Canceling request sir-x5tgs5ij
Terminating instance i-0b3fdff0e12e0897e
Complete

Test Run Successful.
Total tests: 1
    Passed: 1
Total time: 7.6060 Seconds
```

Le test unitaire appelle la méthode `FunctionHandler` pour créer une requête d'instance Spot, la surveiller et la nettoyer. Il est implémenté dans le cadre de test [xUnit.net](#).

Déployer l'application

Exécutez le code dans Lambda en tant que point de départ pour créer une application sans serveur.

Pour déployer et tester l'application

1. Définissez votre région sur `us-east-2`.

```
$ export AWS_DEFAULT_REGION=us-east-2
```

2. Créez un compartiment pour les artefacts de déploiement.

```
$ ./create-bucket.sh
make_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

3. Créez un package de déploiement et déployez l'application.

```
$ ./deploy.sh
```

```
Amazon Lambda Tools for .NET Core applications (3.3.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Executing publish command
...
Created publish archive (ec2spot.zip)
Lambda project successfully packaged: ec2spot.zip
Uploading to ebd38e401ce7d676d05d22b76f0209 1305107 / 1305107.0 (100.00%)
Successfully packaged artifacts and wrote output template to file out.yaml.
Execute the following command to deploy the packaged template
aws cloudformation deploy --template-file out.yaml --stack-name <YOUR STACK NAME>

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - ec2-spot
```

4. Ouvrez la [page Applications](#) de la console Lambda.

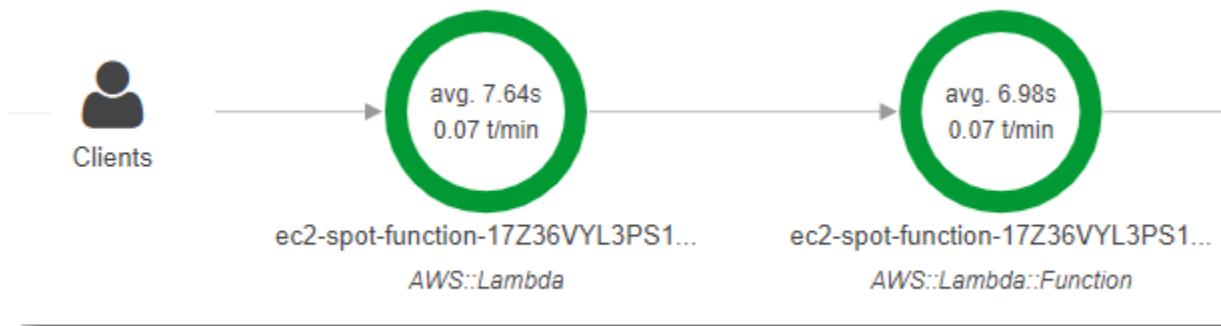
The screenshot shows the AWS Lambda console for the 'ec2-spot' function. At the top, there are three tabs: 'Overview' (highlighted in orange), 'Deployments', and 'Monitoring'. Below the tabs, a 'Getting started' section is visible. Under the 'Resources' heading, there is a table with two items:

| Logical ID | Physical ID | Type |
|------------|---------------------------------|-----------------|
| function | ec2-spot-function-17Z36VYL3PS14 | Lambda Function |
| role | ec2-spot-role-1TDCWJ2ZNNF1M | IAM Role |

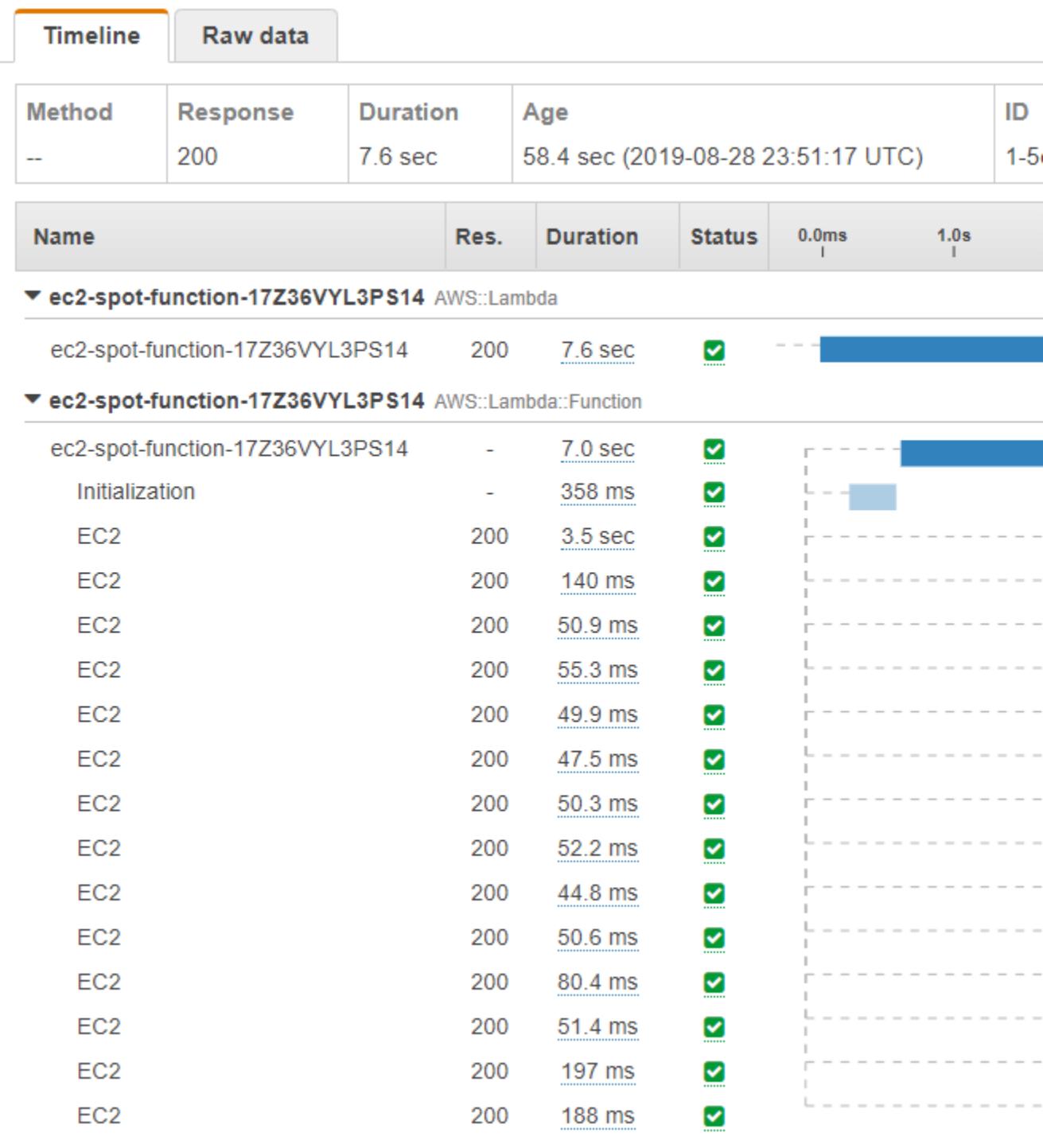
5. Sous Resources (Ressources), choisissez function (fonction).
6. Choisissez Test et créez un événement de test à partir du modèle par défaut.
7. Choisissez Test à nouveau pour appeler la fonction.

Affichez les journaux et les informations de suivi pour voir l'ID de demande Spot et la séquence d'appels vers Amazon EC2.

Pour afficher la carte de service, ouvrez la [page Carte de service](#) dans la console X-Ray.



Choisissez un nœud dans la carte de service, puis choisissez View traces (Afficher les suivis) pour afficher une liste des suivis. Choisissez un suivi dans la liste pour voir la chronologie des appels effectués par la fonction vers Amazon EC2.



Nettoyage

Le code fourni dans ce didacticiel est conçu pour créer et supprimer des demandes d'instance Spot et pour mettre fin aux instances qu'elles lancent. Toutefois, si une erreur se produit, les demandes et les instances

peuvent ne pas être nettoyées automatiquement. Affichez les demandes et les instances Spot dans la console Amazon EC2.

Confirmer que les ressources Amazon EC2 sont nettoyées

1. Ouvrez la [page Demandes Spot](#) dans la console Amazon EC2.
2. Vérifiez que l'état des demandes est Cancelled (Annulé).
3. Choisissez l'ID d'instance dans la colonne Capacity (Capacité) pour afficher l'instance.
4. Vérifiez que l'état des instances est Terminated (Terminé) ou Shutting down (Désactivé).

Pour nettoyer la fonction d'exemple et les ressources de support, supprimez sa pile AWS CloudFormation et le compartiment d'artefacts que vous avez créé.

```
$ ./cleanup.sh
Delete deployment artifacts and bucket (lambda-artifacts-63d5cbbf18fa5ecc)?y
delete: s3://lambda-artifacts-63d5cbbf18fa5ecc/ebd38e401cedd7d676d05d22b76f0209
remove_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

Le groupe de journaux de la fonction n'est pas supprimé automatiquement. Vous pouvez le supprimer dans la [console CloudWatch Logs](#). Les suivis dans X-Ray expirent après quelques semaines et sont supprimés automatiquement.

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.

Pour plus d'informations pour utiliser Lambda avec un Amazon VPC, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC \(p. 81\)](#).

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

```
this is output
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution](#) (p. 33) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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 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 sg-0897d5f549934c2fb
```

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.
    uuid_inserted = str(uuid.uuid4())
    #Put the UUID to the cache.
    memcache_client.set('uuid', uuid_inserted)
    #Get item (UUID) from the cache.
    uuid_obtained = memcache_client.get('uuid')
    if uuid_obtained == str(uuid_inserted):
        # this print should go to the CloudWatch Logs and Lambda console.
        print ("Success: Fetched value %s from memcache" %(uuid_inserted))
    else:
        raise Exception("Value is not the same as we put :(. Expected %s got %s"
        %(uuid_inserted, uuid_obtained))

    return "Fetched value from memcache: " + str(uuid_obtained)
```

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.

Installez les dépendances avec Pip et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

Créer 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 file://function.zip --handler app.handler --runtime python3.8 \
--role arn:aws:iam::123456789012:role/lambda-vpc-role \
--vpc-config SubnetIds=subnet-0532bbe758ce7c71f,subnet-d6b7fda068036e11f,SecurityGroupIds=sg-0897d5f549934c2fb
```

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

Tester 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 [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

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-2:123456789012:targetgroup/lambdac-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
    }
  },
  "httpMethod": "GET",
  "path": "/lambda",
  "queryStringParameters": {
    "query": "1234ABCD"
  },
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "accept-encoding": "gzip",
    "accept-language": "en-US,en;q=0.9",
    "connection": "keep-alive",
    "host": "lambda-alb-123456789012.us-east-2.elb.amazonaws.com",
    "upgrade-insecure-requests": "1",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
  }
}
```

```
        "x-forwarded-for": "72.12.164.125",
        "x-forwarded-port": "80",
        "x-forwarded-proto": "http",
        "x-imforwards": "20"
    },
    "body": "",
    "isBase64Encoded": false
}
```

Votre fonction traite l'événement et renvoie un document de réponse à l'équilibrEUR de charge en JSON. Elastic Load Balancing convertit le document en réponse de réussite ou d'erreur HTTP et la renvoie à l'utilisateur.

Example format du document de réponse

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "isBase64Encoded": False,
    "headers": {
        "Content-Type": "text/html"
    },
    "body": "<h1>Hello from Lambda!</h1>"
}
```

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 AWS IoT

AWS IoT fournit une communication sécurisée entre les périphériques connectés à Internet (tels que les capteurs) et le cloud AWS. Vous pouvez ainsi collecter les données de télémétrie de plusieurs périphériques, les stocker et les analyser.

Vous pouvez créer des règles AWS IoT permettant à vos appareils d'interagir avec les services AWS. L'[moteur de règles AWS IoT](#) fournit un langage SQL pour sélectionner les données à partir des charges utiles de messages et envoyer des données à d'autres services, comme Amazon S3, Amazon DynamoDB et AWS Lambda. Vous définissez une règle pour appeler une fonction Lambda lorsque vous souhaitez appeler un autre service AWS ou un service tiers.

Lorsqu'un message IoT entrant déclenche la règle, AWS IoT appelle la fonction Lambda de manière [asynchrone \(p. 93\)](#) et lui transmet les données du message IoT.

L'exemple suivant montre une mesure de l'humidité à partir d'un capteur de serre. Les valeurs row et pos identifient l'emplacement du capteur. Cet exemple d'événement est basé sur le type de serre indiqué dans les [didacticiels sur les règles AWS IoT](#).

Example événement de message AWS IoT

```
{  
    "row" : "10",  
    "pos" : "23",  
    "moisture" : "75"  
}
```

Pour l'appel asynchrone, Lambda met le message en file d'attente et fait une [nouvelle tentative \(p. 111\)](#) si votre fonction renvoie une erreur. Configurez votre fonction avec une [destination \(p. 96\)](#) pour conserver les événements que la fonction n'a pas pu traiter.

Vous devez accorder l'autorisation au service AWS IoT d'appeler votre fonction Lambda. 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 my-function \  
--statement-id iot-events --action "lambda:InvokeFunction" --principal  
iotevents.amazonaws.com  
{  
    "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":  
    \"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":  
    \"arn:aws:lambda:us-west-2:123456789012:function:my-function\""  
}
```

Pour plus d'informations sur l'utilisation d'Lambda avec AWS IoT, consultez [Création d'une règle AWS Lambda](#).

Utilisation de AWS Lambda avec AWS IoT Events

AWS IoT Events surveille les entrées de plusieurs capteurs et applications IoT pour reconnaître les modèles d'événements. Ensuite, il effectue les actions appropriées lorsque des événements se produisent. AWS IoT Events reçoit ses entrées en tant que charges utiles JSON depuis de nombreuses sources. Il prend en charge des événements simples (chaque entrée déclenche un événement) et des événements complexes (plusieurs entrées doivent se produire pour déclencher l'événement).

Pour utiliser AWS IoT Events, vous définissez un modèle de détecteur, qui est un modèle d'état-machine de votre équipement ou processus. Outre les états, vous définissez des entrées et des événements pour le modèle. Vous définissez également les actions à effectuer lorsqu'un événement se produit. Utilisez une fonction Lambda pour une action lorsque vous souhaitez appeler un autre service AWS (par exemple Amazon Connect) ou effectuer des actions dans une application externe (par exemple, votre application de planification des ressources d'entreprise (ERP)).

Lorsque l'événement se produit, AWS IoT Events appelle votre fonction Lambda de manière asynchrone. Il fournit des informations sur le modèle du détecteur et l'événement qui a déclenché l'action. L'exemple

d'événement de message suivant est basé sur les définitions de l'[exemple simple étape par étape AWS IoT Events](#).

Example événement de message AWS IoT Events

```
{  
  "event": ":{  
    "eventName": "myChargedEvent",  
    "eventTime": 1567797571647,  
    "payload":{  
      "detector":{  
        "detectorModelName": "AWS_IoTEvents_Hello_World1567793458261",  
        "detectorModelVersion": "4",  
        "keyValue": "100009"  
      },  
      "eventTriggerDetails":{  
        "triggerType": "Message",  
        "inputName": "AWS_IoTEvents_HelloWorld_VoltageInput",  
        "messageId": "64c75a34-068b-4a1d-ae58-c16215dc4efd"  
      },  
      "actionExecutionId": "49f0f32f-1209-38a7-8a76-d6ca49dd0bc4",  
      "state":{  
        "variables": {},  
        "stateName": "Charged",  
        "timers": {}  
      }  
    }  
  }  
}
```

L'événement qui est transmis à la fonction Lambda comprend les champs suivants :

- **eventName** – Le nom de cet événement dans le modèle du détecteur.
- **eventTime** – Heure à laquelle l'événement s'est produit.
- **detector** – Nom et version du modèle du détecteur.
- **eventTriggerDetails** – Description de l'entrée qui a déclenché l'événement.
- **actionExecutionId** – Identifiant d'exécution unique de l'action.
- **state** – État du modèle de détecteur lorsque l'événement s'est produit.
- **stateName** – Nom de l'état dans le modèle du détecteur.
- **timers** – Tous les temporisateurs définis dans cet état.
- **variables** – Toutes les valeurs de variable définies dans cet état.

Vous devez accorder l'autorisation au service AWS IoT Events d'appeler votre fonction Lambda. 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 my-function \  
--statement-id iot-events --action "lambda:InvokeFunction" --principal  
ioevents.amazonaws.com  
{  
  "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":{\"Service  
\":\"ioevents.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":  
\"arn:aws:lambda:us-west-2:123456789012:function:my-function\"}"  
}
```

Pour plus d'informations sur l'utilisation de Lambda avec AWS IoT Events, consultez [Utilisation d'AWS IoT Events avec d'autres services](#).

Utilisation de AWS Lambda avec Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose capture, transforme et charge les données de streaming dans des services en aval comme Kinesis Data Analytics ou Amazon S3. Vous pouvez écrire des fonctions Lambda pour demander un traitement personnalisé supplémentaire des données avant que ces dernières ne soient envoyées en aval.

Example Événement de message Amazon Kinesis Data Firehose

```
{  
    "invocationId": "invoked123",  
    "deliveryStreamArn": "aws:lambda:events",  
    "region": "us-west-2",  
    "records": [  
        {  
            "data": "SGVsbG8gV29ybGQ=",  
            "recordId": "record1",  
            "approximateArrivalTimestamp": 151077216000,  
            "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": ""  
            }  
        }  
    ]  
}
```

Pour plus d'informations, consultez [Transformation des données Amazon Kinesis Data Firehose](#) dans le Manuel du développeur Kinesis Data Firehose.

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 \(p. 92\)](#) 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": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"4959033827149025608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
      "awsRegion": "us-east-2",
      "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
    },
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"4959033827149025608559692540925702759324208523137515618",
        "data": "VGhpcyBpcyBvbmx5IGEgdGVzdC4=",
        "approximateArrivalTimestamp": 1545084711.166
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
      "awsRegion": "us-east-2",
      "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
    }
  ]
}
```

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.

Note

Les paramètres de fenêtre de traitement par lots, de gestion des erreurs et de simultanéité ne sont pas disponibles pour les consommateurs de flux HTTP/2.

Par défaut, Lambda appelle votre fonction dès que des enregistrements sont disponibles dans le flux. Si le lot qu'il lit depuis le flux ne comporte qu'un enregistrement, Lambda n'envoie qu'un enregistrement à la fonction. Pour éviter d'appeler la fonction avec un petit nombre d'enregistrements, vous pouvez demander à la source d'événement de les mettre en mémoire tampon pendant 5 minutes maximum en configurant

une fenêtre de lot. Avant d'appeler la fonction, Lambda continue de lire les enregistrements du flux jusqu'à avoir collecté un lot complet ou jusqu'à l'expiration de la fenêtre de lot.

Si votre fonction renvoie une erreur, Lambda retente de traiter le lot jusqu'à ce que le traitement réussisse ou que les données expirent. Pour éviter les partitions bloquées, vous pouvez configurer le mappage de source d'événement pour réessayer avec une taille de lot plus petite, limiter le nombre de nouvelles tentatives ou supprimer les enregistrements qui sont trop vieux. Afin de conserver les événements supprimés, vous pouvez configurer le mappage de source d'événement pour envoyer les informations détaillées sur les lots ayant échoué à une file d'attente SQS ou à une rubrique SNS.

Vous pouvez également augmenter la simultanéité en traitant plusieurs lots à partir de chaque partition en parallèle. Lambda peut traiter jusqu'à 10 lots dans chaque partition simultanément. Si vous augmentez le nombre de lots simultanés par partition, Lambda garantit toujours le traitement dans l'ordre au niveau de la clé de partition.

Sections

- [Configuration de votre fonction et de votre flux de données \(p. 241\)](#)
- [Autorisations du rôle d'exécution \(p. 242\)](#)
- [Configuration d'un flux comme source d'événement \(p. 242\)](#)
- [API de mappage de la source d'événement \(p. 243\)](#)
- [Gestion des erreurs \(p. 245\)](#)
- [Métriques Amazon CloudWatch \(p. 246\)](#)
- [Didacticiel : Utilisation d'AWS Lambda avec Amazon Kinesis \(p. 246\)](#)
- [Exemple de code de fonction \(p. 250\)](#)
- [Modèle AWS SAM pour une application Kinesis \(p. 253\)](#)

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 d'autres enregistrements sont disponibles, Lambda continue de traiter les lots jusqu'à ce que la fonction rattrape le flux. Le mappage de source d'événement partage le débit de lecture avec d'autres utilisateurs 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/lambda-stream
{
    "Consumer": {
        "ConsumerName": "con1",
```

```
    "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/  
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. Il arrête de traiter les 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.

Si votre fonction ne peut pas augmenter sa capacité pour traiter le nombre total de lots simultanés, [demandez une augmentation de limite \(p. 30\)](#) ou [réservez de la simultanéité \(p. 61\)](#) pour votre fonction.

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 \(p. 33\)](#) de votre 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 [Rôle d'exécution AWS Lambda \(p. 33\)](#).

Pour envoyer les enregistrements des lots ayant échoué à une file d'attente ou à une rubrique, votre fonction a besoin d'autorisations supplémentaires. Chaque service de destination nécessite une autorisation différente, comme suit :

- Amazon SQS – [sq:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Configuration d'un flux comme 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 pour 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. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).

4. Choisissez un type de déclencheur.
5. Configurez les options requises, puis choisissez Ajouter.

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 envoyer à la fonction 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 \(p. 30\)](#) pour un appel synchrone (6 MB).
- Batch window (Fenêtre de lot) – Spécifiez l'intervalle de temps maximal pour collecter des enregistrements avant d'appeler la fonction, en secondes.
- 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.

- On-failure destination (Destination en cas d'échec) – File d'attente SQS ou rubrique SNS pour les enregistrements qui ne peuvent pas être traités. Lorsque Lambda supprime un lot d'enregistrements parce qu'il est trop vieux ou parce que toutes les nouvelles tentatives ont été épuisées, il envoie les détails sur ce lot à la file d'attente ou à la rubrique.
- Retry attempts (Nouvelles tentatives) – Nombre maximal de nouvelles tentatives effectuées par Lambda lorsqu'une fonction renvoie une erreur. Cela ne s'applique pas aux limitations ou erreurs de service où le lot n'a pas atteint la fonction.
- Maximum age of record (Âge maximal de l'enregistrement) – Âge maximal d'un enregistrement que Lambda envoie à votre fonction.
- Split batch on error (Fractionner le lot en cas d'erreur) – Lorsque la fonction renvoie une erreur, fractionnez le lot en deux avant de réessayer.
- Concurrent batches per shard (Lots simultanés par partition) – Traitez plusieurs lots simultanément à partir de la même partition.
- 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.

Note

Kinesis facture chaque partition et, pour les diffusions améliorées, les données lues à partir du flux. Pour plus d'informations sur la tarification, consultez [Tarification Amazon Kinesis](#).

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 gérer les mappages de source d'événement avec l'AWS CLI ou les kits SDK AWS, utilisez les actions d'API suivantes :

- [CreateEventSourceMapping \(p. 498\)](#)
- [ListEventSourceMappings \(p. 575\)](#)
- [GetEventSourceMapping \(p. 534\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)
- [DeleteEventSourceMapping \(p. 515\)](#)

Pour créer le mappage de source d'événement avec l'AWS CLI, utilisez la commande `create-event-source-mapping`. 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 \
--batch-size 500 --starting-position AT_TIMESTAMP --starting-position-timestamp 1541139109
\
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541139209.351,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "User action",
    "DestinationConfig": {},
    "MaximumRecordAgeInSeconds": 604800,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 10000
}
```

Pour utiliser un consommateur, spécifiez le consommateur l'ARN au lieu de l'ARN du flux.

Configurez des options supplémentaires pour personnaliser la manière dont les lots sont traités et pour indiquer quand supprimer les enregistrements qui ne peuvent pas être traités. L'exemple suivant met à jour un mappage de source d'événement pour envoyer un enregistrement d'échec à une file d'attente SQS après deux nouvelles tentatives ou si les enregistrements datent de plus d'une heure.

```
$ aws lambda update-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 --maximum-record-age-in-seconds 3600
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-2:123456789012:dlq"}}'
{
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
    "BatchSize": 100,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1573243620.0,
    "LastProcessingResult": "PROBLEM: Function call failed",
    "State": "Updating",
    "StateTransitionReason": "User action",
    "DestinationConfig": {},
    "MaximumRecordAgeInSeconds": 604800,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 10000
}
```

Les paramètres mis à jour sont appliqués de façon asynchrone et ne sont pas reflétés dans la sortie tant que le processus n'est pas terminé. Utilisez la commande `get-event-source-mapping` pour afficher le statut actuel.

```
$ aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
{
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
    "BatchSize": 100,
    "MaximumBatchingWindowInSeconds": 0,
    "ParallelizationFactor": 1,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1573244760.0,
    "LastProcessingResult": "PROBLEM: Function call failed",
    "State": "Enabled",
    "StateTransitionReason": "User action",
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "arn:aws:sqs:us-east-2:123456789012:dlq"
        }
    },
    "MaximumRecordAgeInSeconds": 3600,
    "BisectBatchOnFunctionError": false,
    "MaximumRetryAttempts": 2
}
```

Pour traiter plusieurs lots simultanément, utilisez l'option `--parallelization-factor`.

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

Gestion des erreurs

Le mappage de source d'événement qui lit les enregistrements à partir de votre flux Kinesis appelle votre fonction de façon synchrone et réessaie en cas d'erreur. Si la fonction est limitée ou si le service Lambda renvoie une erreur sans appeler la fonction, Lambda réessaie jusqu'à ce que les enregistrements expirent ou dépassent l'âge maximal que vous configurez dans le mappage de source d'événement.

Si la fonction reçoit les enregistrements mais renvoie une erreur, Lambda réessaie jusqu'à ce que les enregistrements du lot expirent, dépassent l'âge maximal ou atteignent la limite de nouvelles tentatives configurée. Pour les erreurs de fonction, vous pouvez également configurer le mappage de source d'événement pour fractionner un lot ayant échoué en deux lots. Le fait de réessayer avec des lots plus petits permet d'isoler les enregistrements incorrects et de contourner les problèmes de dépassement de délai. Le fractionnement d'un lot n'est pas comptabilisé dans la limite de nouvelles tentatives.

Si les mesures de gestion des erreurs échouent, Lambda supprime les enregistrements et continue de traiter les lots à partir du flux. Avec les paramètres par défaut, cela signifie qu'un enregistrement incorrect peut bloquer le traitement sur la partition concernée pendant jusqu'à one week. Pour éviter cela, configurez le mappage de source d'événement de votre fonction avec un nombre raisonnable de nouvelles tentatives et un âge maximal des enregistrements adapté à votre cas d'utilisation.

Pour conserver un enregistrement des lots supprimés, configurez une destination en cas d'échec. Lambda envoie un document à la rubrique ou à la file d'attente de cette destination avec des détails sur le lot.

Pour configurer une destination pour enregistrer les événements ayant échoué.

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.

3. Sous Designer (Concepteur), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Stream invocation (Appel de flux).
5. Pour Stream (Flux), choisissez un flux qui est mappé à la fonction.
6. Pour Type de destination, choisissez le type de ressource qui reçoit l'enregistrement d'appel.
7. Pour Destination, choisissez une ressource.
8. Choisissez Enregistrer.

L'exemple suivant illustre un enregistrement d'appel pour un flux Kinesis.

Example Enregistrement d'appel

```
{  
    "requestContext": {  
        "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",  
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
        "condition": "RetryAttemptsExhausted",  
        "approximateInvokeCount": 1  
    },  
    "responseContext": {  
        "statusCode": 200,  
        "executedVersion": "$LATEST",  
        "functionError": "Unhandled"  
    },  
    "version": "1.0",  
    "timestamp": "2019-11-14T00:38:06.021Z",  
    "KinesisBatchInfo": {  
        "shardId": "shardId-000000000001",  
        "startSequenceNumber": "49601189658422359378836298521827638475320189012309704722",  
        "endSequenceNumber": "49601189658422359378836298522902373528957594348623495186",  
        "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",  
        "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",  
        "batchSize": 500,  
        "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"  
    }  
}
```

Vous pouvez utiliser ces informations pour récupérer les enregistrements concernés à partir du flux afin de résoudre les problèmes. Les enregistrements réels ne sont pas inclus. Vous devez donc traiter cet enregistrement et les récupérer à partir du flux avant qu'ils expirent et soient perdus.

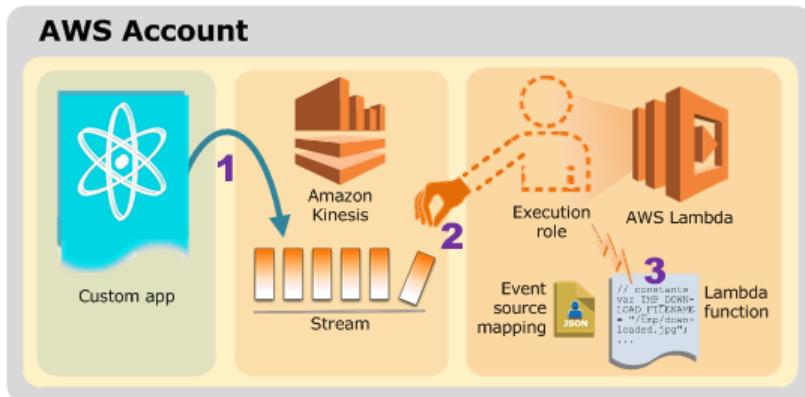
Métriques Amazon CloudWatch

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 de plus amples informations, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

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 Lambda fonction.

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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 \(p. 250\)](#).

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 = Buffer.from(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 ProcessKinesisRecords \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-kinesis-role
```

Tester 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": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgYSB0ZXN0Lg==",
```

```
        "approximateArrivalTimestamp": 1545084650.987
    },
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
}
]
```

2. Utilisez la commande `invoke` pour envoyer l'événement à la fonction.

```
$ aws lambda invoke --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 lambda-stream
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "340282366920746074317682119384634633455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
"49591073947768692513481539594623130411957558361251844610"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream",
        "StreamName": "lambda-stream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1544828156.0
    }
}
```

Vous utilisez l'ARN du flux à l'étape suivante pour associer le flux à la fonction Lambda.

Ajouter 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-west-2:123456789012:stream/lambda-stream \
--batch-size 100 --starting-position LATEST
```

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-west-2:123456789012:stream/lambda-stream
```

Dans la réponse, vous pouvez vérifier que la valeur d'état indique `enabled`. Les mappages de source d'événement peuvent être désactivés pour suspendre temporairement l'interrogation, ce qui entraîne la perte d'enregistrements.

Tester la configuration

Pour tester le mappage de source d'événement, ajoutez des enregistrements d'événements à votre Kinesis. La valeur `--data` est une chaîne que l'interface de ligne de commande encode en base64 avant de l'envoyer à Kinesis. Vous pouvez exécuter la même commande plus d'une fois pour ajouter plusieurs enregistrements dans le flux.

```
$ aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

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

Pour traiter des événements à partir de Amazon Kinesis, parcourez les enregistrements inclus dans l'objet événement et décodez les données codées en Base64 incluses dans chacun d'eux.

Note

Le code de cette page ne prend pas en charge les [enregistrements agrégés](#). Vous pouvez désactiver l'agrégation dans la configuration de la bibliothèque Producteur Kinesis <https://docs.aws.amazon.com/kinesis/latest/dev/kinesis-kpl-config.html>, ou utiliser la [bibliothèque d'agrégation d'enregistrements Kinesis](#) pour désagréger les enregistrements.

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js 8 \(p. 251\)](#)
- [Java 11 \(p. 251\)](#)
- [C# \(p. 252\)](#)

- [Python 3 \(p. 253\)](#)
- [Go \(p. 253\)](#)

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 = Buffer.from(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
};
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

Java 11

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 handleRequest(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.

Dépendances

- aws-lambda-java-core
- aws-lambda-java-events
- aws-java-sdk

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

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();
            }
        }
    }
}
```

Remplacez le fichier `Program.cs` d'un projet .NET Core par l'exemple ci-dessus. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans C# \(p. 414\)](#).

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))
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

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)
    }
}
```

Générez l'exécutable avec `go build` et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Go \(p. 398\)](#).

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 \(p. 246\)](#). La fonction et le gestionnaire du modèle sont compatibles avec le code Node.js. Si vous utilisez un autre code, mettez à jour les valeurs en conséquence.

Example template.yaml : flux Kinesis

```
AWSTemplateFormatVersion: '2010-09-09'
```

```

Transform: AWS::Serverless-2016-10-31
Resources:
  LambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      Timeout: 10
      Tracing: Active
    Events:
      Stream:
        Type: Kinesis
        Properties:
          Stream: !GetAtt stream.Arn
          BatchSize: 100
          StartingPosition: LATEST
  stream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
Outputs:
  FunctionName:
    Description: "Function name"
    Value: !Ref LambdaFunction
  StreamARN:
    Description: "Stream ARN"
    Value: !GetAtt stream.Arn

```

Le modèle crée une fonction Lambda, un flux Kinesis et un mappage de source d'événement. Le mappage de source d'événement lit à partir du flux et appelle la fonction.

Pour utiliser un [flux consommateur HTTP/2 \(p. 241\)](#), créez le consommateur dans le modèle et configurez le mappage de source d'événement pour lire à partir du consommateur plutôt qu'à partir du flux.

Example template.yaml : consommateur de flux Kinesis

```

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A function that processes data from a Kinesis stream.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      Timeout: 10
      Tracing: Active
    Events:
      Stream:
        Type: Kinesis
        Properties:
          Stream: !GetAtt streamConsumer.ConsumerARN
          StartingPosition: LATEST
          BatchSize: 100
  stream:
    Type: "AWS::Kinesis::Stream"
    Properties:
      ShardCount: 1
  streamConsumer:
    Type: "AWS::Kinesis::StreamConsumer"
    Properties:
      StreamARN: !GetAtt stream.Arn
      ConsumerName: "TestConsumer"
Outputs:

```

```
FunctionName:  
  Description: "Function name"  
  Value: !Ref function  
StreamARN:  
  Description: "Stream ARN"  
  Value: !GetAtt stream.Arn  
ConsumerARN:  
  Description: "Stream consumer ARN"  
  Value: !GetAtt streamConsumer.ConsumerARN
```

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 Lex

Vous pouvez utiliser Amazon Lex pour intégrer un bot dans votre application. Le bot Amazon Lex fournit une interface conversationnelle à vos utilisateurs. Amazon Lex offre une intégration préintégrée avec Lambda, qui vous permet d'utiliser une fonction Lambda avec votre bot Amazon Lex.

Lorsque vous configurez un bot Amazon Lex, vous pouvez spécifier une fonction Lambda pour effectuer la validation, l'exécution ou les deux. Pour la validation, Amazon Lex appelle la fonction Lambda après chaque réponse de l'utilisateur. La fonction Lambda peut valider la réponse et fournir une rétroaction corrective à l'utilisateur, si nécessaire. Pour l'exécution, Amazon Lex invoque la fonction Lambda pour exécuter la demande de l'utilisateur une fois que le bot a recueilli avec succès toutes les informations requises et a reçu la confirmation de l'utilisateur.

Vous pouvez [gérer la simultanéité \(p. 61\)](#) de votre fonction Lambda pour contrôler le nombre maximal de conversations simultanées que vous prenez en charge. L'API Amazon Lex renvoie un code d'état HTTP 429 (Too Many Requests) si la fonction est au niveau maximal de conversations simultanées.

L'API renvoie un code d'état HTTP 424 (Dependency Failed Exception) si la fonction Lambda émet une exception.

Le bot Amazon Lex appelle la fonction Lambda de manière [synchrone \(p. 92\)](#). Le paramètre d'événement contient des informations sur le bot et la valeur de chaque emplacement dans la boîte de dialogue. Le paramètre `invocationSource` indique si la fonction Lambda doit valider les entrées (`DialogCodeHook`) ou exécuter l'intention (`FulfillmentCodeHook`).

Example Événement de message Amazon Lex

```
{  
  "messageVersion": "1.0",  
  "invocationSource": "FulfillmentCodeHook",  
  "userId": "ABCD1234",  
  "sessionAttributes": {  
    "key1": "value1",  
    "key2": "value2",  
  },  
  "bot": {  
    "name": "OrderFlowers",  
    "alias": "prod",  
    "version": "1"  
  },  
  "outputDialogMode": "Text",  
  "currentIntent": {  
    "name": "OrderFlowers",  
    "slots": {  
      "FlowerType": "lilies",  
      "PickupDate": "2030-11-08",  
    }  
  }  
}
```

```
        "PickupTime": "10:00"
    },
    "confirmationStatus": "Confirmed"
}
}
```

Amazon Lex attend une réponse d'une fonction Lambda au format suivant. Le champ `dialogAction` est obligatoire. Les champs `sessionAttributes` et `recentIntentSummaryView` sont facultatifs.

Example Événement de message Amazon Lex

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
  "recentIntentSummaryView": [
    {
      "intentName": "Name",
      "checkpointLabel": "Label",
      "slots": {
        "slot name": "value",
        "slot name": "value"
      },
      "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if configured)",
      "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
      "fulfillmentState": "Fulfilled or Failed",
      "slotToElicit": "Next slot to elicit
    }
  ],
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Thanks, your pizza has been ordered."
    },
    "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
        {
          "title": "card-title",
          "subTitle": "card-sub-title",
          "imageUrl": "URL of the image to be shown",
          "attachmentLinkUrl": "URL of the attachment to be associated with the card",
          "buttons": [
            {
              "text": "button-text",
              "value": "Value sent to server on button click"
            }
          ]
        }
      ]
    }
  }
}
```

Notez que les champs supplémentaires requis pour `dialogAction` varient en fonction de la valeur du champ `type`. Pour plus d'informations sur les champs d'événement et de réponse, consultez [Format des réponses et événements Lambda](#) dans le Guide du développeur Amazon Lex. Pour obtenir un exemple de

didacticiel montrant comment utiliser Lambda avec Amazon Lex, consultez [Exercice 1 : Création d'un bot Amazon Lex à l'aide d'un modèle](#) dans le Guide du développeur Amazon Lex.

Rôles et autorisations

Vous devez configurer un rôle lié à un service en tant que [rôle d'exécution \(p. 33\)](#) de votre fonction. Amazon Lex définit le rôle lié au service avec des autorisations prédéfinies. Lorsque vous créez un bot Amazon Lex à l'aide de la console, le rôle lié au service est créé automatiquement. Pour créer un rôle lié à un service avec l'interface de ligne de commande AWS, utilisez la commande `create-service-linked-role`.

```
$ aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

Cette commande crée le rôle suivant :

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
            "Service": "lex.amazonaws.com"
          }
        }
      ]
    },
    "RoleName": "AWSServiceRoleForLexBots",
    "Path": "/aws-service-role/lex.amazonaws.com/",
    "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots"
  }
}
```

Si votre fonction Lambda utilise d'autres services AWS, vous devez ajouter les autorisations correspondantes au rôle lié au service.

Vous utilisez une stratégie d'autorisations basée sur les ressources pour autoriser l'intention Amazon Lex à appeler votre fonction Lambda. Si vous utilisez la console Amazon Lex, la stratégie d'autorisations est créée automatiquement. À partir de l'interface de ligne de commande AWS, utilisez la commande `Lambda add-permission` pour définir l'autorisation. L'exemple suivant définit l'autorisation pour l'intention `OrderFlowers`.

```
aws lambda add-permission \
--function-name OrderFlowersCodeHook \
--statement-id LexGettingStarted-OrderFlowersBot \
--action lambda:InvokeFunction \
--principal lex.amazonaws.com \
--source-arn "arn:aws:lex:us-east-1:123456789012 ID:intent:OrderFlowers:*
```

Utilisation de AWS Lambda avec Amazon RDS

Vous pouvez utiliser AWS Lambda pour traiter les notifications d'événements à partir d'une base de données Amazon Relational Database Service (Amazon RDS). Amazon RDS envoie les notifications à une

rubrique Amazon Simple Notification Service (Amazon SNS), que vous pouvez configurer pour appeler une fonction Lambda. Amazon SNS encapsule le message de Amazon RDS dans son propre document d'événement et l'envoie à votre fonction.

Example Message Amazon RDS d'un événement Amazon SNS

```
{  
    "Records": [  
        {  
            "EventVersion": "1.0",  
            "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:rds-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",  
            "EventSource": "aws:sns",  
            "Sns": {  
                "SignatureVersion": "1",  
                "Timestamp": "2019-01-02T12:45:07.000Z",  
                "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkaAi6RibDsvpi+tE/1+82j...65r==",  
                "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",  
                "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
                "Message": "{\"Event Source\":\"db-instance\", \"Event Time\":\"2019-01-02 12:45:06.000\", \"Identifier Link\":\"https://console.aws.amazon.com/rds/home?region=eu-west-1#dbinstance:id=dbinstanceid\", \"Source ID\":\"dbinstanceid\", \"Event ID\":\"http://docs.amazonaws.com/AmazonRDS/latest/UserGuide/USER_Events.html#RDS-EVENT-0002\", \"Event Message\":\"Finished DB Instance backup\"}",  
                "MessageAttributes": {},  
                "Type": "Notification",  
                "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",  
                "TopicArn": "arn:aws:sns:us-east-2:123456789012:sns-lambda",  
                "Subject": "RDS Notification Message"  
            }  
        }  
    ]  
}
```

Pour plus d'informations sur la configuration d'une base de données Amazon RDS pour envoyer des notifications, consultez [Utilisation de la notification d'événement Amazon RDS](#) dans le Guide de l'utilisateur Amazon Relational Database Service.

Pour plus d'informations sur l'utilisation de Amazon SNS en tant que déclencheur, consultez [Utilisation de AWS Lambda avec Amazon SNS \(p. 280\)](#).

Rubriques

- [Didacticiel : Configuration d'une fonction Lambda pour accéder à Amazon RDS dans un Amazon VPC \(p. 258\)](#)

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.

Pour plus d'informations pour utiliser Lambda avec un Amazon VPC, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC \(p. 81\)](#).

Prérequis

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

- Ouvrez la page [Rôles](#) dans la console IAM.
- Choisissez [Créer un rôle](#).
- 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 de la section [Création d'une instance DB MySQL et connexion à une base de données sur une instance DB 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 pymysql.MySQLError as e:
    logger.error("ERROR: Unexpected error: Could not connect to MySQL instance.")
    logger.error(e)
    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 Employee ( EmpID  int NOT NULL, Name varchar(255) NOT
NULL, PRIMARY KEY (EmpID))")
        cur.execute('insert into Employee (EmpID, Name) values(1, "Joe")')
        cur.execute('insert into Employee (EmpID, Name) values(2, "Bob")')
        cur.execute('insert into Employee (EmpID, Name) values(3, "Mary")')
        conn.commit()
```

```
cur.execute("select * from Employee")
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](#)).

Installez les dépendances avec Pip et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

Créer la fonction Lambda

Créez la fonction Lambda à l'aide de la commande `create-function`. Vous pouvez trouver les ID de sous-réseau et l'ID du groupe de sécurité pour votre VPC par défaut dans la [console Amazon VPC](#).

```
$ aws lambda create-function --function-name CreateTableAddRecordsAndRead --runtime
python3.8 \
--zip-file fileb://app.zip --handler app.handler \
--role arn:aws:iam::123456789012:role/lambda-vpc-role \
--vpc-config SubnetIds=subnet-0532bb6758ce7c71f,subnet-
d6b7fda068036e11f,SecurityGroupIds=sg-0897d5f549934c2fb
```

Tester 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 [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Utilisation de AWS Lambda avec Amazon S3

Vous pouvez utiliser Lambda pour traiter les [notifications](#) provenant d'Amazon Simple Storage Service. Amazon S3 peut envoyer un événement à une fonction Lambda lorsqu'un objet est créé ou supprimé. Vous configurez des paramètres de notification sur un compartiment et accordez à Amazon S3 l'autorisation d'appeler une fonction sur la stratégie d'autorisations basée sur les ressources de la fonction.

Warning

Si votre fonction Lambda utilise le même compartiment que celui qui la déclenche, la fonction risque de s'exécuter en boucle. Par exemple, si le compartiment déclenche une fonction chaque fois qu'un objet est chargé et que la fonction charge un objet dans le compartiment, la fonction se déclenche elle-même indirectement. Afin d'éviter cela, utilisez deux compartiments ou configurez le déclencheur pour qu'il s'applique uniquement à un préfixe utilisé pour les objets entrants.

Amazon S3 appelle votre fonction de manière [asynchrone](#) (p. 93) avec un événement qui contient des détails sur l'objet. L'exemple suivant montre un événement envoyé par Amazon S3 après le chargement d'un package de déploiement dans Amazon S3.

Example Événement de notification Amazon S3

```
{  
  "Records": [  
    {  
      "eventVersion": "2.1",  
      "eventSource": "aws:s3",  
      "awsRegion": "us-east-2",  
      "eventTime": "2019-09-03T19:37:27.192Z",  
      "eventName": "ObjectCreated:Put",  
      "userIdentity": {  
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"  
      },  
      "requestParameters": {  
        "sourceIPAddress": "205.255.255.255"  
      },  
      "responseElements": {  
        "x-amz-request-id": "D82B88E5F771F645",  
        "x-amz-id-2": "  
"v1r7PnpV2Ce81l0PRw6jlUpck7Jo5zsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="}  
      },  
      "s3": {  
        "s3SchemaVersion": "1.0",  
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",  
        "bucket": {  
          "name": "lambda-artifacts-deafc19498e3f2df",  
          "ownerIdentity": {  
            "principalId": "A3I5XTEXAMA13E"  
          },  
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"  
        },  
        "object": {  
          "key": "b21b84d653bb07b05b1e6b33684dc11b".  
        }  
      }  
    }  
  ]  
}
```

```
        "size": 1305107,  
        "eTag": "b21b84d653bb07b05b1e6b33684dc11b",  
        "sequencer": "0C0F6F405D6ED209E1"  
    }  
}  
]  
}
```

Pour appeler la fonction, Amazon S3 a besoin de l'autorisation de la [stratégie basée sur les ressources \(p. 36\)](#) la fonction. Lorsque vous configurez un déclencheur Amazon S3 dans la console Lambda, cette dernière modifie la stratégie basée sur les ressources pour permettre à Amazon S3 d'appeler la fonction si le nom du compartiment et l'ID de compte correspondent. Si vous configurez la notification dans Amazon S3, vous utilisez l'API Lambda pour mettre à jour la stratégie. Vous pouvez également utiliser l'API Lambda pour accorder une autorisation à un autre compte ou limiter l'autorisation à un alias désigné.

Si votre fonction utilise le kit SDK AWS pour gérer les ressources Amazon S3, elle a également besoin d'autorisations Amazon S3 dans son [rôle d'exécution \(p. 33\)](#).

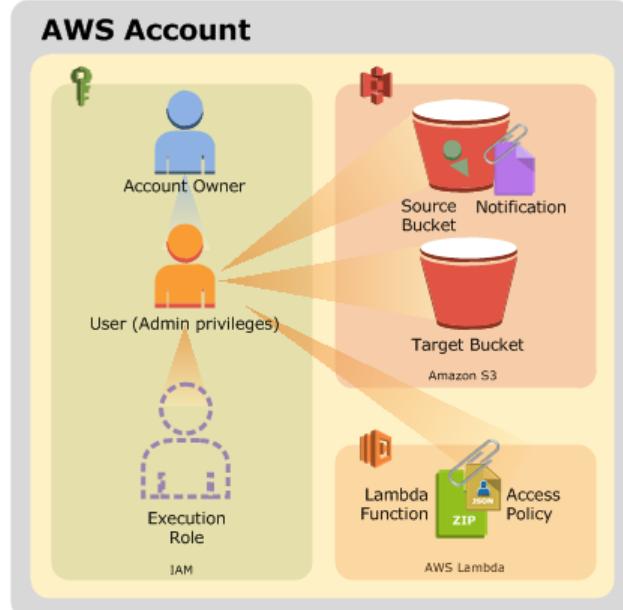
Rubriques

- [Didacticiel : Utilisation d'AWS Lambda avec Amazon S3 \(p. 263\)](#)
- [Exemple de code de fonction Amazon S3 \(p. 270\)](#)
- [Modèle AWS SAM pour une application Amazon S3 \(p. 275\)](#)

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Installez npm pour gérer les dépendances de la fonction.

Le didacticiel utilise des commandes AWS CLI pour créer et appeler la fonction Lambda. Installez l'[AWS CLI et configurez-la avec vos informations d'identification AWS](#)

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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.

Créer des compartiments et charger 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 doit correspondre à celui du compartiment **source**, suivi de la mention **-resized**. Par exemple : `mybucket` et `mybucket-resized`.
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 S3 \(p. 270\)](#).

Example index.js

```
// dependencies
const AWS = require('aws-sdk');
const util = require('util');
const sharp = require('sharp');

// get reference to S3 client
const s3 = new AWS.S3();

exports.handler = async (event, context, callback) => {

    // Read options from the event parameter.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    const srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    const dstBucket = srcBucket + "-resized";
    const dstKey    = "resized-" + srcKey;

    // Infer the image type from the file suffix.
    const typeMatch = srcKey.match(/^(.*\.(?:\w+|image\/*\w+))$/);
    if (!typeMatch) {
        console.log("Could not determine the image type.");
        return;
    }

    // Check that the image type is supported
    const imageType = typeMatch[1].toLowerCase();
    if (imageType != "jpg" && imageType != "png") {
```

```
        console.log(`Unsupported image type: ${imageType}`);
        return;
    }

    // Download the image from the S3 source bucket.

    try {
        const params = {
            Bucket: srcBucket,
            Key: srcKey
        };
        var origimage = await s3.getObject(params).promise();

    } catch (error) {
        console.log(error);
        return;
    }

    // set thumbnail width. Resize will set the height automatically to maintain aspect
ratio.
    const width = 200;

    // Use the Sharp module to resize the image and save in a buffer.
    try {
        var buffer = await sharp(origimage.Body).resize(width).toBuffer();

    } catch (error) {
        console.log(error);
        return;
    }

    // Upload the thumbnail image to the destination bucket
    try {
        const destparams = {
            Bucket: dstBucket,
            Key: dstKey,
            Body: buffer,
            ContentType: "image"
        };

        const putResult = await s3.putObject(destparams).promise();

    } catch (error) {
        console.log(error);
        return;
    }

    console.log('Successfully resized ' + srcBucket + '/' + srcKey +
        ' and uploaded to ' + dstBucket + '/' + dstKey);
};
```

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 `examplebucket-resized`.
- 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`.

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. Enregistrez le code de la fonction comme `index.js` dans un dossier nommé `lambda-s3`.
2. Installez la bibliothèque Sharp avec npm. Pour Linux, utilisez la commande suivante.

```
lambda-s3$ npm install sharp
```

Pour macOS, utilisez la commande suivante.

```
lambda-s3$ npm install --arch=x64 --platform=linux --target=12.13.0 sharp
```

Une fois que vous avez terminé cette étape, vous disposez de la structure de dossier suivante :

```
lambda-s3
|- index.js
|- /node_modules/sharp
# /node_modules/...
```

3. Créez un package de déploiement avec le code de la fonction et les dépendances.

```
lambda-s3$ zip -r function.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 nodejs12.x \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

Pour le paramètre de rôle, remplacez la séquence de numéros par votre l'ID de compte AWS. 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
```

Tester 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": "AIDAJDPLRKG7UEXAMPLE"  
    },  
    "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": "096fKKXTRTl3on89fVO.nfljtsv6qko"  
        }  
    }  
}
```

2. 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
```

3. Vérifiez que la miniature a été créée dans le compartiment cible.

Configurer 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.
 - Le compartiment appartient à votre compte. Si vous supprimez un compartiment, un autre compte peut créer un compartiment avec le même ARN.

```
$ aws lambda add-permission --function-name CreateThumbnail --principal s3.amazonaws.com \
--statement-id s3invoke --action "lambda:InvokeFunction" \
--source-arn arn:aws:s3:::sourcebucket \
--source-account 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 CreateThumbnail
```

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.

Important

Cette procédure configure le compartiment pour qu'il appelle votre fonction chaque fois qu'un objet y est créé. Assurez-vous de configurer cette option uniquement sur le compartiment source et de ne pas créer d'objets dans ce dernier à partir de la fonction déclenchée. Sinon, votre fonction pourrait se faire appeler continuellement dans une boucle (p. 262).

Pour configurer des notifications

1. Ouvrez la [console Amazon S3](#).
2. Choisissez le compartiment source.
3. Choisissez Properties.
4. Sous Événements, configuez 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.

Tester 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 S3

Un exemple de code est disponible pour les langages suivants.

Rubriques

- [Node.js 12.x \(p. 270\)](#)
- [Java 11 \(p. 272\)](#)
- [Python 3 \(p. 274\)](#)

Node.js 12.x

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
const AWS = require('aws-sdk');
const util = require('util');
const sharp = require('sharp');

// get reference to S3 client
const s3 = new AWS.S3();

exports.handler = async (event, context, callback) => {

    // Read options from the event parameter.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    const srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    const dstBucket = srcBucket + "-resized";
    const dstKey    = "resized-" + srcKey;

    // Infer the image type from the file suffix.
    const typeMatch = srcKey.match(/^(.*\.(?:jpe?g|png))$/);
    if (!typeMatch) {
        console.log("Could not determine the image type.");
        return;
    }

    // Check that the image type is supported
    const imageType = typeMatch[1].toLowerCase();
    if (imageType != "jpg" && imageType != "png") {
        console.log(`Unsupported image type: ${imageType}`);
        return;
    }

    // Download the image from the S3 source bucket.

    try {
        const params = {
            Bucket: srcBucket,
            Key: srcKey
        };
        var origimage = await s3.getObject(params).promise();
    }
}
```

```
    } catch (error) {
        console.log(error);
        return;
    }

    // set thumbnail width. Resize will set the height automatically to maintain aspect
    // ratio.
    const width = 200;

    // Use the Sharp module to resize the image and save in a buffer.
    try {
        var buffer = await sharp(origimage.Body).resize(width).toBuffer();

    } catch (error) {
        console.log(error);
        return;
    }

    // Upload the thumbnail image to the destination bucket
    try {
        const destparams = {
            Bucket: dstBucket,
            Key: dstKey,
            Body: buffer,
            ContentType: "image"
        };

        const putResult = await s3.putObject(destparams).promise();

    } catch (error) {
        console.log(error);
        return;
    }

    console.log('Successfully resized ' + srcBucket + '/' + srcKey +
        ' and uploaded to ' + dstBucket + '/' + dstKey);
};
```

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 les dépendances. Les exemples de code utilisent les bibliothèques suivantes :
 - Kit SDK AWS pour JavaScript dans Node.js
 - Sharp pour node.js

L'environnement d'exécution AWS Lambda intègre déjà le kit AWS SDK pour JavaScript en Node.js, c'est pourquoi 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. Pour Linux, utilisez la commande suivante.

```
$ npm install sharp
```

Pour macOS, utilisez la commande suivante.

```
$ npm install --arch=x64 --platform=linux --target=12.13.0 sharp
```

3. Enregistrez l'exemple de code dans un fichier nommé `index.js`.

4. 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 `examplebucket-resized`.
 - 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`.
5. 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/sharp
```

6. Comptez le fichier `index.js` et le dossier `node_modules` dans `CreateThumbnail.zip`.

Java 11

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.handler`). Pour plus d'informations sur l'utilisation des interfaces pour créer un gestionnaire, consultez la page [Interfaces du gestionnaire \(p. 374\)](#).

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 handler.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.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import com.amazonaws.AmazonServiceException;
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.event.S3EventNotification.S3EventNotificationRecord;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

```
public class Handler 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().getUrlDecodedKey();

            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 = AmazonS3ClientBuilder.defaultClient();
            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
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);
try {
    s3Client.putObject(dstBucket, dstKey, is, meta);
}
catch(AmazonServiceException e)
{
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
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.

Dépendances

- `aws-lambda-java-core`
- `aws-lambda-java-events`
- `aws-java-sdk`

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

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 `lambda_function.py`

```
import boto3
```

```
import os
import sys
import uuid
from urllib.parse import unquote_plus
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 lambda_handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), key)
```

Note

La bibliothèque d'images utilisée par ce code doit être installée dans un environnement Linux afin de créer un package de déploiement fonctionnel.

Pour créer un package de déploiement

1. Copiez l'exemple de code dans un fichier nommé `lambda_function.py`.
2. Créez un environnement virtuel.

```
s3-python$ virtualenv v-env
s3-python$ source v-env/bin/activate
```

3. Installez les bibliothèques dans l'environnement virtuel

```
(v-env) s3-python$ pip install Pillow boto3
```

4. Créez un package de déploiement avec le contenu des bibliothèques installées.

```
(v-env) s3-python$ cd $VIRTUAL_ENV/lib/python3.8/site-packages
(v-env) python-s3/v-env/lib/python3.8/site-packages$ zip -r9 ${OLDPWD}/function.zip .
```

5. Ajoutez le code du gestionnaire au package de déploiement et désactivez l'environnement virtuel.

```
(v-env) python-s3/v-env/lib/python3.8/site-packages$ cd ${OLDPWD}
(v-env) python-s3$ zip -g function.zip lambda_function.py
      adding: lambda_function.py (deflated 55%)
(v-env) python-s3$ deactivate
```

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 \(p. 263\)](#). 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 les opérations par lot Amazon S3

Vous pouvez utiliser les opérations par lot Amazon S3 pour appeler une fonction Lambda sur un grand ensemble d'objets Amazon S3. Amazon S3 effectue le suivi de la progression des opérations par lot, envoie des notifications et stocke un rapport d'achèvement indiquant le statut de chaque action.

Pour exécuter une opération par lot, vous créez un [travail d'opérations par lot](#) Amazon S3. Lorsque vous créez le travail, vous fournissez un manifeste (la liste des objets) et configurez l'action à effectuer sur ces objets.

Lorsque le travail par lot démarre, Amazon S3 appelle la fonction Lambda [de manière synchrone \(p. 92\)](#) pour chaque objet figurant dans le manifeste. Le paramètre d'événement inclut les noms du compartiment et de l'objet.

L'exemple suivant illustre l'événement qu'Amazon S3 envoie à la fonction Lambda pour un objet nommé `customerImage1.jpg` dans le compartiment `awsexamplebucket`.

Example événement de demande de lot Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "job": {
    "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
```

```
},
"tasks": [
{
    "taskId": "dGFza2lkZ29lc2hlcmtUK",
    "s3Key": "customerImage1.jpg",
    "s3VersionId": "1",
    "s3BucketArn": "arn:aws:s3:us-east-1:0123456788:awsexamplebucket"
}
]
```

Votre fonction Lambda doit renvoyer un objet JSON avec les champs, comme indiqué dans l'exemple suivant. Vous pouvez copier `invocationId` et `taskId` à partir du paramètre d'événement. Vous pouvez renvoyer une chaîne dans `resultString`. Amazon S3 enregistre les valeurs `resultString` dans le rapport d'achèvement.

Example Réponse à la demande de traitement par lots Amazon S3

```
{
    "invocationSchemaVersion": "1.0",
    "treatMissingKeysAs" : "PermanentFailure",
    "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "results": [
        {
            "taskId": "dGFza2lkZ29lc2hlcmtUK",
            "resultCode": "Succeeded",
            "resultString": "[\"Alice\", \"Bob\"]"
        }
    ]
}
```

Appel de fonctions Lambda à partir d'opérations par lot Amazon S3

Vous pouvez appeler la fonction Lambda avec un ARN de fonction qualifié ou non qualifié. Si vous souhaitez utiliser la même version de fonction pour l'ensemble du travail par lot, configurez une version de fonction spécifique dans le paramètre `FunctionARN` lorsque vous créez votre travail. Si vous configurez un alias ou le qualificateur `$LATEST`, le travail par lot commence immédiatement à appeler la nouvelle version de la fonction si l'alias ou `$LATEST` est mis à jour pendant l'exécution du travail.

Notez que vous ne pouvez pas réutiliser une fonction événementielle Amazon S3 existante pour les opérations par lot. En effet, l'opération par lot Amazon S3 transmet un paramètre d'événement différent à la fonction Lambda et attend un message en retour avec une structure JSON spécifique.

Dans la [stratégie basée sur les ressources \(p. 36\)](#) que vous créez pour le travail par lot Amazon S3, assurez-vous que vous définissez l'autorisation pour que le travail appelle votre fonction Lambda.

Dans le [rôle d'exécution \(p. 33\)](#) de la fonction, définissez une stratégie d'approbation pour qu'Amazon S3 endosse ce rôle lorsqu'il exécute votre fonction.

Si votre fonction utilise le kit AWS SDK pour gérer les ressources Amazon S3, vous devez ajouter des autorisations Amazon S3 au rôle d'exécution.

Lorsque le travail s'exécute, Amazon S3 démarre plusieurs instances de fonction pour traiter les objets Amazon S3 en parallèle, jusqu'à la [limite de simultanéité \(p. 106\)](#) de la fonction. Amazon S3 limite la montée en puissance initiale des instances afin d'éviter des coûts excessifs pour de petits travaux.

Si la fonction Lambda renvoie un code de réponse `TemporaryFailure`, Amazon S3 réessaie l'opération.

Pour plus d'informations sur les opérations par lot Amazon S3, consultez [Exécution d'opérations par lot](#) dans le Manuel du développeur Amazon S3.

Pour voir un exemple d'utilisation d'une fonction Lambda dans des opérations par lot Amazon S3, consultez [Appel d'une fonction Lambda à partir d'opérations par lot Amazon S3](#) dans le Manuel du développeur Amazon S3.

Utilisation de AWS Lambda avec Amazon SES

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.

Example Événement de message 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=jX3F0bCAI7sIbkHyy3mLY028ieDQz2R0P8HwQkk1Fj4=; b=sQwJ+LMe9RjkesGu+vqU56asvMhrLRRYrWCbV"
            },
            {
              "name": "MIME-Version",
            }
          ]
        }
      }
    }
  ]
}
```

```

        "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"
}
]
}
```

Pour plus d'informations, consultez [Action Lambda](#) dans le Manuel du développeur Amazon SES.

Utilisation de AWS Lambda avec Amazon SNS

Vous pouvez utiliser une fonction Lambda pour traiter les notifications Amazon Simple Notification Service. Amazon SNS prend en charge les fonctions Lambda en tant que cible pour les messages envoyés à une rubrique. Vous pouvez abonner votre fonction à des rubriques du même compte ou d'autres comptes AWS.

Amazon SNS appelle votre fonction [de façon asynchrone \(p. 93\)](#) avec un événement qui contient un message et des métadonnées.

Example Événement de message Amazon SNS

```
{  
    "Records": [  
        {  
            "EventVersion": "1.0",  
            "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:sns-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",  
            "EventSource": "aws:sns",  
            "Sns": {  
                "SignatureVersion": "1",  
                "Timestamp": "2019-01-02T12:45:07.000Z",  
                "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",  
                "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",  
                "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
                "Message": "Hello from SNS!",  
                "MessageAttributes": {  
                    "Test": {  
                        "Type": "String",  
                        "Value": "TestString"  
                    },  
                    "TestBinary": {  
                        "Type": "Binary",  
                        "Value": "TestBinary"  
                    }  
                },  
                "Type": "Notification",  
                "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",  
                "TopicArn": "arn:aws:sns:us-east-2:123456789012:sns-lambda",  
                "Subject": "TestInvoke"  
            }  
        }  
    ]  
}
```

Pour les appels asynchrones, Lambda place le message en file d'attente et gère les nouvelles tentatives. Si Amazon SNS n'est pas en mesure d'atteindre Lambda ou si le message est rejeté, Amazon SNS réalise de nouvelles tentatives à intervalles croissants sur plusieurs heures. Pour de plus amples informations, veuillez consulter [Fiabilité](#) dans la FAQ Amazon SNS.

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.

Pour plus d'informations, consultez [Invocation des fonctions Lambda en utilisant des notifications Amazon SNS](#) dans le Amazon Simple Notification Service Manuel du développeur.

Rubriques

- [Didacticiel : Utilisation d'AWS Lambda avec Amazon Simple Notification Service \(p. 281\)](#)
- [Exemple de code de fonction \(p. 283\)](#)

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et 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.

Création d'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 \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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.

Créer 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 \(p. 283\)](#).

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 file://function.zip --handler index.handler --runtime nodejs12.x \
--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.

Configurer 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.

Créer 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 \(p. 284\)](#)
- [Java 11 \(p. 284\)](#)
- [Go \(p. 285\)](#)
- [Python 3 \(p. 285\)](#)

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");
};
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

Java 11

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;
    }
}
```

Dépendances

- [aws-lambda-java-core](#)
- [aws-lambda-java-events](#)

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

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)
}
```

Générez l'exécutable avec `go build` et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Go \(p. 398\)](#).

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
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

Utilisation de AWS Lambda avec Amazon SQS

Vous pouvez utiliser une fonction AWS Lambda pour traiter les messages dans une file d'attente Amazon Simple Queue Service (Amazon SQS). Les [mappages de source d'événement \(p. 101\)](#) Lambda prennent en charge les [files d'attente standard](#) et les [files d'attente FIFO \(premier entré, premier sorti\)](#). 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 \(p. 92\)](#) 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. L'exemple suivant présente un événement pour un lot de deux messages.

Example Événement de message Amazon SQS (file d'attente standard)

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAENQZJOL023YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
      "messageAttributes": {},
      "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
      "awsRegion": "us-east-2"
    },
    {
      "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
      "receiptHandle": "AQEBzWwaftRI0KuVm4tP+/7q1rGgNqicHq...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082650636",
        "SenderId": "AIDAENQZJOL023YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082650649"
      },
      "messageAttributes": {},
      "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
      "awsRegion": "us-east-2"
    }
  ]
}
```

Pour les files d'attente FIFO, les enregistrements contiennent des attributs supplémentaires liés à la déduplication et au séquençage.

Example Événement de message Amazon SQS (file d'attente FIFO)

```
{
  "Records": [
    {
      "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
      "receiptHandle": "AQEBBX8nesZExmkhsmZeyIE8iQAMig7qw...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1573251510774",
        "SequenceNumber": "18849496460467696128",
        "MessageGroupId": "1",
        "SenderId": "AIDAI023YVJENQZJOL4VO",
        "MessageDuplicationId": "1",
        "ApproximateFirstReceiveTimestamp": "1573251510774"
      }
    }
  ]
}
```

```
        },
        "messageAttributes": {},
        "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
        "eventSource": "aws:sqs",
        "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo fifo",
        "awsRegion": "us-east-2"
    ]
}
```

Lorsque Lambda lit un lot, les messages restent dans la file d'attente mais sont masqués pendant toute la durée du [délai de visibilité](#) de la file d'attente. Si votre fonction traite un lot avec succès, Lambda supprime les messages depuis la file d'attente. Si votre fonction est [limitée \(p. 106\)](#), renvoie une erreur ou ne répond pas, le message devient visible à nouveau. Tous les messages d'un lot en échec retournent dans la file d'attente, ainsi le code de la fonction peut traiter le même message plusieurs fois sans effets secondaires.

Dimensionnement et traitement

Pour les files d'attente standard, Lambda utilise une [longue interrogation](#) pour interroger une file d'attente jusqu'à ce qu'elle devienne active. Lorsque les messages sont disponibles, Lambda lit jusqu'à 5 lots et les envoie à votre fonction. Si les messages sont toujours disponibles, Lambda augmente le nombre de processus de lecture de lots jusqu'à 60 instances supplémentaires par minute. Le nombre maximum de lots qui peut être traité simultanément par un mappage de source d'événement est 1 000.

Pour les files d'attente FIFO, Lambda envoie des messages à votre fonction dans l'ordre où elle les reçoit. Lorsque vous envoyez un message à une file d'attente FIFO, vous spécifiez un [ID de groupe de messages](#). Amazon SQS garantit que les messages du même groupe sont remis à Lambda dans l'ordre. Lambda trie les messages en groupes et n'envoie qu'un seul lot à la fois pour un groupe. Si la fonction renvoie une erreur, les nouvelles tentatives au niveau des messages concernés ont lieu avant qu'Lambda ne puisse recevoir des messages supplémentaires provenant du même groupe.

Votre fonction peut être dimensionnée en simultanéité au nombre de groupes de messages actifs. Pour plus d'informations, consultez [SQS FIFO en tant que source d'événements](#) sur le blog AWS Compute.

Configuration d'une file d'attente à utiliser avec Lambda

[Créez une file d'attente SQS](#) 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 \(p. 53\)](#) que vous configuez 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](#). Lorsque votre fonction renvoie une erreur, Lambda la laisse dans la file d'attente. Une fois le délai de visibilité expiré, Lambda reçoit à nouveau le message. Pour envoyer des messages à une deuxième file d'attente après plusieurs réceptions, configurez une file d'attente de lettres mortes sur votre file d'attente source.

Note

Assurez-vous que vous configurez la file d'attente de lettres mortes dans la file d'attente source, et dans sur la fonction Lambda. La file d'attente de lettres mortes que vous configurez dans une

fonction est utilisée pour la [file d'attente d'appels asynchrones \(p. 93\)](#) et non pour les files d'attente source d'événement.

Si votre fonction renvoie une erreur ou si elle ne peut pas être appelée parce qu'elle a atteint le niveau de simultanéité maximal, le traitement peut aboutir avec des tentatives supplémentaires. Pour donner aux messages une meilleure chance d'être traités avant de les envoyer dans la file d'attente de lettres mortes, définissez `maxReceiveCount` sur 5 au minimum dans la stratégie de réacheminement de la file d'attente source.

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 de votre fonction.

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

Pour plus d'informations, consultez [Rôle d'exécution AWS Lambda \(p. 33\)](#).

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. Quand Lambda appelle la fonction cible, l'événement peut contenir plusieurs éléments, jusqu'à une taille de lot maximale configurable.

Pour configurer votre fonction afin de lire depuis Amazon SQS dans la console Lambda, créez un déclencheur SQS.

Pour créer un déclencheur

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Designer (Concepteur), choisissez Add trigger (Ajouter un déclencheur).
4. Choisissez un type de déclencheur.
5. Configurez les options requises, puis choisissez Ajouter.

Lambda prend en charge les options suivantes pour les sources d'événement Amazon SQS.

Options de source d'événement

- File d'attente SQS – La file d'attente Amazon SQS à partir de laquelle lire les enregistrements.
- Taille du lot – Nombre d'éléments à lire à partir de la file d'attente dans chaque lot, jusqu'à 10. L'événement peut contenir moins d'éléments si le lot lu par Lambda à partir de la file d'attente avait moins d'éléments.
- Enabled – Désactivez la source de l'événement pour arrêter le traitement des éléments.

Note

Amazon SQS propose une offre gratuite perpétuelle pour les demandes. Au-delà de cette offre gratuite, Amazon SQS vous facture par tranche d'un million de demandes. Lorsque le mappage de

la source d'événement est actif, Lambda demande à la file d'attente d'obtenir les éléments. Pour plus d'informations sur la tarification, consultez [Tarification Amazon Simple Queue Service](#).

Pour gérer ultérieurement la configuration de la source d'événement, choisissez le déclencheur dans le concepteur.

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 \(p. 61\)](#) sur votre fonction, définissez un minimum de 5 exécutions simultanées pour réduire le risque d'erreurs de limitation lorsque Lambda appelle votre fonction.

API de mappage de la source d'événement

Pour gérer les mappages de source d'événement avec l'AWS CLI ou les kits SDK AWS, utilisez les actions d'API suivantes :

- [CreateEventSourceMapping \(p. 498\)](#)
- [ListEventSourceMappings \(p. 575\)](#)
- [GetEventSourceMapping \(p. 534\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)
- [DeleteEventSourceMapping \(p. 515\)](#)

L'exemple suivant utilise l'AWS CLI pour mapper une fonction nommée `my-function` à une file d'attente Amazon SQS spécifiée par son Amazon Resource Name (ARN), avec une taille de lot égale à 5.

```
$ aws lambda create-event-source-mapping --function-name my-function --batch-size 5 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 5,
    "EventSourceArn": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541139209.351,
    "State": "Creating",
    "StateTransitionReason": "USER_INITIATED"
}
```

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

Ce didacticiel suppose que vous avez quelques connaissances de la console Lambda et des opérations Lambda de base. Si ça n'est pas déjà fait, suivez les instructions de [Mise en route avec AWS Lambda \(p. 3\)](#) pour créer votre première fonction Lambda.

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

Créer le rôle d'exécution

Créez le [rôle d'exécution \(p. 33\)](#) qui donne à votre fonction l'autorisation d'accéder aux ressources AWS.

Pour créer un rôle d'exécution

1. Ouvrez la page [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 \(p. 292\)](#).

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
```

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

```
$ aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-sqs-role
```

Tester 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.

- Copiez le code JSON suivant dans un fichier et enregistrez-le sous le nom `input.txt`.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAENQZJLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

- Exécutez la commande `invoke` suivante.

```
$ aws lambda invoke --function-name ProcessSQSRecord \
--payload file://input.txt outputfile.txt
```

- 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

- Connectez-vous à la AWS Management Console et ouvrez la console Amazon SQS à l'adresse <https://console.aws.amazon.com/sqs/>.
- 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, détecte 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 --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

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-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

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 a pas de problèmes. Dans le cas contraire, vous recevez un message d'erreur.

Tester 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 \(p. 293\)](#)
- [Java \(p. 293\)](#)

- [C# \(p. 294\)](#)
- [Go \(p. 294\)](#)
- [Python \(p. 295\)](#)

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");
};
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#).

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 handler.java

```
package example;

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 Handler implements RequestHandler<SQSEvent, Void>{
    @Override
    public Void handleRequest(SQSEvent event, Context context)
    {
        for(SQSMessage msg : event.getRecords()){
            System.out.println(new String(msg.getBody()));
        }
    }
}
```

```
        return null;
    }
}
```

Dépendances

- aws-lambda-java-core
- aws-lambda-java-events

Générez le code avec les dépendances de bibliothèque Lambda pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda en Java \(p. 366\)](#).

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 HandleSQSEvent(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.";
        }
    }
}
```

Remplacez le fichier `Program.cs` d'un projet .NET Core par l'exemple ci-dessus. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans C# \(p. 414\)](#).

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)
}
```

Générez l'exécutable avec `go build` et créez un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Go \(p. 398\)](#).

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))
```

Compressez l'exemple de code pour créer un package de déploiement. Pour obtenir des instructions, consultez [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

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 \(p. 289\)](#). 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
Description: Example of processing messages on an SQS queue with Lambda
Resources:
  MySQSQueueFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      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 d'AWS Lambda avec AWS X-Ray

Vous pouvez utiliser AWS X-Ray pour visualiser les composants de votre application, identifier les goulets d'étranglement des performances et résoudre les demandes qui ont entraîné une erreur. Vos fonctions Lambda envoient des données de suivi X-Ray, et X-Ray les traitent pour générer une carte de service et des résumés de suivi consultables.



Si vous avez activé le suivi X-Ray dans un service qui appelle votre fonction, Lambda envoie automatiquement à X-Ray des suivis. Le service en amont, par exemple Amazon API Gateway, ou une application hébergée sur Amazon EC2 qui est instrumentée avec le kit SDK X-Ray, échantillonner les demandes entrantes et ajoute un en-tête de suivi qui indique à Lambda d'envoyer ou pas des suivis.

Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

Tarification

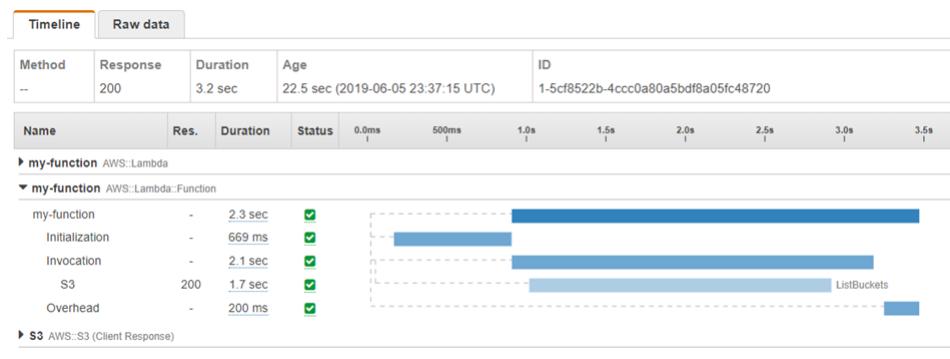
X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au [rôle d'exécution](#) (p. 33) de votre fonction. Sinon, ajoutez la stratégie `AWSXRayDaemonWriteAccess` au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Dans X-Ray, une trace enregistre des informations sur une demande qui est traitée par un ou plusieurs services. Les services enregistrent des segments qui contiennent des couches de sous-segments. Lambda enregistre un segment pour le service Lambda qui gère la demande d'appel, et un segment pour le travail effectué par la fonction. Le segment de fonction comprend des sous-segments pour `Initialization`, `Invocation` et `Overhead`.

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés `my-function`, mais l'un est de type `AWS::Lambda` et l'autre de type `AWS::Lambda::Function`. Le segment de fonction est développé pour afficher ses sous-segments.



Important

Dans Lambda, vous pouvez utiliser le kit SDK X-Ray pour étendre le sous-segment `Invocation` avec des sous-segments supplémentaires pour les appels en aval, les annotations et les métadonnées. Vous ne pouvez pas accéder directement au segment de fonction ou enregistrer une tâche effectuée en dehors de la portée d'appel du gestionnaire.

Consultez les rubriques suivantes pour consulter une introduction du suivi dans Lambda (en fonction du langage utilisé) :

- [Instrumentation du code Node.js dans AWS Lambda \(p. 324\)](#)
- [Instrumentation du code Python dans AWS Lambda \(p. 342\)](#)

- [Instrumentation du code Ruby dans AWS Lambda \(p. 358\)](#)
- [Instrumentation du code Java dans AWS Lambda \(p. 390\)](#)
- [Instrumentation du code Go dans AWS Lambda \(p. 408\)](#)
- [Instrumentation du code C # dans AWS Lambda \(p. 430\)](#)

Pour obtenir la liste complète des services prenant en charge l'instrumentation active, consultez [Services AWS pris en charge](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Autorisations du rôle d'exécution \(p. 298\)](#)
- [Démon AWS X-Ray \(p. 298\)](#)
- [Activation du suivi actif avec l'API Lambda \(p. 298\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 299\)](#)

Autorisations du rôle d'exécution

Lambda a besoin des autorisations suivantes pour envoyer des données de suivi X-Ray. Ajoutez-les au [rôle d'exécution \(p. 33\)](#) de la fonction.

- [xray:PutTraceSegments](#)
- [xray:PutTelemetryRecords](#)

Ces autorisations sont incluses dans la stratégie gérée [AWSXRayDaemonWriteAccess](#).

Démon AWS X-Ray

Au lieu d'envoyer des données de trace directement à l'API X-Ray, le kit SDK X-Ray utilise un processus de démon. Le démon AWS X-Ray est une application qui s'exécute dans l'environnement Lambda et qui écoute le trafic UDP contenant des segments et des sous-segments. Il met en mémoire tampon les données entrantes et les écrit par lots sur X-Ray, ce qui réduit la surcharge de traitement et de mémoire requise pour suivre les appels.

L'environnement d'exécution Lambda permet au démon d'utiliser jusqu'à 3 % de la mémoire configurée pour votre fonction ou 16 Mo, selon la valeur la plus élevée. Si votre fonction manque de mémoire pendant l'appel, l'environnement d'exécution arrête d'abord le processus de démon pour libérer de la mémoire.

Pour plus d'informations, consultez [Démon X-Ray](#) dans le Guide du développeur X-Ray.

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
```

```
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

Exemples d'applications Lambda

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation de divers langages et services AWS. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle AWS SAM et des ressources de support.

Node.js

Exemples d'applications Lambda dans Node.js

- [blank-nodejs](#) – Fonction Node.js qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi de AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.
- [nodejs-apig](#) – Fonction avec un point de terminaison d'API public qui traite un événement depuis API Gateway et renvoie une réponse HTTP.
- [rds-mysql](#) – Fonction qui relaie les requêtes vers une base de données MySQL pour RDS. Cet exemple inclut un VPC privé et une instance de base de données configurée avec un mot de passe dans AWS Secrets Manager.
- [list-manager](#) – Fonction qui traite les événements à partir d'un flux de données Amazon Kinesis et met à jour les listes agrégées dans Amazon DynamoDB. La fonction stocke un enregistrement de chaque événement dans une base de données MySQL pour RDS dans un VPC privé. Cet exemple inclut un VPC privé avec un point de terminaison VPC pour DynamoDB et une instance de base de données.
- [error-processor](#) – Fonction Node.js qui génère des erreurs pour un pourcentage spécifié de requêtes. Un abonnement CloudWatch Logs invoque une seconde fonction lorsqu'une erreur est enregistrée. La fonction processeur utilise le kit AWS SDK pour collecter des détails sur la demande et les stocke dans un compartiment Amazon S3.

Python

Exemples d'applications Lambda en Python

- [blank-python](#) – Fonction Python qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.

Ruby

Exemples d'applications Lambda dans Ruby

- [blank-ruby](#) – Fonction Ruby qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi de AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.

Java

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.

- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Go

Exemples d'applications Lambda en Go

- [blank-go](#) – Fonction Go qui montre l'utilisation des bibliothèques Go de Lambda, la journalisation, les variables d'environnement et le kit SDK AWS.

C#

Exemples d'applications Lambda en C#

- [blank-csharp](#) – Fonction C # qui montre l'utilisation des bibliothèques .NET de Lambda, la journalisation, les variables d'environnement, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [ec2-spot](#) – Fonction qui gère les demandes d'instance Spot dans Amazon EC2.

PowerShell

Exemples d'applications Lambda dans PowerShell

- [blank-powershell](#) – Fonction PowerShell qui montre l'utilisation de la journalisation, des variables d'environnement et du kit SDK AWS.

Pour déployer un exemple d'application, suivez les instructions de son fichier README. Pour de plus amples informations sur l'architecture et les cas d'utilisation d'une application, veuillez consulter les rubriques de ce chapitre.

Rubriques

- [Exemple d'application de fonction vide pour AWS Lambda \(p. 301\)](#)
- [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#)

Exemple d'application de fonction vide pour AWS Lambda

L'exemple d'application de fonction vide est une application de démarrage qui montre des opérations courantes dans Lambda avec une fonction qui appelle l'API Lambda. Il montre l'utilisation de la journalisation, des variables d'environnement, du suivi AWS X-Ray, des couches, des tests unitaires et du kit AWS SDK. Explorez cette application pour en savoir plus sur la création de fonctions Lambda dans votre langage de programmation, ou utilisez-la comme point de départ pour vos propres projets.



Les variantes de cet exemple d'application sont disponibles dans les langues suivantes :

Variantes

- Node.js – [blank-nodejs](#).
- Python – [blank-python](#).
- Ruby – [blank-ruby](#).
- Java – [blank-java](#).
- Go – [blank-go](#).
- C# – [blank-csharp](#).
- PowerShell – [blank-powershell](#).

Les exemples de cette rubrique mettent en évidence le code de la version Node.js, mais les détails sont généralement applicables à toutes les variantes.

Vous pouvez déployer l'exemple en quelques minutes avec l'AWS CLI et AWS CloudFormation. Suivez les instructions indiquées dans le fichier [README](#) pour le télécharger, le configurer et le déployer dans votre compte.

Sections

- [Architecture et code de gestionnaire \(p. 302\)](#)
- [Automatisation du déploiement avec AWS CloudFormation et l'AWS CLI \(p. 303\)](#)
- [Instrumentation avec le AWS X-Ray \(p. 306\)](#)
- [Gestion des dépendances avec des couches \(p. 307\)](#)

Architecture et code de gestionnaire

L'exemple d'application se compose d'un code de fonction, d'un modèle AWS CloudFormation et de ressources associées. Lorsque vous déployez l'exemple, vous utilisez les services AWS suivants :

- **AWS Lambda** – Exécute le code de la fonction, envoie les journaux à CloudWatch Logs et envoie les données de suivi à X-Ray. La fonction appelle également l'API Lambda pour obtenir des détails sur les limites et l'utilisation du compte dans la région actuelle.
- **AWS X-Ray** – Collecte les données de suivi, indexe les suivis de recherche et génère une cartographie des services.
- **Amazon CloudWatch** – Stocke les journaux et les métriques.
- **AWS Identity and Access Management (IAM)** – Octroie l'autorisation.
- **Amazon Simple Storage Service (Amazon S3)** – Stocke le package de déploiement de la fonction pendant le déploiement.
- **AWS CloudFormation** – Crée des ressources applicatives et déploie le code de la fonction.

Des frais standard s'appliquent pour chaque service. Pour plus d'informations, consultez [Tarification AWS](#).

Le code de la fonction affiche un flux de travail de base pour le traitement d'un événement. Le gestionnaire prend un événement Amazon Simple Queue Service (Amazon SQS) en entrée et parcourt les enregistrements qu'il contient, en consignant le contenu de chaque message. Il enregistre le contenu de l'événement, l'objet de contexte et les variables d'environnement. Ensuite, il effectue un appel avec le kit AWS SDK et transmet la réponse à l'exécution Lambda.

Example [blank-nodejs/function/index.js](#) – Code de gestionnaire

```
// Handler
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    console.log(record.body)
  })
  console.log('## ENVIRONMENT VARIABLES: ' + serialize(process.env))
  console.log('## CONTEXT: ' + serialize(context))
  console.log('## EVENT: ' + serialize(event))

  return getAccountSettings()
}

// Use SDK client
var getAccountSettings = function(){
  return lambda.getAccountSettings().promise()
}

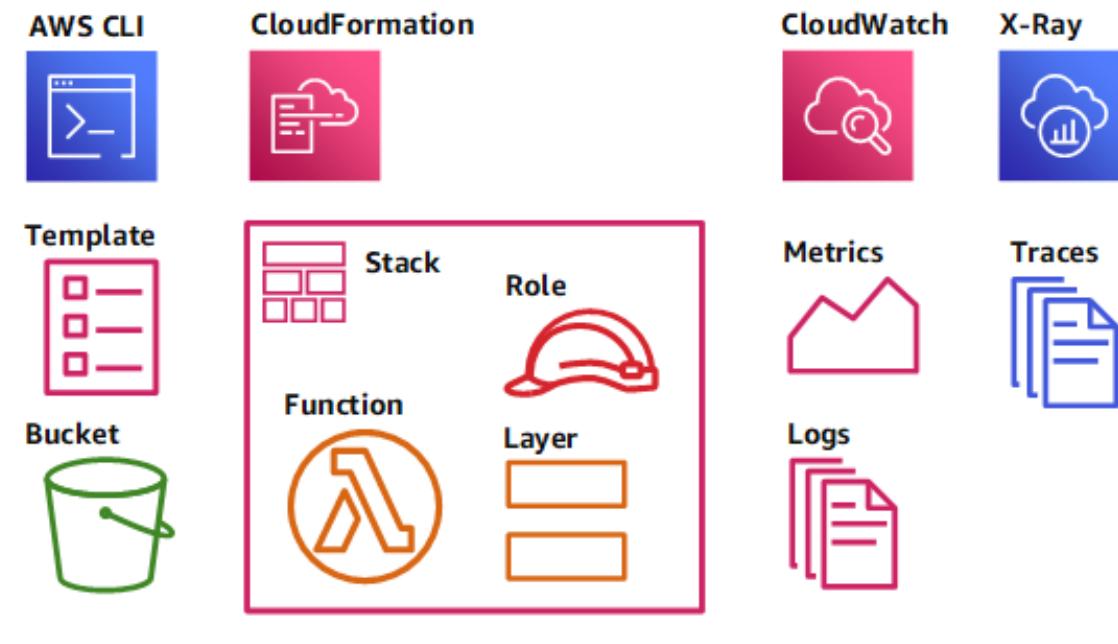
var serialize = function(object) {
  return JSON.stringify(object, null, 2)
}
```

Les types d'entrée/sortie pour le gestionnaire et la prise en charge de la programmation asynchrone varient en fonction de l'exécution. Dans cet exemple, la méthode de gestionnaire est `async`, donc dans Node.js cela signifie qu'il doit retourner une promesse à l'exécution. L'environnement d'exécution Lambda attend que la promesse soit résolue et renvoie la réponse à l'appelant. Si le code de fonction ou le client AWS SDK renvoie une erreur, le moteur d'exécution formate l'erreur dans un document JSON et le renvoie.

L'exemple d'application n'inclut pas de file d'attente Amazon SQS pour envoyer des événements, mais utilise un événement de Amazon SQS ([event.json](#)) pour illustrer le traitement des événements. Pour ajouter une file d'attente Amazon SQS à votre application, consultez [Utilisation de AWS Lambda avec Amazon SQS \(p. 285\)](#).

Automatisation du déploiement avec AWS CloudFormation et l'AWS CLI

Les ressources de l'exemple d'application sont définies dans un modèle AWS CloudFormation et déployées avec le AWS CLI. Le projet comprend des scripts shell simples qui automatisent le processus de configuration, de déploiement, d'appel et de déchirement de l'application.



Le modèle d'application utilise un type de ressource Modèle d'application sans serveur AWS (AWS SAM) pour définir le modèle. AWS SAM simplifie la création de modèles pour les applications sans serveur en automatisant la définition des rôles d'exécution, des API et d'autres ressources.

Le modèle définit les ressources dans la pile d'applications. Cela inclut la fonction, son rôle d'exécution et une couche Lambda qui fournit les dépendances de bibliothèque de la fonction. La pile n'inclut pas le compartiment que l'AWS CLI utilise pendant le déploiement ou le groupe de journaux CloudWatch Logs.

Example [blank-nodejs/template.yml](#) – Ressources sans serveur

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambdaReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Tracing: Active
      Layers:
        - !Ref libs
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
```

```
ContentUri: lib/.  
CompatibleRuntimes:  
  - nodejs12.x
```

Lorsque vous déployez l'application, AWS CloudFormation applique la transformation AWS SAM au modèle pour générer un modèle AWS CloudFormation avec des types standard tels que `AWS::Lambda::Function` et `AWS::IAM::Role`.

Example modèle traité

```
{  
  "AWSTemplateFormatVersion": "2010-09-09",  
  "Description": "An AWS Lambda application that calls the Lambda API.",  
  "Resources": {  
    "function": {  
      "Type": "AWS::Lambda::Function",  
      "Properties": {  
        "Layers": [  
          {  
            "Ref": "libs32xmpl61b2"  
          }  
        ],  
        "TracingConfig": {  
          "Mode": "Active"  
        },  
        "Code": {  
          "S3Bucket": "lambda-artifacts-6b000xmpl1e9bf2a",  
          "S3Key": "3d3axmpl473d249d039d2d7a37512db3"  
        },  
        "Description": "Call the AWS Lambda API",  
        "Tags": [  
          {  
            "Value": "SAM",  
            "Key": "lambda:createdBy"  
          }  
        ]  
      }  
    }  
  }  
}
```

Dans cet exemple, la propriété `Code` spécifie un objet dans un compartiment Amazon S3. Ceci correspond au chemin local de la propriété `CodeUri` dans le modèle de projet :

```
CodeUri: function/.
```

Pour télécharger les fichiers de projet vers Amazon S3, le script de déploiement utilise les commandes de la AWS CLI. La commande `cloudformation package` prétraite le modèle, télécharge les artefacts et remplace les chemins locaux par des emplacements d'objet Amazon S3. La commande `cloudformation deploy` déploie le modèle traité avec un jeu de modifications AWS CloudFormation.

Example `blank-nodejs/3-deploy.sh` – Package et déploiement

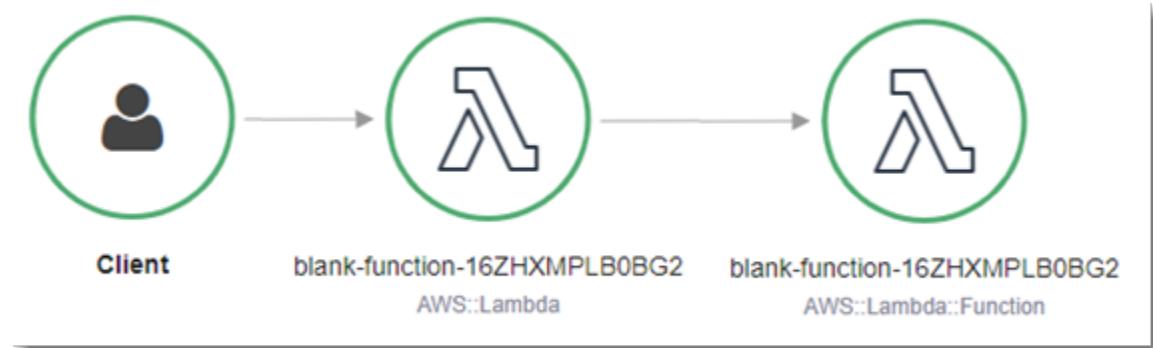
```
#!/bin/bash  
set -eo pipefail  
ARTIFACT_BUCKET=$(cat bucket-name.txt)  
aws cloudformation package --template-file template.yml --s3-bucket $ARTIFACT_BUCKET --output-template-file out.yml  
aws cloudformation deploy --template-file out.yml --stack-name blank-nodejs --capabilities CAPABILITY_NAMED_IAM
```

La première fois que vous exécutez ce script, il crée une pile AWS CloudFormation nommée `blank-nodejs`. Si vous apportez des modifications au code de fonction ou au modèle, vous pouvez l'exécuter à nouveau pour mettre à jour la pile.

Le script de nettoyage ([blank-nodejs/5-cleanup.sh](#)) supprime la pile et supprime éventuellement le compartiment de déploiement et les journaux de fonctions.

Instrumentation avec le AWS X-Ray

L'exemple de fonction est configuré pour le suivi avec [AWS X-Ray](#). Lorsque le mode de suivi est activé, Lambda enregistre les informations de synchronisation d'un sous-ensemble d'invocations et les envoie à X-Ray. X-Ray traite les données pour générer une carte de service qui affiche un nœud client et deux nœuds de service :



Le premier nœud de service (AWS::Lambda) représente le service Lambda, qui valide la demande d'invocation et l'envoie à la fonction. Le deuxième nœud, AWS::Lambda::Function, représente la fonction elle-même.

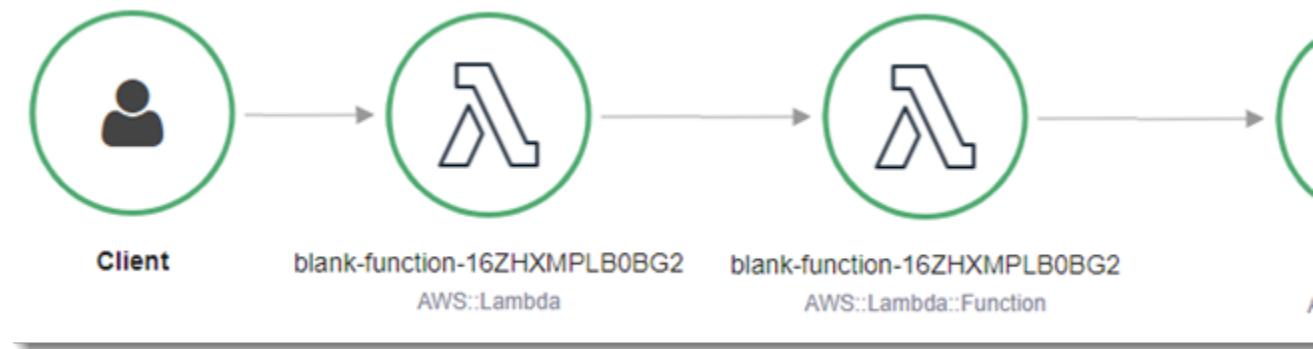
Pour enregistrer des détails supplémentaires, l'exemple de fonction utilise le kit X-Ray SDK. Avec des modifications minimales apportées au code de fonction, le kit X-Ray SDK enregistre des détails sur les appels effectués avec le kit AWS SDK aux services AWS.

Example [blank-nodejs/function/index.js](#) – Instrumentation

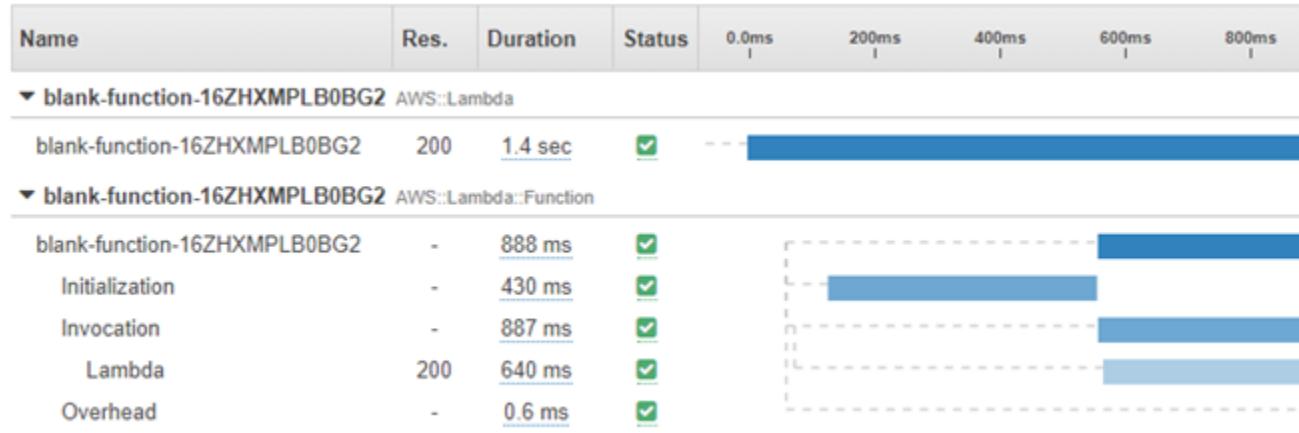
```
const AWSXRay = require('aws-xray-sdk-core')
const AWS = AWSXRay.captureAWS(require('aws-sdk'))

// Create client outside of handler to reuse
const lambda = new AWS.Lambda()
```

L'instrumentation du client AWS SDK ajoute un nœud supplémentaire à la carte de service et plus de détails dans les suivis. Dans cet exemple, la carte de service affiche l'exemple de fonction appelant l'API Lambda pour obtenir des détails sur le stockage et l'utilisation simultanée dans la région actuelle.



Le suivi affiche les détails de synchronisation de l'invocation, avec des sous-segments pour l'initialisation de la fonction, l'invocation et la surcharge. Le sous-segment d'appel comporte un sous-segment pour l'appel AWS SDK à l'opération d'API GetAccountSettings.



Vous pouvez inclure le kit X-Ray SDK et d'autres bibliothèques dans le package de déploiement de votre fonction, ou les déployer séparément dans une couche Lambda. Pour Node.js, Ruby et Python, le moteur d'exécution Lambda inclut le kit AWS SDK dans l'environnement d'exécution.

Gestion des dépendances avec des couches

Vous pouvez installer des bibliothèques localement et les inclure dans le package de déploiement sur lequel vous téléchargez Lambda, mais cela présente ses inconvénients. Des tailles de fichiers plus grandes entraînent une augmentation des temps de déploiement et peuvent vous empêcher de tester les modifications apportées à votre code de fonction dans la console Lambda. Pour conserver un package de déploiement de petite taille et éviter le téléchargement de dépendances qui n'ont pas changé, l'exemple d'application crée une [couche Lambda \(p. 76\)](#) et l'associe à la fonction.

Example [blank-nodejs/template.yml](#) – Couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambdaReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Tracing: Active
      Layers:
        - !Ref libs
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      ContentUri: lib/
      CompatibleRuntimes:
```

- nodejs12.x

Le script `2-build-layer.sh` installe les dépendances de la fonction avec `npm` et les place dans un dossier avec la [structure requise par le moteur d'exécution Lambda \(p. 79\)](#).

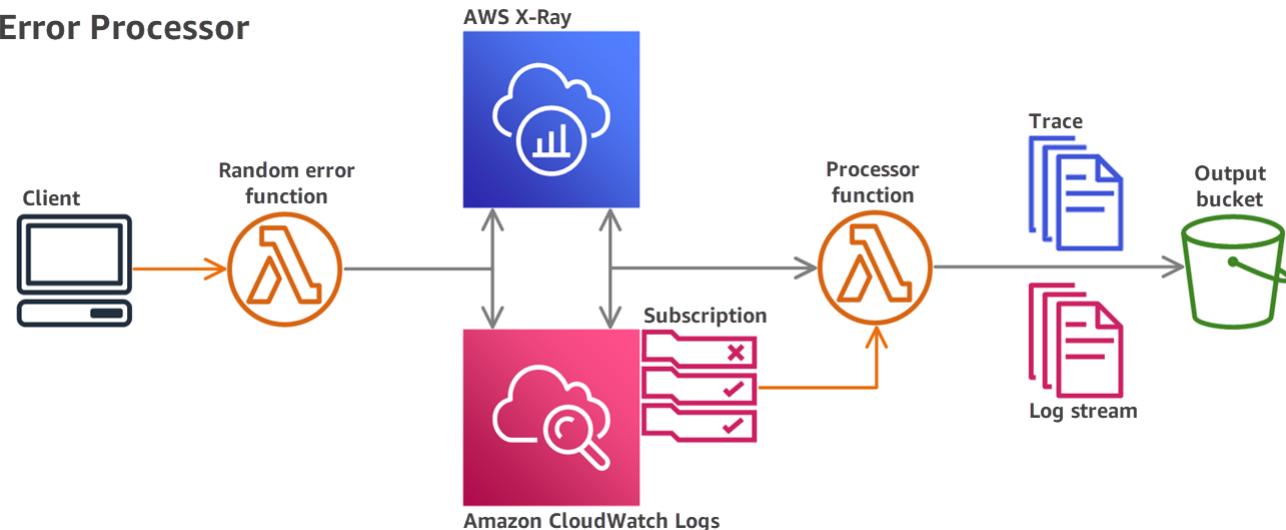
Example `2-build-layer.sh` – Préparation de la couche

```
#!/bin/bash
set -eo pipefail
mkdir -p lib/nodejs
rm -rf node_modules lib/nodejs/node_modules
npm install --production
mv node_modules lib/nodejs/
```

La première fois que vous déployez l'exemple d'application, AWS CLI crée un package pour la couche distinct de celui du code de fonction et déploie les deux. Pour les déploiements ultérieurs, l'archive de couches n'est téléchargée que si le contenu du dossier `lib` a changé.

Exemple d'application du processeur d'erreurs pour AWS Lambda

L'exemple d'application du processeur d'erreurs illustre l'utilisation de AWS Lambda pour gérer les événements à partir d'un [abonnement Amazon CloudWatch Logs \(p. 194\)](#). CloudWatch Logs vous permet d'appeler une fonction Lambda lorsqu'une entrée de journal correspond à un modèle. L'abonnement dans cette application surveille le groupe de journaux d'une fonction afin d'identifier les entrées qui contiennent le mot `ERROR`. Il appelle une fonction Lambda de processeur en réponse. La fonction de processeur récupère tous les flux de journaux et données de suivi pour la demande qui a provoqué l'erreur et les stocke en vue d'une utilisation ultérieure.



Le code de fonction est disponible dans les fichiers suivants.

- Erreur aléatoire – [random-error/index.js](#)
- Processeur – [processor/index.js](#)

Vous pouvez déployer l'exemple en quelques minutes avec l'AWS CLI et AWS CloudFormation. Suivez les instructions indiquées dans le fichier [README](#) pour le télécharger, le configurer et le déployer dans votre compte.

Sections

- [Structure d'événement et architecture \(p. 309\)](#)
- [Instrumentation avec AWS X-Ray \(p. 310\)](#)
- [Modèle AWS CloudFormation et ressources supplémentaires \(p. 311\)](#)

Structure d'événement et architecture

L'exemple d'application utilise les services AWS suivants.

- AWS Lambda – Exécute le code de la fonction, envoie les journaux à CloudWatch Logs et envoie les données de suivi à X-Ray.
- Amazon CloudWatch Logs – Collecte les journaux et appelle une fonction lorsqu'une entrée de journal correspond à un modèle de filtre.
- AWS X-Ray – Collecte les données de suivi, indexe les suivis de recherche et génère une cartographie des services.
- Amazon Simple Storage Service (Amazon S3) – Stocke les artefacts de déploiement et la sortie de l'application.

Des frais standard s'appliquent pour chaque service.

Une fonction Lambda dans l'application génère des erreurs de façon aléatoire. Lorsque CloudWatch Logs détecte le mot `ERROR` dans les journaux de la fonction, il envoie un événement à la fonction du processeur pour traitement.

Example événement de message CloudWatch Logs

```
{  
  "awslogs": {  
    "data": "H4sIAAAAAAAAHWQT0/DMAzFv0vEkbLYcdJkt4qVXmCDteIAm1DbZKjs  
+kdpB0Jo350MhsQFyVLsZ+un1/fJWje05asrPgbH5..."  
  }  
}
```

Une fois décodées, les données contiennent des détails sur l'événement de journal. La fonction utilise ces détails pour identifier le flux de journaux et analyse le message de journal pour obtenir l'ID de la demande qui a provoqué l'erreur.

Example données d'événement CloudWatch Logs décodées

```
{  
  "messageType": "DATA_MESSAGE",  
  "owner": "123456789012",  
  "logGroup": "/aws/lambda/lambda-error-processor-randomerror-1GD4SSDNACNP4",  
  "logStream": "2019/04/04/[LATEST]63311769a9d742f19cedf8d2e38995b9",  
  "subscriptionFilters": [  
    "lambda-error-processor-subscription-15OPDVQ59CG07"  
  ],  
  "logEvents": [  
    {  
      "id": "34664632210239891980253245280462376874059932423703429141",  
      "timestamp": 1554415868243,  
    }  
  ]  
}
```

```

        "message": "2019-04-04T22:11:08.243Z\tt1d2c1444-efd1-43ec-  

b16e-8fb2d37508b8\tERROR\n"  

    }  

}

```

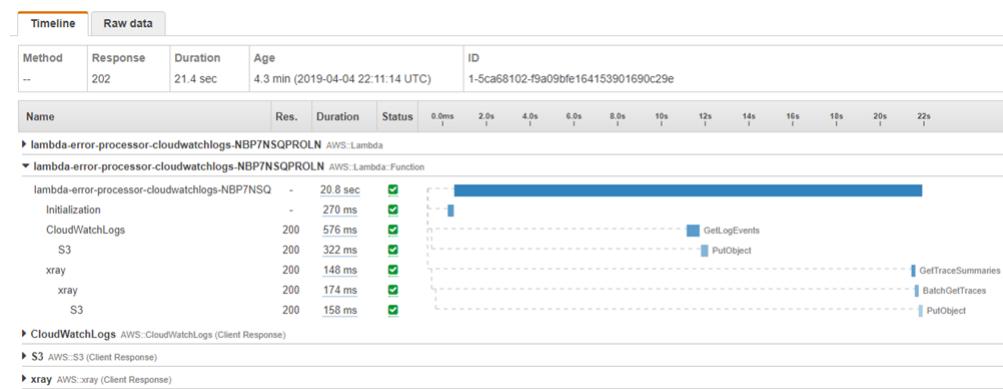
La fonction de processeur utilise les informations de l'événement CloudWatch Logs pour télécharger l'ensemble du flux de journaux et les données de suivi X-Ray pour une demande qui a provoqué une erreur. Elle stocke les deux dans un compartiment Amazon S3. Pour autoriser la finalisation du flux de journaux et des heures de suivi, la fonction attend pendant une courte période de temps avant d'accéder aux données.

Instrumentation avec AWS X-Ray

L'application utilise [AWS X-Ray \(p. 296\)](#) pour suivre les appels de la fonction et les appels que les fonctions passent aux services AWS. X-Ray utilise les données de suivi reçues des fonctions pour créer une cartographie des services qui vous aide à identifier les erreurs. Le mappage de service suivant illustre la fonction d'erreur aléatoire générant des erreurs pour des demandes. Il montre également la fonction de processeur appelant X-Ray, CloudWatch Logs et Amazon S3.



Les deux fonctions Node.js sont configurées pour un suivi actif dans le modèle et sont équipées du Kit SDK AWS X-Ray pour Node.js dans le code. Avec le suivi actif, les balises Lambda ajoutent un en-tête de suivi aux demandes entrantes et envoient des données de suivi avec l'heure à X-Ray. En outre, la fonction d'erreur aléatoire utilise le kit SDK X-Ray pour enregistrer l'ID de demande et les informations utilisateur dans des annotations. Les annotations sont attachées au suivi et vous pouvez les utiliser pour rechercher le suivi d'une demande spécifique.



La fonction de processeur obtient l'ID de demande à partir de l'événement CloudWatch Logs et utilise le AWS SDK for JavaScript afin de rechercher cette demande dans X-Ray. Elle utilise les clients de kit SDK

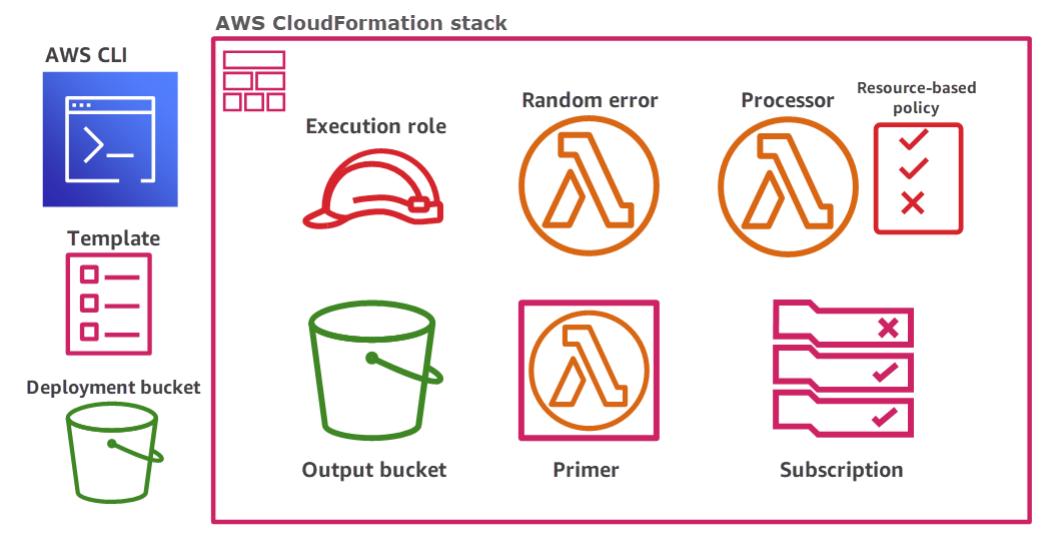
AWS instrumentés avec le kit SDK X-Ray pour télécharger le suivi et le flux de journaux. Elle les stocke ensuite dans le compartiment de sortie. Le kit SDK X-Ray enregistre ces appels et ils apparaissent comme des sous-segments dans le suivi.

Modèle AWS CloudFormation et ressources supplémentaires

L'application est mise en œuvre dans deux modules Node.js et déployée avec un modèle AWS CloudFormation et les scripts shell associés. Le modèle crée la fonction de processeur, la fonction d'erreur aléatoire et les ressources associées suivantes.

- Rôle d'exécution – Un rôle IAM qui accorde aux fonctions l'autorisation d'accéder à d'autres services AWS.
- Fonction de base – Une fonction supplémentaire qui appelle la fonction d'erreur aléatoire pour créer un groupe de journaux.
- Ressource personnalisée – Une ressource personnalisée AWS CloudFormation qui appelle la fonction de base pendant le déploiement afin de s'assurer de l'existence du groupe de journaux.
- Abonnement CloudWatch Logs – Un abonnement pour le flux de journaux qui déclenche la fonction de processeur lorsque le mot ERROR est consigné.
- Stratégie basée sur les ressources – Une instruction d'autorisation sur la fonction de processeur qui autorise CloudWatch Logs à l'invoquer.
- Compartiment Amazon S3 – Un emplacement de stockage pour la sortie de la fonction de processeur.

Affichez le modèle [template.yml](#) sur GitHub.



Pour contourner une limitation de l'intégration de Lambda avec AWS CloudFormation, le modèle crée une fonction supplémentaire qui s'exécute pendant les déploiements. Toutes les fonctions Lambda sont fournies avec un groupe de journaux CloudWatch Logs qui stocke la sortie des exécutions de fonctions. Toutefois, le groupe de journaux n'est pas créé tant que la fonction n'est pas appelée pour la première fois.

Pour créer l'abonnement, qui dépend de l'existence du groupe de journaux, l'application utilise une troisième fonction Lambda pour appeler la fonction d'erreur aléatoire. Le modèle inclut le code pour la fonction de base en ligne. Une ressource personnalisée AWS CloudFormation l'appelle pendant le déploiement. Les propriétés `DependsOn` s'assurent que le flux de journaux et la stratégie basée sur les ressources sont créés avant l'abonnement.

Création de fonctions Lambda avec Node.js

Vous pouvez exécuter Code JavaScript avec Node.js dans AWS Lambda. Lambda fournit des [exécutions \(p. 122\)](#) pour Node.js qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement comprenant le kit AWS SDK for JavaScript, avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez.

Lambda prend en charge les exécutions Node.js suivantes.

Environnements d'exécution Node.js

| Nom | Identifiant | Kit AWS SDK for JavaScript | Système d'exploitation |
|------------|-------------------------|----------------------------|------------------------|
| Node.js 12 | <code>nodejs12.x</code> | 2.631.0 | Amazon Linux 2 |
| Node.js 10 | <code>nodejs10.x</code> | 2.631.0 | Amazon Linux 2 |

Les fonctions Lambda utilisent un [rôle d'exécution \(p. 33\)](#) pour obtenir l'autorisation d'écrire des journaux dans Amazon CloudWatch Logs et d'accéder à d'autres services et ressources. 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 [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 Node.js

1. Ouvrez la [console Lambda](#).
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :
 - Nom – **my-function**.
 - Exécution – Node.js 12.x.
 - Rôle – Sélectionner un rôle existant.
 - Rôle existant – **lambda-role**.

4. Sélectionnez Créez une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Nom d'événement, saisissez **test**.
7. Sélectionnez Créez.
8. Pour exécuter la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `index.js`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'[éditeur de code \(p. 6\)](#) intégré. Choisissez Save (Enregistrer) pour enregistrer les changements. 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.

Le fichier `index.js` exporte une fonction nommée `handler` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la [fonction de gestionnaire \(p. 314\)](#) que Lambda appelle lorsque la fonction est invoquée. L'exécution de la fonction Node.js obtient les événements d'appels de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `index.handler`.

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 \(p. 316\)](#) et mettre à jour votre code dans la ligne de commande.

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda dans Node.js

- [blank-nodejs](#) – Fonction Node.js qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi de AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.
- [nodejs-apig](#) – Fonction avec un point de terminaison d'API public qui traite un événement depuis API Gateway et renvoie une réponse HTTP.
- [rds-mysql](#) – Fonction qui relaie les requêtes vers une base de données MySQL pour RDS. Cet exemple inclut un VPC privé et une instance de base de données configurée avec un mot de passe dans AWS Secrets Manager.
- [list-manager](#) – Fonction qui traite les événements à partir d'un flux de données Amazon Kinesis et met à jour les listes agrégées dans Amazon DynamoDB. La fonction stocke un enregistrement de chaque événement dans une base de données MySQL pour RDS dans un VPC privé. Cet exemple inclut un VPC privé avec un point de terminaison VPC pour DynamoDB et une instance de base de données.
- [error-processor](#) – Fonction Node.js qui génère des erreurs pour un pourcentage spécifié de requêtes. Un abonnement CloudWatch Logs invoque une seconde fonction lorsqu'une erreur est enregistrée. La fonction processeur utilise le kit AWS SDK pour collecter des détails sur la demande et les stocke dans un compartiment Amazon S3.

L'exécution de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'appel. L'[objet de contexte \(p. 318\)](#) contient des informations supplémentaires sur l'appel, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda s'accompagne d'un groupe de journaux CloudWatch Logs. L'exécution de la fonction envoie des informations sur chaque appel à CloudWatch Logs. Elle transmet tous les journaux que votre fonction génère (p. 319) pendant l'appel. Si votre fonction renvoie une erreur (p. 322), Lambda met en forme l'erreur et la renvoie au mécanisme d'appel.

Rubriques

- [Gestionnaire de fonctions AWS Lambda dans Node.js \(p. 314\)](#)
- [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#)
- [Objet de contexte AWS Lambda dans Node.js \(p. 318\)](#)
- [Journalisation des fonctions AWS Lambda dans Node.js \(p. 319\)](#)
- [Erreurs de fonction AWS Lambda dans Node.js \(p. 322\)](#)
- [Instrumentation du code Node.js dans AWS Lambda \(p. 324\)](#)

Gestionnaire de fonctions AWS Lambda dans Node.js

Le gestionnaire est la méthode de votre fonction Lambda qui traite les événements. Lorsque vousappelez une fonction, le [runtime \(p. 122\)](#) exécute la méthode du gestionnaire. Lorsque le gestionnaire se termine ou renvoie une réponse, il devient disponible pour gérer un autre événement.

Dans l'exemple suivant, la fonction enregistre le contenu de l'objet événement et renvoie l'emplacement des journaux.

Example index.js

```
exports.handler = async function(event, context) {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2))
  return context.logStreamName
}
```

Lorsque vous [configurez une fonction \(p. 53\)](#), la valeur du paramètre du gestionnaire est le nom du fichier et le nom du module du gestionnaire exporté, séparés par un point. La valeur par défaut dans la console et pour les exemples de ce guide est `index.handler`. Cela indique le module `handler` qui est exporté par `index.js`.

L'environnement d'exécution transmet trois arguments à la méthode du gestionnaire. Le premier argument est l'objet `event`, qui contient les informations de l'appelant. L'appelant transmet ces informations sous la forme d'une chaîne au format JSON lorsqu'il appelle [Invoke \(p. 565\)](#), et l'environnement d'exécution les convertit en objet. Lorsqu'un service AWS appelle votre fonction, la structure de l'événement [varie en fonction du service \(p. 155\)](#).

Le deuxième argument est l'[objet de contexte \(p. 318\)](#), qui contient des informations sur l'appel, la fonction et l'environnement d'exécution. Dans l'exemple précédent, la fonction obtient le nom du [flux de journaux \(p. 319\)](#) de l'objet de contexte et le renvoie au mécanisme d'appel.

Le troisième argument, `callback`, est une fonction que vous pouvez appeler dans les [gestionnaires non asynchrones \(p. 315\)](#) pour envoyer une réponse. La fonction de rappel accepte deux arguments : `un(e) Error` et une réponse. Lorsque vous l'appelez, Lambda attend que la boucle d'événement soit vide, puis renvoie la réponse ou l'erreur au mécanisme d'appel. L'objet de réponse doit être compatible avec `JSON.stringify`.

Pour les gestionnaires asynchrones, vous renvoyez une réponse, une erreur ou une promesse à l'exécution au lieu d'utiliser `callback`.

Gestionnaires asynchrones

Pour les gestionnaires asynchrones, vous pouvez utiliser `return` et `throw` afin d'envoyer une réponse ou une erreur, respectivement. Les fonctions doivent utiliser le mot-clé `async` pour utiliser ces méthodes pour renvoyer une réponse ou une erreur.

Si votre code effectue une tâche asynchrone, renvoyez une promesse pour vous assurer qu'il termine son exécution. Lorsque vous résolvez ou rejetez la promesse, Lambda envoie la réponse ou l'erreur au mécanisme d'appel.

Example Fichier index.js – Demande HTTP avec gestionnaire asynchrone et promesses

```
const https = require('https')
let url = "https://docs.aws.amazon.com/lambda/latest/dg/welcome.html"

exports.handler = async function(event) {
    const promise = new Promise(function(resolve, reject) {
        https.get(url, (res) => {
            resolve(res.statusCode)
        }).on('error', (e) => {
            reject(Error(e))
        })
    })
    return promise
}
```

Pour les bibliothèques qui renvoient une promesse, vous pouvez renvoyer cette promesse directement l'environnement d'exécution.

Example Fichier index.js – Kit SDK AWS avec gestionnaire asynchrone et promesses

```
const AWS = require('aws-sdk')
const s3 = new AWS.S3()

exports.handler = async function(event) {
    return s3.listBuckets().promise()
}
```

Gestionnaires non asynchrones

L'exemple suivant vérifie la fonction d'une URL et renvoie le code de statut au mécanisme d'appel.

Example Fichier index.js – Demande HTTP avec rappel

```
const https = require('https')
let url = "https://docs.aws.amazon.com/lambda/latest/dg/welcome.html"

exports.handler = function(event, context, callback) {
    https.get(url, (res) => {
        callback(null, res.statusCode)
    }).on('error', (e) => {
        callback(Error(e))
    })
}
```

Pour les gestionnaires non asynchrones, l'exécution de la fonction se poursuit jusqu'à ce que la **boucle d'événement** soit vide ou que la fonction arrive à expiration. La réponse n'est pas envoyée à l'appelant tant que toutes les tâches d'événement de boucle ne sont pas terminées. Si la fonction expire, une erreur

est renvoyée à la place. Vous pouvez configurer le runtime pour envoyer la réponse immédiatement en définissant [context.callbackWaitsForEmptyEventLoop \(p. 318\)](#) sur false.

Dans l'exemple suivant, la réponse Amazon S3 est renvoyée au mécanisme d'appel dès qu'elle est disponible. Le délai d'expiration de l'événement en boucle est gelé et reprend lors de l'appel suivant de la fonction.

Example Fichier index.js – callbackWaitsForEmptyEventLoop

```
const AWS = require('aws-sdk')
const s3 = new AWS.S3()

exports.handler = function(event, context, callback) {
    context.callbackWaitsForEmptyEventLoop = false
    s3.listBuckets(null, callback)
    setTimeout(function () {
        console.log('Timeout complete.')
    }, 5000)
}
```

Package de déploiement AWS Lambda dans Node.js

Un package de déploiement est une archive ZIP qui contient le code et les dépendances de la fonction. Vous devez créer un package de déploiement si vous utilisez l'API Lambda pour gérer des fonctions ou si vous avez besoin d'inclure des bibliothèques et des dépendances autres que le kit SDK AWS. Vous pouvez charger le package directement dans Lambda, ou vous pouvez utiliser un compartiment Amazon S3 puis le charger dans Lambda. Si le package de déploiement est supérieur à 50 Mo, vous devez utiliser Amazon S3.

Si vous utilisez l'[éditeur de console \(p. 6\)](#) Lambda pour créer votre fonction, la console gère le package de déploiement. Vous pouvez utiliser cette méthode tant que vous n'avez pas besoin d'ajouter des bibliothèques. Vous pouvez également l'utiliser pour mettre à jour une fonction qui a déjà des bibliothèques dans le package de déploiement, tant que la taille totale ne dépasse pas 3 MB.

Note

Pour que la taille de votre package de déploiement reste petite, empaquetez les dépendances de votre fonction en couches. Les couches vous permettent de gérer vos dépendances de manière indépendante, peuvent être utilisées par plusieurs fonctions et être partagées avec d'autres comptes. Pour plus d'informations, consultez [Couches AWS Lambda \(p. 76\)](#).

Sections

- [Mise à jour d'une fonction sans dépendances \(p. 316\)](#)
- [Mise à jour d'une fonction avec dépendances supplémentaires \(p. 317\)](#)

Mise à jour d'une fonction sans dépendances

Pour mettre à jour une fonction à l'aide de l'API Lambda, utilisez l'opération [UpdateFunctionCode \(p. 636\)](#). Créez une archive contenant le code de votre fonction et téléchargez-la à l'aide de l'AWS CLI.

Pour mettre à jour une fonction Node.js sans dépendance

1. Créez une archive ZIP.

```
~/my-function$ zip function.zip index.js
```

2. Utilisez la commande update-function-code pour charger le package.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Runtime": "nodejs12.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "index.handler",
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
    ...
}
```

Mise à jour d'une fonction avec dépendances supplémentaires

Si votre fonction dépend de bibliothèques autres que le SDK for JavaScript, installez-les dans un répertoire local avec [npm](#), et incluez-les dans votre package de déploiement. Vous pouvez également inclure le SDK for JavaScript si vous avez besoin d'une version plus récente que celle [incluse sur le runtime \(p. 312\)](#) ou pour vous assurer que la version ne changera pas à l'avenir.

Pour mettre à jour une fonction Node.js avec des dépendances.

1. Installez les bibliothèques dans le répertoire node_modules avec la commande `npm install`.

```
~/my-function$ npm install aws-xray-sdk
```

Cela crée une structure de dossiers similaire à ce qui suit.

```
~/my-function
### index.js
### node_modules
###  async
###  async-listener
###  atomic-batcher
###  aws-sdk
###  aws-xray-sdk
###  aws-xray-sdk-core
```

2. Créez un fichier ZIP contenant le contenu de votre dossier de projet.

```
~/my-function$ zip -r function.zip .
```

3. Utilisez la commande update-function-code pour charger le package.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
```

```
"FunctionName": "my-function",
"FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
"Runtime": "nodejs12.x",
"Role": "arn:aws:iam::123456789012:role/lambda-role",
"Handler": "index.handler",
"CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
"Version": "$LATEST",
"TracingConfig": {
    "Mode": "Active"
},
"RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
...
}
```

En plus de code et de bibliothèques, votre package de déploiement peut également contenir des fichiers exécutables et d'autres ressources. Pour plus d'informations, consultez les ressources suivantes :

- [Exécution de fichiers exécutables dans AWS Lambda](#)
- [Utilisation de packages et modules nodejs natifs dans AWS Lambda](#)

Objet de contexte AWS Lambda dans Node.js

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 314\)](#). 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()` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `functionName` – Nom de la fonction Lambda.
- `functionVersion` – [Version \(p. 70\)](#) de la fonction.
- `invokedFunctionArn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `memoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `awsRequestId` – Identifiant de la demande d'appel.
- `logGroupName` – Groupe de journaux de la fonction.
- `logStreamName` – Flux de journal pour l'instance de la fonction.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
 - `cognitoIdentityId` – Identité Amazon Cognito authentifiée.
 - `cognitoIdentityPoolId` – Pool d'identités Amazon Cognito qui a autorisé l'appel.
- `clientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
 - `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 – Personnalisez les valeurs qui sont définies par l'application mobile.
- `callbackWaitsForEmptyEventLoop` – Définissez ce paramètre sur faux pour envoyer la réponse immédiatement lorsque le [rappel \(p. 315\)](#) 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.

Dans l'exemple suivant, la fonction enregistre des informations de contexte et renvoie l'emplacement des journaux.

Example Fichier index.js

```
exports.handler = async function(event, context) {
    console.log('Remaining time: ', context.getRemainingTimeInMillis())
    console.log('Function name: ', context.functionName)
    return context.logStreamName
}
```

Journalisation des fonctions AWS Lambda dans Node.js

Votre fonction Lambda est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur [l'objet console](#) ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

Example Fichier index.js – Journalisation

```
exports.handler = async function(event, context) {
    console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
    console.info("EVENT\n" + JSON.stringify(event, null, 2))
    console.warn("Event not processed.")
    return context.logStreamName
}
```

Example format des journaux

```
START RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Version: $LATEST
2019-06-07T19:11:20.562Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO ENVIRONMENT VARIABLES
{
    "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
    "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/my-function",
    "AWS_LAMBDA_LOG_STREAM_NAME": "2019/06/07[$LATEST]e6f4a0c4241adcd70c262d34c0bbc85c",
    "AWS_EXECUTION_ENV": "AWS_Lambda_nodejs12.x",
    "AWS_LAMBDA_FUNCTION_NAME": "my-function",
    "PATH": "/var/lang/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin",
    "NODE_PATH": "/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/node_modules",
```

```
...
2019-06-07T19:11:20.563Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO EVENT
{
  "key": "value"
}
2019-06-07T19:11:20.564Z c793869b-ee49-115b-a5b6-4fd21e8dedac WARN Event not processed.
END RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac
REPORT RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Duration: 128.83 ms Billed Duration: 200 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms XRAY TraceId: 1-5d9d007f-0a8c7fd02xmpl480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled: true
```

Le runtime Node.js enregistre les lignes START, END et REPORT pour chaque appel. Il ajoute un horodatage, un ID de demande et un niveau de journalisation sur chaque entrée consignée par la fonction. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId – Pour les demandes suivies, l'[ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 320\)](#)
- [Utilisation de l'AWS CLI \(p. 321\)](#)
- [Suppression de journaux \(p. 322\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires

sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBUL0gUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTexZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"/\n/g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
```

```
}

{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Erreurs de fonction AWS Lambda dans Node.js

Lorsque votre code renvoie une erreur, Lambda génère une représentation JSON de l'erreur. Ce document d'erreur s'affiche dans le journal d'appels et, pour les appels synchrones, dans la sortie.

Example Fichier index.js – Erreur de référence

```
exports.handler = async function() {
  return x + 10
}
```

Ce code débouche sur une erreur de référence. 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.

```
{
```

```
"errorType": "ReferenceError",
"errorMessage": "x is not defined",
"trace": [
    "ReferenceError: x is not defined",
    "    at Runtime.exports.handler (/var/task/index.js:2:3)",
    "    at Runtime.handleOnce (/var/runtime/Runtime.js:63:25)",
    "    at process._tickCallback (internal/process/next_tick.js:68:7)"
]
}
```

Lorsque vous appelez la fonction à partir de la ligne de commande, l'AWS CLI divise la réponse en deux documents. Pour indiquer une erreur de fonction, la réponse affichée dans le terminal comprend un champ `FunctionError`. La réponse ou l'erreur renvoyée par la fonction est écrite sur le fichier de sortie.

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Consultez le fichier de sortie pour accéder au document d'erreur.

```
$ cat out.json
{"errorType": "ReferenceError", "errorMessage": "x is not defined", "trace": ["ReferenceError: x is not defined", "    at Runtime.exports.handler (/var/task/index.js:2:3)", "    at Runtime.handleOnce (/var/runtime/Runtime.js:63:25)", "    at process._tickCallback (internal/process/next_tick.js:68:7)"]}
```

Note

Le code d'état 200 (réussite) de la réponse de Lambda indique qu'aucune erreur ne s'est produite avec la demande que vous avez envoyée à Lambda. Pour les problèmes qui génèrent un code d'état d'erreur, veuillez consulter [Errors \(p. 567\)](#).

Lambda enregistre également 256 Ko de l'objet d'erreur dans les journaux de la fonction. Pour afficher les journaux lorsque vousappelez la fonction à partir de la ligne de commande, utilisez l'option `--log-type` et décodez la chaîne en base64 de la réponse.

```
$ aws lambda invoke --function-name my-function out.json --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8bbbfb91-a3ff-4502-b1b7-cb8f6658de64 Version: $LATEST
2019-06-05T22:11:27.082Z          8bbbfb91-a3ff-4502-b1b7-cb8f6658de64      ERROR      Invoke
Error      {"errorType": "ReferenceError", "errorMessage": "x is not defined", "stack": [
    "ReferenceError: x is not defined", "    at Runtime.exports.handler (/var/task/index.js:2:3)", "    at Runtime.handleOnce (/var/runtime/Runtime.js:63:25)", "    at process._tickCallback (internal/process/next_tick.js:68:7)"]
}
END RequestId: 8bbbfb91-a3ff-4502-b1b7-cb8f6658de64
REPORT RequestId: 8bbbfb91-a3ff-4502-b1b7-cb8f6658de64 Duration: 76.85 ms      Billed
Duration: 100 ms           Memory Size: 128 MB      Max Memory Used: 74 MB
```

Pour de plus amples informations sur les journaux, veuillez consulter [Journalisation des fonctions AWS Lambda dans Node.js \(p. 319\)](#).

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 [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).

Instrumentation du code Node.js dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle parcourt les ressources de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

Tarification

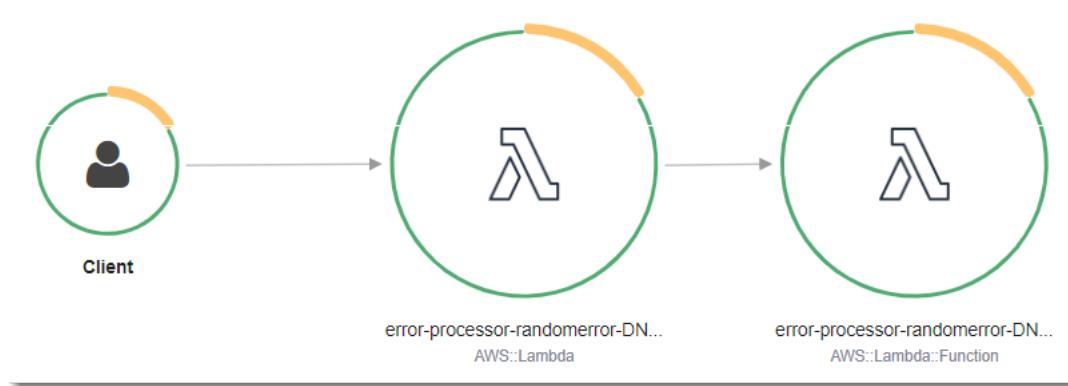
X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au [rôle](#)

d'exécution (p. 33) de votre fonction. Sinon, ajoutez la stratégie [AWSXRayDaemonWriteAccess](#) au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'environnement d'exécution (p. 18) de la fonction.



Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer des détails sur les appels que le gestionnaire effectue vers d'autres ressources et services, utilisez le kit SDK X-Ray pour Node.js. Pour obtenir ce kit SDK, ajoutez le package `aws-xray-sdk-core` aux dépendances de votre application.

Example [blank-nodejs/package.json](#)

```
{  
  "name": "blank-nodejs",  
  "version": "1.0.0",  
  "private": true,  
  "devDependencies": {  
    "aws-sdk": "2.631.0",  
    "jest": "25.4.0"  
  },  
  "dependencies": {  
    "aws-xray-sdk-core": "1.1.2"  
  },  
  "scripts": {  
    "test": "jest"  
  }  
}
```

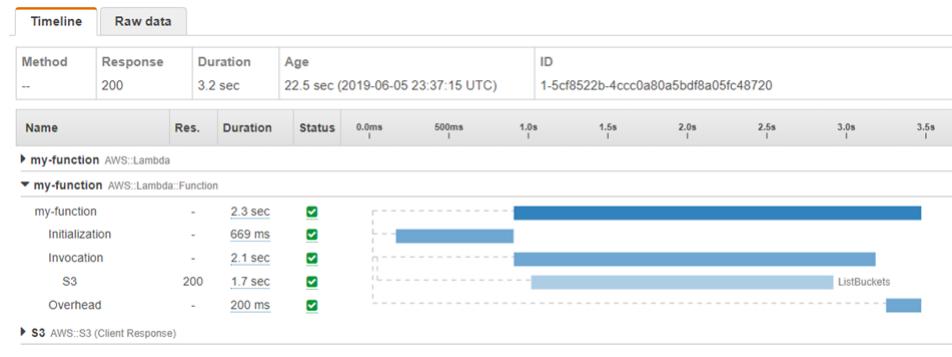
Pour instrumenter des clients AWS SDK, enveloppez la bibliothèque `aws-sdk` avec la méthode `captureAWS`.

Example [blank-nodejs/function/index.js](#) – Suivi d'un client AWS SDK

```
const AWSXRay = require('aws-xray-sdk-core')  
const AWS = AWSXRay.captureAWS(require('aws-sdk'))  
  
// Create client outside of handler to reuse  
const lambda = new AWS.Lambda()
```

```
// Handler
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    ...
  })
}
```

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type AWS::Lambda et l'autre de type AWS::Function. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez [Kit SDK X-Ray pour Node.js](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 326\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 327\)](#)
- [Stockage des dépendances d'exécution dans une couche \(p. 327\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
    ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
```

Stockage des dépendances d'exécution dans une couche

Si vous utilisez le kit X-Ray SDK pour instrumentiser les clients du kit AWS SDK pour votre fonction, votre package de déploiement peut devenir assez volumineux. Pour éviter de charger des dépendances d'exécution chaque fois que vous mettez à jour votre code de fonction, empaquetez-les dans une [couche Lambda \(p. 76\)](#).

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le kit SDK X-Ray pour Node.js.

Example [template.yml](#) – Couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/
      Tracing: Active
      Layers:
        - !Ref libs
```

```
...
libs:
  Type: AWS::Serverless::LayerVersion
  Properties:
    LayerName: blank-nodejs-lib
    Description: Dependencies for the blank sample app.
    ContentUri: lib/.
    CompatibleRuntimes:
      - nodejs12.x
```

Avec cette configuration, vous ne mettez à jour les fichiers JAR de bibliothèque que si vous modifiez vos dépendances d'exécution. Le package de déploiement de fonction contient uniquement votre code. Lorsque vous mettez à jour votre code de fonction, le temps de téléchargement est beaucoup plus rapide que si vous incluez des dépendances dans le package de déploiement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application [blank-nodejs](#).

Création de fonctions Lambda avec Python

Vous pouvez exécuter Python code dans AWS Lambda. Lambda fournit des [exécutions \(p. 122\)](#) pour Python qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement comprenant le kit SDK pour Python (Boto3), avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez.

Lambda prend en charge les exécutions Python suivantes.

Environnements d'exécution Python

| Nom | Identifiant | Kit SDK AWS pour Python | Système d'exploitation |
|------------|------------------------|-----------------------------------|------------------------|
| Python 3.8 | <code>python3.8</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux 2 |
| Python 3.7 | <code>python3.7</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |
| Python 3.6 | <code>python3.6</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |
| Python 2.7 | <code>python2.7</code> | boto3-1.12.49 botocore-1.15.49 | Amazon Linux |

Les fonctions Lambda utilisent un [rôle d'exécution \(p. 33\)](#) pour obtenir l'autorisation d'écrire des journaux dans Amazon CloudWatch Logs et d'accéder à d'autres services et ressources. 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 [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 Python

1. Ouvrez la [console Lambda](#).

2. Sélectionnez Créez une fonction.
3. Configurez les paramètres suivants :
 - Nom – **my-function**.
 - Exécution – Python 3.8.
 - Rôle – Sélectionnez un rôle existant.
 - Rôle existant – **lambda-role**.
4. Sélectionnez Créez une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Nom d'événement, saisissez **test**.
7. Sélectionnez Créez.
8. Pour exécuter la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `lambda_function`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'[éditeur de code \(p. 6\)](#) intégré. Choisissez Save (Enregistrer) pour enregistrer les changements. 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.

Le fichier `lambda_function` exporte une fonction nommée `lambda_handler` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la [fonction de gestionnaire \(p. 331\)](#) que Lambda appelle lorsque la fonction est invoquée. L'exécution de la fonction Python obtient les événements d'appels de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `lambda_function.lambda_handler`.

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 \(p. 332\)](#) et mettre à jour votre code dans la ligne de commande.

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda en Python

- [blank-python](#) – Fonction Python qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.

L'exécution de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'appel. L'[objet de contexte \(p. 336\)](#) contient des informations supplémentaires sur l'appel, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda s'accompagne d'un groupe de journaux CloudWatch Logs. L'exécution de la fonction envoie des informations sur chaque appel à CloudWatch Logs. Elle transmet tous les journaux [que votre fonction génère \(p. 337\)](#) pendant l'appel. Si votre fonction [renvoie une erreur \(p. 341\)](#), Lambda met en forme l'erreur et la renvoie au mécanisme d'appel.

Rubriques

- [Gestionnaire de fonctions AWS Lambda dans Python \(p. 331\)](#)
- [Package de déploiement AWS Lambda dans Python \(p. 332\)](#)
- [Objet de contexte AWS Lambda dans Python \(p. 336\)](#)
- [Journalisation des fonctions AWS Lambda dans Python \(p. 337\)](#)
- [Erreurs de fonction AWS Lambda dans Python \(p. 341\)](#)
- [Instrumentation du code Python dans AWS Lambda \(p. 342\)](#)

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`.

Lorsque vous appelez votre fonction, vous déterminez le contenu et la structure de l'événement. Lorsqu'un service AWS appelle votre fonction, la structure de l'événement varie en fonction du service. Pour de plus amples informations, veuillez consulter [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

- **context** – AWS Lambda utilise ce paramètre pour fournir les informations d'exécution au gestionnaire. Pour plus d'informations, veuillez consulter [Objet de contexte AWS Lambda dans Python \(p. 336\)](#).
- 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 vousappelez la fonction à l'aide de la console, cette dernière affiche la valeur renvoyée.
 - Si le gestionnaire renvoie des objets qui ne peuvent pas être sérialisées par `json.dumps`, le runtime renvoie une erreur.
 - Si le gestionnaire renvoie `None`, comme le font implicitement les fonctions Python sans une instruction `return`, le runtime 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.

Package de déploiement AWS Lambda dans Python

Un package de déploiement est une archive ZIP qui contient le code et les dépendances de la fonction. Vous devez créer un package de déploiement si vous utilisez l'API Lambda pour gérer des fonctions ou si vous avez besoin d'inclure des bibliothèques et des dépendances autres que le kit SDK AWS. Vous pouvez charger le package directement dans Lambda, ou vous pouvez utiliser un compartiment Amazon S3 puis le charger dans Lambda. Si le package de déploiement est supérieur à 50 Mo, vous devez utiliser Amazon S3.

Si vous utilisez l'[éditeur de console \(p. 6\)](#) Lambda pour créer votre fonction, la console gère le package de déploiement. Vous pouvez utiliser cette méthode tant que vous n'avez pas besoin d'ajouter des bibliothèques. Vous pouvez également l'utiliser pour mettre à jour une fonction qui a déjà des bibliothèques dans le package de déploiement, tant que la taille totale ne dépasse pas 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. L' interface de ligne de commande AWS SAM fournit également une option pour générer votre package de déploiement à l'intérieur d'une image Docker compatible avec l'environnement d'exécution Lambda. Pour obtenir des instructions, consultez [Création d'applications avec des dépendances](#) dans le manuel du développeur AWS SAM.

Sections

- [Prérequis \(p. 332\)](#)
- [Mise à jour d'une fonction sans dépendances \(p. 333\)](#)
- [Mise à jour d'une fonction avec dépendances supplémentaires \(p. 333\)](#)
- [Avec un environnement virtuel \(p. 334\)](#)

Prérequis

Ces instructions supposent que vous avez déjà une fonction. Si vous n'avez pas encore créé de fonction, veuillez consulter [Création de fonctions Lambda avec Python \(p. 329\)](#).

Pour suivre les procédures décrites dans ce manuel, vous aurez besoin d'un shell ou d'un terminal de ligne de commande pour exécuter des commandes. Les commandes sont affichées dans les listes précédées d'un symbole d'invite (\$) et du nom du répertoire actuel, le cas échéant :

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

Pour les commandes longues, un caractère d'échappement (\) est utilisé afin de fractionner la commande sur plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré. Sur Windows 10, vous pouvez [installer le sous-système Windows pour Linux](#) afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash.

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 Python sans dépendances

1. Créez une archive ZIP.

```
~/my-function$ zip function.zip lambda_function.py
  adding: lambda_function.py (deflated 17%)
```

2. Utilisez la commande `update-function-code` pour charger le package.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
fileb://function.zip
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Runtime": "python3.8",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 815,
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "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 pour Python (Boto3), installez-les dans un répertoire local avec [pip](#), et incluez-les dans votre package de déploiement.

Note

Pour les bibliothèques qui utilisent des modules d'extension écrits en C ou C++, créez votre package de déploiement dans un environnement Amazon Linux. Vous pouvez utiliser la [commande SAM CLI build](#), qui utilise Docker, ou construire votre package de déploiement sur Amazon EC2 ou AWS CodeBuild.

L'exemple suivant montre comment créer un package de déploiement qui inclut une bibliothèque de traitement d'images nommée Pillow.

Pour mettre à jour une fonction Python avec des dépendances

1. Installez les bibliothèques dans un nouveau répertoire package local au projet avec l'option `--target` de pip.

```
~/my-function$ pip install --target ./package 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
```

Note

Pour que --target fonctionne sur les [systèmes basés sur Debian](#) comme Ubuntu, vous pouvez aussi avoir besoin de transmettre l'indicateur --system pour empêcher les erreurs distutils.

- Créez une archive ZIP des dépendances.

```
~/my-function$ cd package
~/my-function/package$ zip -r9 ${OLDPWD}/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%)
  ...
```

- Ajoutez le code de votre fonction dans l'archive.

```
~/my-function/package$ cd $OLDPWD
~/my-function$ zip -g function.zip lambda_function.py
  adding: lambda_function.py (deflated 56%)
```

- Mettez à jour le code de la fonction.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
  fileb://function.zip
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Runtime": "python3.8",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "lambda_function.lambda_handler",
  "CodeSize": 2269409,
  "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2Ihsizc=",
  "Version": "$LATEST",
  "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

- Créez un environnement virtuel.

```
~/my-function$ virtualenv v-env
Using base prefix '/.local/python-3.7.0'
```

```
New python executable in v-env/bin/python3.8
Also creating executable in v-env/bin/python
Installing setuptools, pip, wheel...
done.
```

Note

Pour Python 3.3 et versions ultérieures, vous pouvez utiliser le [module venv](#) intégré pour créer un environnement virtuel, au lieu d'installer `virtualenv`.

```
~/my-function$ python3 -m venv v-env
```

2. Activez l'environnement.

```
~/my-function$ source v-env/bin/activate
(v-env) ~/my-function$
```

3. Installez les bibliothèques avec pip.

```
(v-env) ~/my-function$ pip install Pillow
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf81ecb6aea57b25ba00b240ee3fb0bd/
  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 de la bibliothèque.

```
~/my-function$ cd v-env/lib/python3.8/site-packages
~/my-function/v-env/lib/python3.8/site-packages$ zip -r9 ${OLDPWD}/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%)
  ...
```

En fonction de la bibliothèque, les dépendances peuvent apparaître dans `site-packages` ou `dist-packages`, et le premier dossier dans l'environnement virtuel peut être `lib` ou `lib64`. Vous pouvez utiliser la commande `pip show` pour localiser un package spécifique.

6. Ajoutez le code de votre fonction dans l'archive.

```
~/my-function/v-env/lib/python3.8/site-packages$ cd ${OLDPWD}
~/my-function$ zip -g function.zip lambda_function.py
  adding: lambda_function.py (deflated 56%)
```

7. Mettez à jour le code de la fonction.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
  fileb://function.zip
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Runtime": "python3.8",
```

```
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 5912988,
    "CodeSha256": "A2P0NUWq1J+LtSbkuP8tm9uNYqs1TAa3M76ptmZCw5g=",
    "Version": "$LATEST",
    "RevisionId": "5afdc7dc-2fcb-4ca8-8f24-947939ca707f",
    ...
}
```

Objet de contexte AWS Lambda dans Python

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 331\)](#). 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` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `function_name` – Nom de la fonction Lambda.
- `function_version` – [Version \(p. 70\)](#) de la fonction.
- `invoked_function_arn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `memory_limit_in_mb` – Quantité de mémoire allouée à la fonction.
- `aws_request_id` – Identifiant de la demande d'appel.
- `log_group_name` – Groupe de journaux de la fonction.
- `log_stream_name` – Flux de journal pour l'instance de la fonction.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
 - `cognito_identity_id` – Identité Amazon Cognito authentifiée.
 - `cognito_identity_pool_id` – Pool d'identités Amazon Cognito qui a autorisé l'appel.
- `client_context` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
 - `client.installation_id`
 - `client.app_title`
 - `client.app_version_name`
 - `client.app_version_code`
 - `client.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 \(p. 342\)](#) 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 est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser [la méthode print](#) ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

Example `lambda_function.py`

```
import json
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
```

Example format des journaux

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
         'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[LATEST]3893xmpl7fac4485b47bb75b671a283c',
         'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Duration: 15.74 ms Billed
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85 Sampled:
true
```

L'environnement d'exécution Python enregistre les lignes START, END et REPORT pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.

- Durée— Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY Traceld – Pour les demandes suivies, [l'ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 338\)](#)
- [Utilisation de l'AWS CLI \(p. 338\)](#)
- [Suppression de journaux \(p. 340\)](#)
- [Bibliothèque de journalisation \(p. 340\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
```

```
        "StatusCode": 200,
        "LogResult":
"U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
        "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's://"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003175,
            "message": "2019-06-05T19:30:03.175Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
            "ingestionTime": 1559763018355
        }
    ]
}
```

```
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r \\"key\\": \\"value\\r}\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration:
26.73 ms\tBilled Duration: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Bibliothèque de journalisation

Pour obtenir des journaux plus détaillés, utilisez la [bibliothèque de journalisation](#).

```
import json
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ)
    logger.info('## EVENT')
    logger.info(event)
```

La sortie de logger inclut le niveau de journal, l'horodatage et l'ID de demande.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO] 2020-01-31T22:12:58.534Z      1c8df7d3-xmpl-46da-9778-518e6eca8125      ## ENVIRONMENT
VARIABLES

[INFO] 2020-01-31T22:12:58.534Z      1c8df7d3-xmpl-46da-9778-518e6eca8125
    environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
    'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d',
    'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})

[INFO] 2020-01-31T22:12:58.535Z      1c8df7d3-xmpl-46da-9778-518e6eca8125      ## EVENT

[INFO] 2020-01-31T22:12:58.535Z      1c8df7d3-xmpl-46da-9778-518e6eca8125      {'key':
'value'}

END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
```

```
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Duration: 2.75 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861 Sampled: true
```

Erreurs de fonction AWS Lambda dans Python

Lorsque votre code renvoie une erreur, Lambda génère une représentation JSON de l'erreur. Ce document d'erreur s'affiche dans le journal d'appels et, pour les appels synchrones, dans la sortie.

Example Fichier lambda_function.py – Exception

```
def lambda_handler(event, context):
    return x + 10
```

Ce code débouche sur une erreur de nom. Lambda identifie l'erreur et génère un document JSON avec des champs pour le message d'erreur, le type et la trace de la pile.

```
{
    "errorMessage": "name 'x' is not defined",
    "errorType": "NameError",
    "stackTrace": [
        "  File \"/var/task/error_function.py\", line 2, in lambda_handler\n      return x +
10\n"
    ]
}
```

Lorsque vous appelez la fonction à partir de la ligne de commande, l'AWS CLI divise la réponse en deux documents. Pour indiquer une erreur de fonction, la réponse affichée dans le terminal comprend un champ `FunctionError`. La réponse ou l'erreur renvoyée par la fonction est écrite sur le fichier de sortie.

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Consultez le fichier de sortie pour accéder au document d'erreur.

```
$ cat out.json
{"errorMessage": "name 'x' is not defined", "errorType": "NameError", "stackTrace": [
  "  File \"/var/task/error_function.py\", line 2, in lambda_handler\n      return x + 10\n"]}
```

Note

Le code d'état 200 (réussite) de la réponse de Lambda indique qu'aucune erreur ne s'est produite avec la demande que vous avez envoyée à Lambda. Pour les problèmes qui génèrent un code d'état d'erreur, veuillez consulter [Errors \(p. 567\)](#).

Lambda enregistre également 256 Ko de l'objet d'erreur dans les journaux de la fonction. Pour afficher les journaux lorsque vousappelez la fonction à partir de la ligne de commande, utilisez l'option `--log-type` et décodez la chaîne en base64 de la réponse.

```
$ aws lambda invoke --function-name my-function out.json --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9 Version: $LATEST
    return x + 10/lambda_function.py", line 2, in lambda_handler
END RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9
REPORT RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9 Duration: 12.33 ms Billed Duration:
100 ms Memory Size: 128 MB Max Memory Used: 56 MB
```

Pour de plus amples informations sur les journaux, veuillez consulter [Journalisation des fonctions AWS Lambda dans Python \(p. 337\)](#).

Instrumentation du code Python dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle parcourt les ressources de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

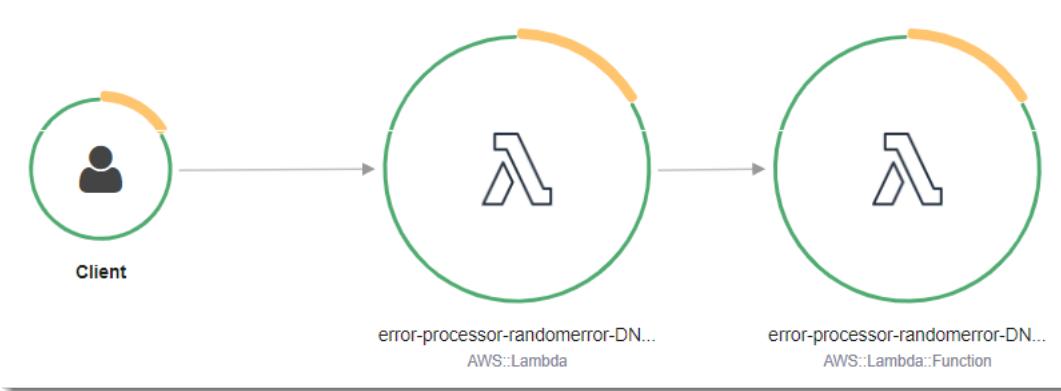
Tarification

X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au rôle d'exécution (p. 33) de votre fonction. Sinon, ajoutez la stratégie `AWSXRayDaemonWriteAccess` au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'environnement d'exécution (p. 18) de la fonction.



Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer des détails sur les appels que le gestionnaire effectue vers d'autres ressources et services, utilisez le kit SDK X-Ray pour Python. Pour obtenir ce kit SDK, ajoutez le package `aws-xray-sdk` aux dépendances de votre application.

Example [blank-python/function/requirements.txt](#)

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Pour instrumenter des clients AWS SDK, corrigez la bibliothèque `boto3` avec le module `aws_xray_sdk.core`.

Example [blank-python/function/lambda_function.py](#) – Suivi d'un client AWS SDK

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

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

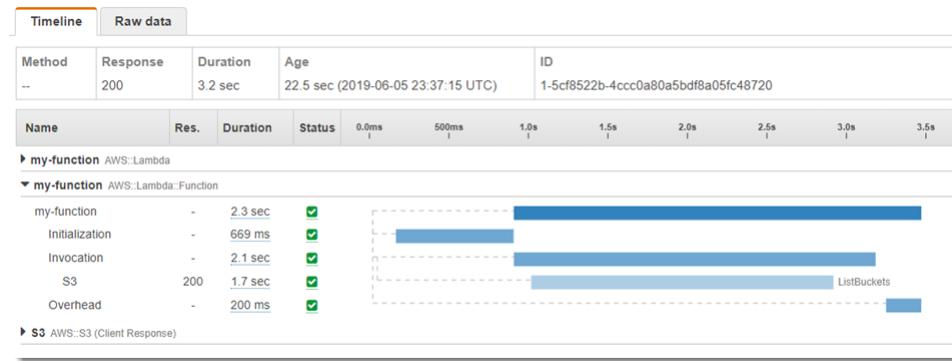
client = boto3.client('lambda')
```

```
client.get_account_settings()

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
    ...

```

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type AWS::Lambda et l'autre de type AWS::Function. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez [Kit SDK X-Ray pour Python](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 344\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 345\)](#)
- [Stockage des dépendances d'exécution dans une couche \(p. 345\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
    ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
```

Stockage des dépendances d'exécution dans une couche

Si vous utilisez le kit X-Ray SDK pour instrumentiser les clients du kit AWS SDK pour votre fonction, votre package de déploiement peut devenir assez volumineux. Pour éviter de charger des dépendances d'exécution chaque fois que vous mettez à jour votre code de fonction, empaquetez-les dans une [couche Lambda \(p. 76\)](#).

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le kit SDK X-Ray pour Python.

Example [template.yml](#) – Couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/
      Tracing: Active
      Layers:
        - !Ref libs
```

```
...
libs:
  Type: AWS::Serverless::LayerVersion
  Properties:
    LayerName: blank-python-lib
    Description: Dependencies for the blank-python sample app.
    ContentUri: package/.
    CompatibleRuntimes:
      - python3.8
```

Avec cette configuration, vous ne mettez à jour les fichiers JAR de bibliothèque que si vous modifiez vos dépendances d'exécution. Le package de déploiement de fonction contient uniquement votre code. Lorsque vous mettez à jour votre code de fonction, le temps de téléchargement est beaucoup plus rapide que si vous incluez des dépendances dans le package de déploiement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application [blank-python](#).

Création de fonctions Lambda avec Ruby

Vous pouvez exécuter Ruby code dans AWS Lambda. Lambda fournit des [exécutions \(p. 122\)](#) pour Ruby qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement comprenant le kit 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 exécutions Ruby suivantes.

Environnements d'exécution Ruby

| Nom | Identifiant | Kit AWS SDK for Ruby | Système d'exploitation |
|----------|----------------------|----------------------|------------------------|
| Ruby 2.7 | <code>ruby2.7</code> | 3.0.1 | Amazon Linux 2 |
| Ruby 2.5 | <code>ruby2.5</code> | 3.0.1 | Amazon Linux |

Les fonctions Lambda utilisent un [rôle d'exécution \(p. 33\)](#) pour obtenir l'autorisation d'écrire des journaux dans Amazon CloudWatch Logs et d'accéder à d'autres services et ressources. 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 [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 Créer une fonction.
3. Configurez les paramètres suivants :
 - Nom – **my-function**.
 - Exécution – Ruby 2.7.

- Rôle – Sélectionner un rôle existant.
 - Rôle existant – **lambda-role**.
4. Sélectionnez Crée une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Nom d'événement, saisissez **test**.
7. Sélectionnez Créeer.
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 \(p. 6\)](#) intégré. Choisissez Save (Enregistrer) pour enregistrer les changements. 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.

Le fichier `lambda_function.rb` exporte une fonction nommée `lambda_handler` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la [fonction de gestionnaire \(p. 349\)](#) que Lambda appelle lorsque la fonction est invoquée. L'exécution de la fonction Ruby obtient les événements d'appels de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `lambda_function.lambda_handler`.

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 \(p. 350\)](#) et mettre à jour votre code dans la ligne de commande.

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda dans Ruby

- [blank-ruby](#) – Fonction Ruby qui montre l'utilisation de la journalisation, des variables d'environnement, du suivi de AWS X-Ray, des couches, des tests unitaires et du kit SDK AWS.

L'exécution de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'appel. L'[objet de contexte \(p. 351\)](#) contient des informations supplémentaires sur l'appel, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda s'accompagne d'un groupe de journaux CloudWatch Logs. L'exécution de la fonction envoie des informations sur chaque appel à CloudWatch Logs. Elle transmet tous les journaux [que votre fonction génère \(p. 352\)](#) pendant l'appel. Si votre fonction [renvoie une erreur \(p. 356\)](#), Lambda met en forme l'erreur et la renvoie au mécanisme d'appel.

Rubriques

- [Gestionnaire de fonctions AWS Lambda en Ruby \(p. 349\)](#)
- [Package de déploiement AWS Lambda en Ruby \(p. 350\)](#)

- [Objet de contexte AWS Lambda en Ruby \(p. 351\)](#)
- [Journalisation des fonctions AWS Lambda en Ruby \(p. 352\)](#)
- [Erreurs de fonctions AWS Lambda en Ruby \(p. 356\)](#)
- [Instrumentation du code Ruby dans AWS Lambda \(p. 358\)](#)

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 \(p. 351\)](#) 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 de plus amples informations sur le contexte d'exécution de Lambda, consultez [Contexte d'exécution AWS Lambda \(p. 124\)](#).

Package de déploiement AWS Lambda en Ruby

Un package de déploiement est une archive ZIP qui contient le code et les dépendances de la fonction. Vous devez créer un package de déploiement si vous utilisez l'API Lambda pour gérer des fonctions ou si vous avez besoin d'inclure des bibliothèques et des dépendances autres que le kit SDK AWS. Vous pouvez charger le package directement dans Lambda, ou vous pouvez utiliser un compartiment Amazon S3 puis le charger dans Lambda. Si le package de déploiement est supérieur à 50 Mo, vous devez utiliser Amazon S3.

Si vous utilisez l'[éditeur de console \(p. 6\)](#) Lambda pour créer votre fonction, la console gère le package de déploiement. Vous pouvez utiliser cette méthode tant que vous n'avez pas besoin d'ajouter des bibliothèques. Vous pouvez également l'utiliser pour mettre à jour une fonction qui a déjà des bibliothèques dans le package de déploiement, tant que la taille totale ne dépasse pas 3 MB.

Note

Pour que la taille de votre package de déploiement reste petite, empaquetez les dépendances de votre fonction en couches. Les couches vous permettent de gérer vos dépendances de manière indépendante, peuvent être utilisées par plusieurs fonctions et être partagées avec d'autres comptes. Pour plus d'informations, consultez [Couches AWS Lambda \(p. 76\)](#).

Sections

- [Mise à jour d'une fonction sans dépendances \(p. 350\)](#)
- [Mise à jour d'une fonction avec dépendances supplémentaires \(p. 351\)](#)

Mise à jour d'une fonction sans dépendances

Pour mettre à jour une fonction à l'aide de l'API Lambda, utilisez l'opération [UpdateFunctionCode \(p. 636\)](#). Créez une archive contenant le code de votre fonction et téléchargez-la à l'aide de l'AWS CLI.

Pour mettre à jour une fonction Ruby sans dépendance

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 my-function --zip-file fileb://function.zip
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Runtime": "ruby2.5",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
    ...
}
```

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/2.7.0/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/build_info/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/cache/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/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 my-function --zip-file
fileb://function.zip
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Runtime": "ruby2.5",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 300,
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
    "Version": "$LATEST",
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
    ...
}
```

Objet de contexte AWS Lambda en Ruby

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 349\)](#). 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` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `function_name` – Nom de la fonction Lambda.
- `function_version` – [Version \(p. 70\)](#) de la fonction.
- `invoked_function_arn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `memory_limit_in_mb` – Quantité de mémoire allouée à la fonction.
- `aws_request_id` – Identifiant de la demande d'appel.
- `log_group_name` – Groupe de journaux de la fonction.
- `log_stream_name` – Flux de journal pour l'instance de la fonction.
- `deadline_ms` – Date d'expiration de l'exécution, exprimée en millisecondes au format horaire Unix.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `client_context` – (applications mobiles) Contexte client fourni à Lambda par l'application client.

Journalisation des fonctions AWS Lambda en Ruby

Votre fonction Lambda est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

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`. L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

Example `lambda_function.rb`

```
# lambda_function.rb

def handler(event:, context:)
    puts "## ENVIRONMENT VARIABLES"
    puts ENV.to_a
    puts "## EVENT"
    puts event.to_a
end
```

Pour obtenir des journaux plus détaillés, utilisez la [bibliothèque logger](#).

```
# lambda_function.rb

require 'logger'

def handler(event:, context:)
    logger = Logger.new($stdout)
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(ENV.to_a)
    logger.info('## EVENT')
    logger.info(event)
    event.to_a
end
```

La sortie de logger inclut l'horodatage, l'ID de processus, le niveau du journal et l'ID de demande.

```
I, [2019-10-26T10:04:01.689856 #8] INFO 6573a3a0-2fb1-4e78-a582-2c769282e0bd -- : ## EVENT
I, [2019-10-26T10:04:01.689874 #8] INFO 6573a3a0-2fb1-4e78-a582-2c769282e0bd -- :
 {"key1"=>"value1", "key2"=>"value2", "key3"=>"value3"}
```

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Example format des journaux

```
START RequestId: 50aba555-99c8-4b21-8358-644ee996a05f Version: $LATEST
```

```
## ENVIRONMENT VARIABLES
AWS_LAMBDA_FUNCTION_VERSION
$LATEST
AWS_LAMBDA_LOG_GROUP_NAME
/aws/lambda/my-function
AWS_LAMBDA_LOG_STREAM_NAME
2020/01/31/[ $LATEST]3f34xmpl069f4018b4a773bcfe8ed3f9
AWS_EXECUTION_ENV
AWS_Lambda_ruby2.5
...
## EVENT
key
value
END RequestId: 50aba555-xmpl-4b21-8358-644ee996a05f
REPORT RequestId: 50aba555-xmpl-4b21-8358-644ee996a05f Duration: 12.96 ms Billed Duration:
100 ms Memory Size: 128 MB Max Memory Used: 48 MB Init Duration: 117.86 ms
XRAY TraceId: 1-5e34a246-2a04xmpl0fa44eb60ea08c5f SegmentId: 454xmpl46ca1c7d3 Sampled: true
```

L'environnement d'exécution Ruby enregistre les lignes START, END et REPORT pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY Traceld – Pour les demandes suivies, [l'ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 338\)](#)
- [Utilisation de l'AWS CLI \(p. 338\)](#)
- [Suppression de journaux \(p. 340\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"/\//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Erreurs de fonctions AWS Lambda en Ruby

Lorsque votre code renvoie une erreur, Lambda génère une représentation JSON de l'erreur. Ce document d'erreur s'affiche dans le journal d'appels et, pour les appels synchrones, 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, l'AWS CLI divise la réponse en deux documents. Pour indiquer une erreur de fonction, la réponse affichée dans le terminal comprend un champ `FunctionError`. La réponse ou l'erreur renvoyée par la fonction est écrite sur le fichier de sortie.

```
$ aws lambda invoke --function-name my-function out.json  
{  
    "StatusCode": 200,  
    "FunctionError": "Unhandled",  
    "ExecutedVersion": "$LATEST"  
}
```

Consultez le fichier de sortie pour accéder au document d'erreur.

```
$ cat out.json  
{"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'"]}
```

Note

Le code d'état 200 (réussite) de la réponse de Lambda indique qu'aucune erreur ne s'est produite avec la demande que vous avez envoyée à Lambda. Pour les problèmes qui génèrent un code d'état d'erreur, veuillez consulter [Errors \(p. 567\)](#).

Lambda enregistre également 256 Ko de l'objet d'erreur dans les journaux de la fonction. Pour afficher les journaux lorsque vousappelez la fonction à partir de la ligne de commande, utilisez l'option `--log-type` et décodez la chaîne en base64 de la réponse.

```
$ aws lambda invoke --function-name my-function out.json --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 sur les journaux, consultez [Journalisation des fonctions AWS Lambda en Ruby \(p. 352\)](#).

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 [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).

Instrumentation du code Ruby dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle parcourt les ressources de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

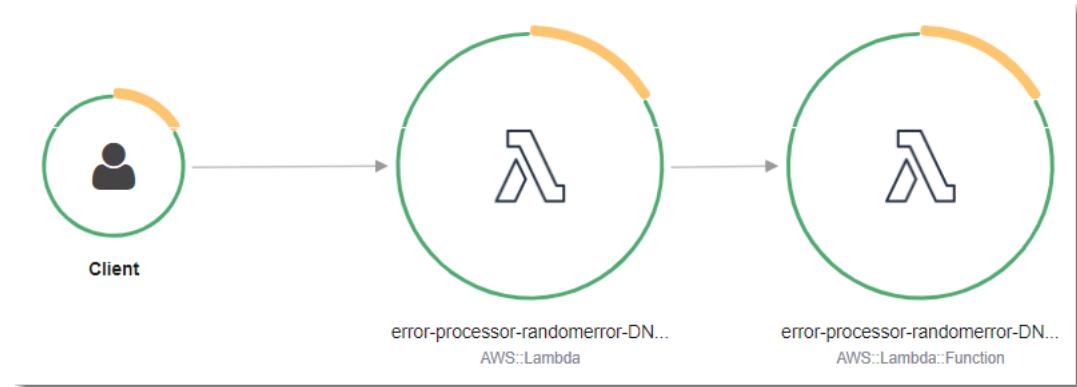
Tarification

X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au rôle d'exécution (p. 33) de votre fonction. Sinon, ajoutez la stratégie [AWSXRayDaemonWriteAccess](#) au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'environnement d'exécution (p. 18) de la fonction.



Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer des détails sur les appels que le gestionnaire effectue vers d'autres ressources et services, utilisez le kit SDK X-Ray pour Ruby. Pour obtenir ce kit SDK, ajoutez le package `aws-xray-sdk` aux dépendances de votre application.

Example [blank-ruby/function/Gemfile](#)

```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Pour instrumenter les clients AWS SDK, imposez le module `aws-xray-sdk/lambda` après avoir créé un client dans le code d'initialisation.

Example [blank-ruby/function/lambda_function.rb](#) – Suivi d'un client AWS SDK

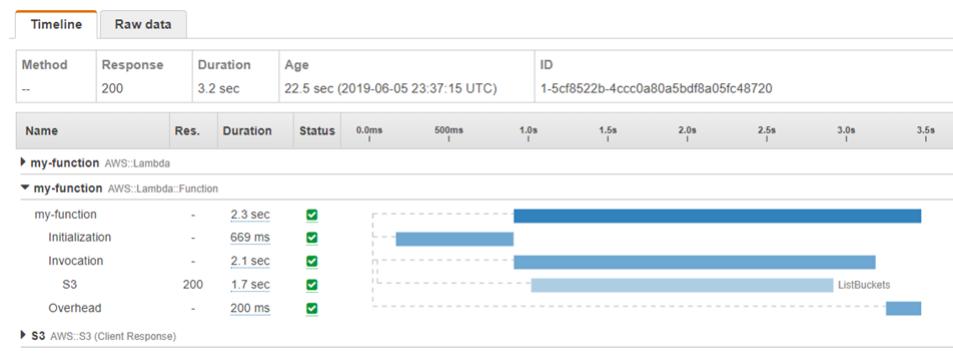
```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
```

...

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type AWS::Lambda et l'autre de type AWS::Lambda::Function. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez [Kit SDK X-Ray pour Ruby](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 360\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 361\)](#)
- [Stockage des dépendances d'exécution dans une couche \(p. 361\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

Stockage des dépendances d'exécution dans une couche

Si vous utilisez le kit X-Ray SDK pour instrumentiser les clients du kit AWS SDK pour votre fonction, votre package de déploiement peut devenir assez volumineux. Pour éviter de charger des dépendances d'exécution chaque fois que vous mettez à jour votre code de fonction, empaquetez-les dans une [couche Lambda](#) (p. 76).

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le kit SDK X-Ray pour Ruby.

Example [template.yml](#) – Couche de dépendances

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: function/.  
      Tracing: Active  
      Layers:  
        - !Ref libs  
      ...  
  libs:  
    Type: AWS::Serverless::LayerVersion  
    Properties:  
      LayerName: blank-ruby-lib
```

```
Description: Dependencies for the blank-ruby sample app.  
ContentUri: lib/.  
CompatibleRuntimes:  
- ruby2.5
```

Avec cette configuration, vous ne mettez à jour les fichiers JAR de bibliothèque que si vous modifiez vos dépendances d'exécution. Le package de déploiement de fonction contient uniquement votre code. Lorsque vous mettez à jour votre code de fonction, le temps de téléchargement est beaucoup plus rapide que si vous incluez des dépendances dans le package de déploiement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application [blank-ruby](#).

Création de fonctions Lambda avec Java

Vous pouvez exécuter Code Java dans AWS Lambda. Lambda fournit des [exécutions \(p. 122\)](#) pour Java qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement Amazon Linux comprenant les informations d'identification AWS d'un rôle AWS Identity and Access Management (IAM) que vous gérez.

Lambda prend en charge les exécutions Java suivantes.

Environnements d'exécution Java

| Nom | Identificateur | JDK | Système d'exploitation |
|---------|----------------|--------------------|------------------------|
| Java 11 | java11 | amazon-corretto-11 | Amazon Linux 2 |
| Java 8 | java8 | java-1.8.0-openjdk | Amazon Linux |

Les fonctions Lambda utilisent un [rôle d'exécution \(p. 33\)](#) pour obtenir l'autorisation d'écrire des journaux dans Amazon CloudWatch Logs et d'accéder à d'autres services et ressources. 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 [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 Java

1. Ouvrez la [console Lambda](#).
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :
 - Nom – **my-function**.
 - Exécution – Java 11.
 - Rôle – Sélectionner un rôle existant.
 - Rôle existant – **lambda-role**.

4. Sélectionnez Créez une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Nom d'événement, saisissez **test**.
7. Sélectionnez Créez.
8. Pour exécuter la fonction, choisissez Test.

La console crée une fonction Lambda avec une classe de gestionnaire nommée `Hello`. Étant donné que Java est un langage compilé, vous ne pouvez pas afficher ou modifier le code source dans la console Lambda, mais vous pouvez modifier sa configuration, l'appeler et configurer des déclencheurs.

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des [exemples d'applications \(p. 364\)](#) disponibles dans le référentiel GitHub de ce guide.

La classe `Hello` comporte une fonction nommée `handleRequest` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la [fonction de gestionnaire \(p. 372\)](#) que Lambda appelle lorsque la fonction est invoquée. L'exécution de la fonction Java obtient les événements d'appels de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `example.Hello::handleRequest`.

Pour mettre à jour le code de la fonction, vous créez 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 \(p. 366\)](#) et mettre à jour votre code dans la ligne de commande.

L'exécution de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'appel. L'[objet de contexte \(p. 376\)](#) contient des informations supplémentaires sur l'appel, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda s'accompagne d'un groupe de journaux CloudWatch Logs. L'exécution de la fonction envoie des informations sur chaque appel à CloudWatch Logs. Elle transmet tous les journaux que [votre fonction génère \(p. 378\)](#) pendant l'appel. Si votre fonction [renvoie une erreur \(p. 384\)](#), Lambda met en forme l'erreur et la renvoie au mécanisme d'appel.

Rubriques

- [Exemples d'applications Java pour AWS Lambda \(p. 364\)](#)
- [Package de déploiement AWS Lambda en Java \(p. 366\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Java \(p. 372\)](#)
- [Objet de contexte AWS Lambda en Java \(p. 376\)](#)
- [Journalisation des fonctions AWS Lambda dans Java \(p. 378\)](#)
- [Erreurs de fonction AWS Lambda dans Java \(p. 384\)](#)
- [Instrumentation du code Java dans AWS Lambda \(p. 390\)](#)
- [Création d'un package de déploiement à l'aide d'Eclipse \(p. 395\)](#)

Exemples d'applications Java pour AWS Lambda

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation de Java dans AWS Lambda. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle AWS CloudFormation et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Utilisez l'exemple d'application `blank-java` pour apprendre les bases ou comme point de départ pour votre propre application. Il montre l'utilisation des bibliothèques Java de Lambda, des variables d'environnement, du kit AWS SDK et du AWS X-Ray SDK. Il utilise une couche Lambda pour empaqueter ses dépendances séparément du code de fonction, ce qui accélère les temps de déploiement lorsque vous itérez votre code de fonction. Le projet nécessite une configuration minimale et peut être déployé à partir de la ligne de commande en moins d'une minute.



Les autres exemples d'applications montrent d'autres configurations de build, des interfaces de gestionnaire et des cas d'utilisation pour les services qui s'intègrent à Lambda. L'exemple `java-basic` montre une fonction avec des dépendances minimales. Vous pouvez utiliser cet exemple pour les cas où vous n'avez pas besoin de bibliothèques supplémentaires comme le kit SDK AWS, et pouvez représenter l'entrée et la sortie de votre fonction avec des types Java standard. Pour essayer un autre type de gestionnaire, vous pouvez simplement modifier le paramètre de gestionnaire sur la fonction.

Example [java-basic/src/main/java/example/HandlerStream.java](#) – Gestionnaire de flux

```
// Handler value: example.HandlerStream
public class HandlerStream implements RequestStreamHandler {
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context) throws IOException
    {
        LambdaLogger logger = context.getLogger();
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
        Charset.forName("US-ASCII")));
        PrintWriter writer = new PrintWriter(new BufferedWriter(new
        OutputStreamWriter(outputStream, Charset.forName("US-ASCII"))));
        try
        {
            HashMap event = gson.fromJson(reader, HashMap.class);
            writer.println(event);
        }
        catch (IOException e)
        {
            logger.error(e.getMessage());
        }
    }
}
```

```
logger.log("STREAM TYPE: " + inputStream.getClass().toString());
logger.log("EVENT TYPE: " + event.getClass().toString());
writer.write(gson.toJson(event));
...
```

Les exemples `java-events` et `java-events-v1sdk` montrent l'utilisation des types d'événements fournis par la bibliothèque `aws-lambda-java-events`. Ces types représentent les documents d'événement que les [services AWS \(p. 155\)](#) envoient à votre fonction. `java-events` inclut des gestionnaires pour les types qui ne nécessitent pas de dépendances supplémentaires. Pour les types d'événements comme `DynamodbEvent` qui nécessitent des types provenant de AWS SDK for Java, `java-events-v1sdk` inclut le SDK dans sa configuration de build.

Example [java-events-v1sdk/src/main/java/example/HandlerDynamoDB.java](#) – Enregistrements DynamoDB

```
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.amazonaws.services.dynamodbv2.model.Record;
...
// Handler value: example.HandlerDynamoDB
public class HandlerDynamoDB implements RequestHandler<DynamodbEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(HandlerDynamoDB.class);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(DynamodbEvent event, Context context)
    {
        String response = new String("200 OK");
        for (DynamodbStreamRecord record : event.getRecords()){
            logger.info(record.getEventID());
            logger.info(record.geteventName());
            logger.info(record.getDynamodb().toString());
        }
    ...
}
```

Pour de plus amples informations, veuillez consulter les autres rubriques de ce chapitre.

Package de déploiement AWS Lambda en Java

Un package de déploiement est une archive ZIP qui contient le code compilé de la fonction et les dépendances. Vous pouvez charger le package directement dans Lambda, ou vous pouvez utiliser un compartiment Amazon S3 puis le charger dans Lambda. Si le package de déploiement est supérieur à 50 Mo, vous devez utiliser Amazon S3.

AWS Lambda fournit les bibliothèques suivantes pour les fonctions Java :

- `com.amazonaws:aws-lambda-java-core` (obligatoire) – Définit les interfaces de méthode de gestion et l'objet de contexte que le moteur d'exécution transmet au gestionnaire. Si vous définissez vos propres types d'entrée, c'est la seule bibliothèque dont vous avez besoin.
- `com.amazonaws:aws-lambda-java-events` – Types d'entrée pour les événements des services qui appellent des fonctions Lambda.
- `com.amazonaws:aws-lambda-java-log4j2` – Une bibliothèque appender pour Log4j 2 que vous pouvez utiliser afin d'ajouter l'ID de requête pour l'appel en cours à vos [journaux de fonctions \(p. 378\)](#).

Ces bibliothèques sont disponibles via le [Référentiel central Maven](#). Ajoutez-les à votre définition de build comme suit.

Gradle

```
dependencies {  
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'  
    implementation 'com.amazonaws:aws-lambda-java-events:2.2.9'  
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'  
}
```

Maven

```
<dependencies>  
    <dependency>  
        <groupId>com.amazonaws</groupId>  
        <artifactId>aws-lambda-java-core</artifactId>  
        <version>1.2.1</version>  
    </dependency>  
    <dependency>  
        <groupId>com.amazonaws</groupId>  
        <artifactId>aws-lambda-java-events</artifactId>  
        <version>2.2.9</version>  
    </dependency>  
    <dependency>  
        <groupId>com.amazonaws</groupId>  
        <artifactId>aws-lambda-java-log4j2</artifactId>  
        <version>1.2.0</version>  
    </dependency>  
</dependencies>
```

Pour créer un package de déploiement, compilez votre code de fonction et vos dépendances dans un seul fichier ZIP ou Java Archive (JAR). Pour Gradle, [utilisez le type de build Zip \(p. 368\)](#). Pour Maven, [utilisez le plug-in Maven Shade \(p. 368\)](#).

Note

Pour que la taille de votre package de déploiement reste petite, empaquetez les dépendances de votre fonction en couches. Les couches vous permettent de gérer vos dépendances de manière indépendante, peuvent être utilisées par plusieurs fonctions et être partagées avec d'autres comptes. Pour plus d'informations, consultez [Couches AWS Lambda \(p. 76\)](#).

Vous pouvez télécharger votre package de déploiement à l'aide de la console Lambda, de l'API Lambda ou de l'AWS SAM.

Pour télécharger un package de déploiement avec la console Lambda

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous Code de fonction, choisissez Charger.
4. Charger un package de déploiement
5. Choisissez Enregistrer.

Sections

- [Création d'un package de déploiement avec Gradle \(p. 368\)](#)
- [Création d'un package de déploiement avec Maven \(p. 368\)](#)
- [Chargement d'un package de déploiement avec l'API Lambda \(p. 370\)](#)
- [Chargement d'un package de déploiement avec AWS SAM \(p. 371\)](#)

Création d'un package de déploiement avec Gradle

Utilisez le type de build `Zip` pour créer un package de déploiement avec le code et les dépendances de votre fonction.

Example `build.gradle` – Tâche de build

```
task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtimeClasspath
    }
}
```

Cette configuration de build produit un package de déploiement dans le dossier `build/distributions`. La tâche `compileJava` compile les classes de votre fonction. Les tâches `processResources` copient les bibliothèques du chemin de classe de la build dans un dossier nommé `lib`.

Example `build.gradle` – Dépendances

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.10.73')
    implementation 'software.amazon.awssdk:lambda'
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
    implementation 'com.amazonaws:aws-lambda-java-events:2.2.9'
    implementation 'com.google.code.gson:gson:2.8.6'
    implementation 'org.apache.logging.log4j:log4j-api:2.13.0'
    implementation 'org.apache.logging.log4j:log4j-core:2.13.0'
    runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.13.0'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'
}
```

Lambda charge les fichiers JAR dans l'ordre alphabétique Unicode. Si plusieurs fichiers JAR dans le dossier `lib` contiennent la même classe, le premier fichier est utilisé. Vous pouvez utiliser le script shell ci-après pour identifier les classes en double.

Example `test-zip.sh`

```
mkdir -p expanded
unzip path/to/my/function.zip -d expanded
find ./expanded/lib -name '*.jar' | xargs -n1 zipinfo -1 | grep '.*.class' | sort | uniq -c
| sort
```

Création d'un package de déploiement avec Maven

Pour créer un package de déploiement avec Maven, utilisez le [plug-in Maven Shade](#). Le plug-in crée un fichier JAR qui contient le code de fonction compilé et toutes ses dépendances.

Example `pom.xml` – Configuration du plug-in

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
```

```
<version>3.2.2</version>
<configuration>
    <createDependencyReducedPom>false</createDependencyReducedPom>
</configuration>
<executions>
    <execution>
        <phase>package</phase>
        <goals>
            <goal>shade</goal>
        </goals>
    </execution>
</executions>
</plugin>
```

Pour créer le package de déploiement, utilisez la commande `mvn package`.

```
[INFO] Scanning for projects...
[INFO] -----< com.example:java-maven >-----
[INFO] Building java-maven-function 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
...
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ java-maven ---
[INFO] Building jar: target/java-maven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.2.2:shade (default) @ java-maven ---
[INFO] Including com.amazonaws:aws-lambda-java-core:jar:1.2.1 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:2.2.9 in the shaded jar.
[INFO] Including joda-time:joda-time:jar:2.6 in the shaded jar.
[INFO] Including com.google.code.gson:gson:jar:2.8.6 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing target/java-maven-1.0-SNAPSHOT.jar with target/java-maven-1.0-SNAPSHOT-
shaded.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.321 s
[INFO] Finished at: 2020-03-03T09:07:19Z
[INFO] -----
```

Cette commande génère un fichier JAR dans le dossier `target`.

Si vous utilisez la bibliothèque appender (`aws-lambda-java-log4j2`), vous devez également configurer un transformateur pour le plug-in Maven Shade. La bibliothèque de transformateurs combine les versions d'un fichier cache qui apparaissent à la fois dans la bibliothèque appender et dans Log4j.

Example pom.xml – Configuration du plug-in avec l'appender Log4j 2

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>3.2.2</version>
    <configuration>
        <createDependencyReducedPom>false</createDependencyReducedPom>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
            <configuration>
                <transformers>
```

```
<transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCacheFileTransformer"
    </transformer>
</transformers>
</configuration>
</execution>
</executions>
<dependencies>
<dependency>
<groupId>com.github.edwgiz</groupId>
<artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
<version>2.13.0</version>
</dependency>
</dependencies>
</plugin>
```

Chargement d'un package de déploiement avec l'API Lambda

Pour mettre à jour le code d'une fonction avec le AWS CLI ou le kit AWS SDK, utilisez l'opération d'API [UpdateFunctionCode \(p. 636\)](#). Pour l'interface de ligne de commande AWS, utilisez la commande `update-function-code`. La commande suivante charge un package de déploiement nommé `my-function.zip` dans le répertoire courant.

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
fileb://my-function.zip
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "java8",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "example.Handler",
  "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "Active"
  },
  "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
  ...
}
```

Si votre package de déploiement est plus grand que 50 MB, vous ne pouvez pas le charger directement. Chargez-le dans un compartiment Amazon S3 et pointez Lambda sur l'objet. Les exemples de commandes suivants téléchargent un package de déploiement dans un compartiment nommé `my-bucket` et l'utilisent pour mettre à jour le code d'une fonction.

```
~/my-function$ aws s3 cp my-function.zip s3://my-bucket
upload: my-function.zip to s3://my-bucket/my-function
~/my-function$ aws lambda update-function-code --function-name my-function \
--s3-bucket my-bucket --s3-key my-function.zip
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "java8",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "example.Handler",
  "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "Active"
```

```
    },
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
    ...
}
```

Vous pouvez utiliser cette méthode pour charger des packages de fonctions jusqu'à 250 MB (décompressé).

Chargement d'un package de déploiement avec AWS SAM

Vous pouvez utiliser le Modèle d'application sans serveur AWS pour automatiser les déploiements de votre code de fonction, de votre configuration et de vos dépendances. AWS SAM est une extension de AWS CloudFormation qui fournit une syntaxe simplifiée pour définir des applications sans serveur. L'exemple de modèle suivant définit une fonction avec un package de déploiement dans le répertoire `build/distributions` utilisé par Gradle.

Example template.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: build/distributions/java-basic.zip
      Handler: example.Handler
      Runtime: java8
      Description: Java function
      MemorySize: 512
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambdaReadOnlyAccess
        - AWSXrayWriteOnlyAccess
        - AWSLambdaVPCAccessExecutionRole
      Tracing: Active
```

Pour créer la fonction, utilisez les commandes `package` et `deploy`. Ces commandes sont des personnalisations de l'interface de ligne de commande AWS. Elles encapsulent d'autres commandes pour télécharger le package de déploiement Amazon S3, réécrivent le modèle avec l'URI de l'objet et mettent à jour le code de la fonction.

L'exemple de script suivant exécute une build Gradle et télécharge le package de déploiement qu'il crée. Il crée une pile AWS CloudFormation la première fois que vous l'exéutez. Si la pile existe déjà, le script la met à jour.

Example deploy.sh

```
#!/bin/bash
set -eo pipefail
aws cloudformation package --template-file template.yml --s3-bucket MY_BUCKET --output-template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name java-basic --capabilities CAPABILITY_NAMED_IAM
```

Pour obtenir un exemple complet de travail, consultez les exemples d'applications suivants.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Gestionnaire de fonctions AWS Lambda dans Java

Le gestionnaire de votre fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est appelée, Lambda exécute la méthode du gestionnaire. Lorsque le gestionnaire se termine ou renvoie une réponse, il devient disponible pour gérer un autre événement.

Dans l'exemple suivant, une classe nommée `Handler` définit une méthode de gestionnaire nommée `handleRequest`. La méthode de gestionnaire prend un événement et un objet de contexte en entrée et renvoie une chaîne.

Example [handler.java](#)

```
package example;
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.RequestHandler
import com.amazonaws.services.lambda.runtime.LambdaLogger
...

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Map<String, String> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("200 OK");
        // log execution details
        logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        logger.log("CONTEXT: " + gson.toJson(context));
        // process event
        logger.log("EVENT: " + gson.toJson(event));
        logger.log("EVENT TYPE: " + event.getClass().toString());
        return response;
    }
}
```

Le [moteur d'exécution Lambda \(p. 122\)](#) reçoit un événement sous la forme d'une chaîne au format JSON et le convertit en objet. Il transmet l'objet d'événement à votre gestionnaire de fonction avec un objet de contexte qui fournit des détails sur l'appel et la fonction. Vous indiquez au moteur d'exécution quelle méthode appeler en définissant le paramètre de gestionnaire sur la configuration de votre fonction.

Formats du gestionnaire

- `package.Class::method` – Format complet. Par exemple : `example.Handler::handleRequest`.
- `package.Class` – Format abrégé pour les fonctions qui implémentent une [interface de gestionnaire](#) (p. 374). Par exemple : `example.Handler`.

Vous pouvez ajouter du [code d'initialisation](#) (p. 19) en dehors de votre méthode de gestionnaire pour réutiliser les ressources sur plusieurs appels. Lorsque le moteur d'exécution charge votre gestionnaire, il exécute du code statique et le constructeur de classe. Les ressources créées pendant l'initialisation restent en mémoire entre les appels et peuvent être réutilisées par le gestionnaire des milliers de fois.

Dans l'exemple suivant, l'enregistreur, le sérialiseur et le client SDK AWS sont créés lorsque la fonction sert son premier événement. Les événements suivants servis par la même instance de fonction sont beaucoup plus rapides car ces ressources existent déjà.

Example `Handler.java` – Code d'initialisation

```
// Handler value: example.Handler
public class Handler implements RequestHandler<SQSEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    private static final Gson gson = new GsonBuilder().setPrettyPrinting().create();
    private static final LambdaAsyncClient lambdaClient = LambdaAsyncClient.create();
    ...
    @Override
    public String handleRequest(SQSEvent event, Context context)
    {
        String response = new String();
        // call Lambda API
        logger.info("Getting account settings");
        CompletableFuture<GetAccountSettingsResponse> accountSettings =
            lambdaClient.getAccountSettings(GetAccountSettingsRequest.builder().build());
        // log execution details
        logger.info("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        ...
    }
}
```

Le référentiel GitHub de ce guide propose des exemples d'applications faciles à déployer qui illustrent divers types de gestionnaires. Pour de plus amples informations, veuillez consulter [la fin de cette rubrique](#) (p. 375).

Sections

- [Choix des types d'entrée et de sortie](#) (p. 373)
- [Interfaces du gestionnaire](#) (p. 374)
- [Exemple de code de gestionnaire](#) (p. 375)

Choix des types d'entrée et de sortie

Vous spécifiez le type d'objet auquel l'événement est mappé dans la signature de la méthode de gestion. Dans l'exemple précédent, le moteur d'exécution Java déserialise l'événement en un type qui implémente l'interface `Map<String, String>`. Les cartes chaîne-chaîne fonctionnent pour les événements plats tels que les suivants :

Example `Event.json` – Données météorologiques

```
{
    "temperatureK": 281,
    "windKmh": -3,
```

```
    "humidityPct": 0.55,  
    "pressureHPa": 1020  
}
```

Cependant, la valeur de chaque champ doit être une chaîne ou un nombre. Si l'événement inclut un champ qui a un objet comme valeur, le moteur d'exécution ne peut pas le déserialiser et renvoie une erreur.

Choisissez un type d'entrée qui fonctionne avec les données d'événement traitées par votre fonction. Vous pouvez utiliser un type de base, un type générique ou un type bien défini.

Types d'entrée

- `Integer`, `Long`, `Double`, etc. – L'événement est un nombre sans mise en forme supplémentaire—par exemple, `3.5`. Le moteur d'exécution convertit la valeur en un objet du type spécifié.
- `String` – L'événement est une chaîne JSON, y compris des guillemets—par exemple, `"My string."`. Le moteur d'exécution convertit la valeur (sans guillemets) en objet `String`.
- `Type`, `Map<String, Type>` etc. – L'événement est un objet JSON. Le moteur d'exécution le déserialise en un objet du type ou de l'interface spécifié.
- `List<Integer>, List<String>, List<Object>`, etc. – L'événement est un tableau JSON. Le moteur d'exécution le déserialise en un objet du type ou de l'interface spécifié.
- `InputStream` – L'événement a n'importe quel type JSON. Le moteur d'exécution transmet un flux d'octets du document au gestionnaire sans modification. Vous déserialisez l'entrée et écrivez la sortie dans un flux de sortie.
- Type de bibliothèque – Pour les événements envoyés par les services AWS, utilisez les types de la bibliothèque [aws-lambda-java-events \(p. 366\)](#).

Si vous définissez votre propre type d'entrée, il doit s'agir d'un ancien objet Java (POJO) déserialisable, mutable, avec un constructeur par défaut et des propriétés pour chaque champ de l'événement. Les clés de l'événement qui ne correspondent pas à une propriété, ainsi que les propriétés qui ne sont pas incluses dans l'événement sont supprimées sans erreur.

Le type de sortie peut être un objet ou `void`. Le moteur d'exécution sérialise les valeurs de retour dans le texte. Si la sortie est un objet avec des champs, le mot d'exécution le sérialise dans un document JSON. S'il s'agit d'un type qui enveloppe une valeur primitive, le moteur d'exécution renvoie une représentation textuelle de cette valeur.

Interfaces du gestionnaire

La bibliothèque `aws-lambda-java-core` définit deux interfaces pour les méthodes de gestion. Utilisez les interfaces fournies pour simplifier la configuration du gestionnaire et valider la signature de la méthode du gestionnaire au moment de la compilation.

- `com.amazonaws.services.lambda.runtime.RequestHandler`
- `com.amazonaws.services.lambda.runtime.RequestStreamHandler`

L'interface `RequestHandler` est un type générique qui prend deux paramètres : le type d'entrée et le type de sortie. Les deux types doivent être des objets. Lorsque vous utilisez cette interface, le moteur d'exécution Java déserialise l'événement dans un objet avec le type d'entrée et sérialise la sortie en texte. Utilisez cette interface lorsque la sérialisation intégrée fonctionne avec vos types d'entrée et de sortie.

Example `Handler.java` – Interface du gestionnaire

```
// Handler value: example.Handler  
public class Handler implements RequestHandler<Map<String, String>, String>{
```

```
@Override  
public String handleRequest(Map<String, String> event, Context context)
```

Pour utiliser votre propre sérialisation, implémentez l'interface `RequestStreamHandler`. Avec cette interface, Lambda transmet à votre gestionnaire un flux d'entrée et un flux de sortie. Le gestionnaire lit les octets du flux d'entrée, écrit dans le flux de sortie et renvoie une valeur vide.

L'exemple suivant utilise les types de lecteur et de rédacteur en mémoire tampon pour travailler avec les flux d'entrée et de sortie. Il utilise la bibliothèque `Gson` pour la sérialisation et la désérialisation.

Example HandlerStream.java

```
import com.amazonaws.services.lambda.runtime.Context  
import com.amazonaws.services.lambda.runtime.RequestStreamHandler  
import com.amazonaws.services.lambda.runtime.LambdaLogger  
...  
// Handler value: example.HandlerStream  
public class HandlerStream implements RequestStreamHandler {  
    Gson gson = new GsonBuilder().setPrettyPrinting().create();  
    @Override  
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context) throws IOException  
    {  
        LambdaLogger logger = context.getLogger();  
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,  
Charset.forName("US-ASCII")));  
        PrintWriter writer = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(outputStream, Charset.forName("US-ASCII"))));  
        try  
        {  
            HashMap event = gson.fromJson(reader, HashMap.class);  
            logger.log("STREAM TYPE: " + inputStream.getClass().toString());  
            logger.log("EVENT TYPE: " + event.getClass().toString());  
            writer.write(gson.toJson(event));  
            if (writer.checkError())  
            {  
                logger.log("WARNING: Writer encountered an error.");  
            }  
        }  
        catch (IllegalStateException | JsonSyntaxException exception)  
        {  
            logger.log(exception.toString());  
        }  
        finally  
        {  
            reader.close();  
            writer.close();  
        }  
    }  
}
```

Exemple de code de gestionnaire

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation de différents types de gestionnaires et interfaces. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle AWS SAM et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.

- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Les applications `blank-java` et `s3-java` prennent un événement de service AWS en entrée et retournent une chaîne. L'application `java-basic` comprend plusieurs types de gestionnaires :

- [Handler.java](#) – Prend un `Map<String, String>` comme entrée.
- [HandlerInteger.java](#) – Prend un `Integer` comme entrée.
- [HandlerList.java](#) – Prend `List<Integer>` comme entrée.
- [HandlerStream.java](#) – Prend un `InputStream` et un `OutputStream` comme entrée.
- [HandlerString.java](#) – Prend un `String` comme entrée.
- [HandlerWeatherData.java](#) – Prend un type personnalisé comme entrée.

Pour tester différents types de gestionnaire, modifiez simplement la valeur du gestionnaire dans le modèle AWS SAM. Pour obtenir des instructions détaillées, consultez le fichier `readme` de l'exemple d'application.

Objet de contexte AWS Lambda en Java

Lorsque le service Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 372\)](#). 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()` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.
- `getFunctionName()` – Renvoie le nom de la fonction Lambda.
- `getFunctionVersion()` – Renvoie la [version \(p. 70\)](#) de la fonction.
- `getInvokedFunctionArn()` – Renvoie l'Amazon Resource Name (ARN) utilisé pour appeler la fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `getMemoryLimitInMB()` – Renvoie la quantité de mémoire allouée à la fonction.
- `getAwsRequestId()` – Renvoie l'identifiant de la demande d'appel.
- `getLogGroupName()` – Renvoie le groupe de journaux pour la fonction.
- `getLogStreamName()` – Renvoie le flux de journal pour l'instance de la fonction.
- `getIdentity()` – (applications mobiles) Renvoie des informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `getClientContext()` – (applications mobiles) Renvoie le contexte client fourni à Lambda par l'application client.
- `getLogger()` – Renvoie l'[objet Logger \(p. 378\)](#) pour la fonction.

L'exemple suivant montre une fonction qui utilise l'objet de contexte pour accéder à l'enregistreur Lambda.

Example [handler.java](#)

```
package example;
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.RequestHandler
import com.amazonaws.services.lambda.runtime.LambdaLogger
...
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Map<String, String> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("200 OK");
        // log execution details
        logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        logger.log("CONTEXT: " + gson.toJson(context));
        // process event
        logger.log("EVENT: " + gson.toJson(event));
        logger.log("EVENT TYPE: " + event.getClass().toString());
        return response;
    }
}
```

La fonction sérialise l'objet de contexte dans JSON et l'enregistre dans son flux de journal.

Example sortie de journal

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
...
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
...
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed Duration:
200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

L'interface de l'objet contextuel est disponible dans la bibliothèque [aws-lambda-java-core](#). Vous pouvez implémenter cette interface pour créer une classe de contexte à des fins de test. L'exemple suivant montre une classe de contexte qui renvoie des valeurs factices pour la plupart des propriétés et un enregistreur de test opérationnel.

Example [src/test/java/example/TestContext.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.ClientContext;
import com.amazonaws.services.lambda.runtime.LambdaLogger

public class TestContext implements Context{
    public TestContext() {}
    public String getAwsRequestId(){
```

```
        return new String("495b12a8-xmpl-4eca-8168-160484189f99");
    }
    public String getLogGroupName(){
        return new String("/aws/lambda/my-function");
    }
    ...
    public LambdaLogger getLogger(){
        return new TestLogger();
    }
}
```

Pour de plus amples informations sur la journalisation, veuillez consulter [Journalisation des fonctions AWS Lambda dans Java \(p. 378\)](#).

Contexte dans des exemples d'applications

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation de l'objet contextuel. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle Modèle d'application sans serveur AWS (AWS SAM) et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Tous les exemples d'applications ont une classe de contexte de test pour les tests unitaires. L'application `java-basic` montre l'utilisation de l'objet contexte pour obtenir un enregistreur. Il utilise SLF4J et Log4J 2 pour fournir un enregistreur qui fonctionne pour les tests unitaires locaux.

Journalisation des fonctions AWS Lambda dans Java

Votre fonction Lambda est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur `java.lang.System` ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. La bibliothèque [aws-lambda-java-core \(p. 366\)](#) fournit une classe d'enregistreur nommée `LambdaLogger` à laquelle vous pouvez accéder à partir de l'objet de contexte. La classe d'enregistreur prend en charge les journaux multilignes.

L'exemple suivant utilise l'enregistreur LambdaLogger fourni par l'objet de contexte.

Example handler.java

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Object, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Object event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("SUCCESS");
        // log execution details
        logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        logger.log("CONTEXT: " + gson.toJson(context));
        // process event
        logger.log("EVENT: " + gson.toJson(event));
        return response;
    }
}
```

Example format des journaux

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
EVENT:
{
    "records": [
        {
            "messageId": "19dd0b57-xmpl-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            ...
        }
    ]
}
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed Duration:
200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

L'environnement d'exécution Java enregistre les lignes START, END et REPORT pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.

- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY Traceld – Pour les demandes suivies, [l'ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 380\)](#)
- [Utilisation de l'AWS CLI \(p. 380\)](#)
- [Suppression de journaux \(p. 382\)](#)
- [Journalisation avancée avec Log4j 2 et SLF4J \(p. 382\)](#)
- [Exemple de code de journalisation \(p. 384\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
```

```
    "LogResult":  
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d  
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST  
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-  
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",  
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed  
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash  
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out  
sed -i'' -e 's://"//g' out  
sleep 15  
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat  
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh  
{  
    "StatusCode": 200,  
    "ExecutedVersion": "$LATEST"  
}  
{  
    "events": [  
        {  
            "timestamp": 1559763003171,  
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:  
$LATEST\\n",  
            "ingestionTime": 1559763003309  
        },  
        {  
            "timestamp": 1559763003173,  
            "message": "2019-06-05T19:30:03.173Z\\t4ce9340a-b765-490f-ad8a-02ab3415e2bf  
\\tINFO\\tENVIRONMENT VARIABLES\\r\\r \\\"AWS_LAMBDA_FUNCTION_VERSION\\\": \\\"$LATEST\\\",\\r ...",  
            "ingestionTime": 1559763018353  
        },  
        {  
            "timestamp": 1559763003173,
```

```
"message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tINFO\tEVENT\r{\r\t\t\"key\": \"value\"\r}\n",
"ingestionTime": 1559763018353
},
{
  "timestamp": 1559763003218,
  "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
  "ingestionTime": 1559763018353
},
{
  "timestamp": 1559763003218,
  "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
  "ingestionTime": 1559763018353
}
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Journalisation avancée avec Log4j 2 et SLF4J

Pour personnaliser la sortie du journal, prendre en charge la journalisation pendant les tests unitaires et enregistrer les appels SDK AWS, utilisez Apache Log4j 2 avec SLF4J. Log4j est une bibliothèque de journalisation pour les programmes Java qui vous permet de configurer les niveaux de journalisation et d'utiliser les bibliothèques appender. SLF4J est une bibliothèque de façade qui vous permet de changer la bibliothèque que vous utilisez sans modifier votre code de fonction.

Pour ajouter l'ID de requête aux journaux de votre fonction, utilisez l'appender dans la bibliothèque [aws-lambda-java-log4j2 \(p. 366\)](#). L'exemple suivant montre un fichier de configuration Log4j 2 qui ajoute un horodatage et un ID de requête à tous les journaux.

Example [src/main/resources/log4j2.xml](#) – Configuration de l'appender

```
<Configuration status="WARN">
<Appenders>
  <Lambda name="Lambda">
    <PatternLayout>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n</pattern>
    </PatternLayout>
  </Lambda>
</Appenders>
<Loggers>
  <Root level="INFO">
    <AppenderRef ref="Lambda"/>
  </Root>
  <Logger name="software.amazon.awssdk" level="WARN" />
  <Logger name="software.amazon.awssdk.request" level="DEBUG" />
</Loggers>
</Configuration>
```

Avec cette configuration, chaque ligne est précédée de la date, de l'heure, de l'ID de demande, du niveau de journal et du nom de classe.

Example format du journal avec appender

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
2020-03-18 08:52:43 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 INFO Handler - ENVIRONMENT
VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
2020-03-18 08:52:43 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 INFO Handler - CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
```

SLF4J est une bibliothèque de façade pour la journalisation en code Java. Dans votre code de fonction, vous utilisez la fabrique d'enregistreurs SLF4J pour récupérer un enregistreur avec des méthodes pour les niveaux de journalisation comme `info()` et `warn()`. Dans votre configuration de build, vous incluez la bibliothèque de journalisation et l'adaptateur SLF4J dans le chemin de classe. En changeant les bibliothèques dans la configuration de build, vous pouvez changer le type d'enregistreur sans changer votre code de fonction. SLF4J est nécessaire pour capturer les journaux de la SDK pour Java.

Dans l'exemple suivant, la classe de gestionnaire utilise SLF4J pour récupérer un enregistreur.

Example [src/main/java/example/Handler.java](#) – Journalisation avec SLF4J

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

// Handler value: example.Handler
public class Handler implements RequestHandler<SQSEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    LambdaAsyncClient lambdaClient = LambdaAsyncClient.create();
    @Override
    public String handleRequest(SQSEvent event, Context context)
    {
        String response = new String();
        // call Lambda API
        logger.info("Getting account settings");
        CompletableFuture<GetAccountSettingsResponse> accountSettings =
            lambdaClient.getAccountSettings(GetAccountSettingsRequest.builder().build());
        // log execution details
        logger.info("ENVIRONMENT VARIABLES: {}", gson.toJson(System.getenv()));
        ...
    }
}
```

La configuration de build prend des dépendances d'exécution sur l'appender Lambda et l'adaptateur SLF4J, et des dépendances d'implémentation sur Log4J 2.

Example [build.gradle](#) – Dépendances de journalisation

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.10.73')
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.4.0')
    implementation 'software.amazon.awssdk:lambda'
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-core'
```

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2'  
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'  
implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'  
implementation 'com.amazonaws:aws-lambda-java-events:2.2.9'  
implementation 'com.google.code.gson:gson:2.8.6'  
implementation 'org.apache.logging.log4j:log4j-api:2.13.0'  
implementation 'org.apache.logging.log4j:log4j-core:2.13.0'  
runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.13.0'  
runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'  
testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'  
testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'  
}
```

Lorsque vous exécutez votre code localement pour des tests, l'objet contextuel avec l'enregistreur Lambda n'est pas disponible et il n'y a pas d'ID de requête pour l'appender Lambda à utiliser. Pour des exemples de configurations de test, consultez les exemples d'applications dans la section suivante.

Exemple de code de journalisation

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation de diverses configurations de journalisation. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle AWS SAM et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

L'exemple [java-basic](#) d'application présente une configuration de journalisation minimale qui prend en charge les tests de journalisation. Le code du gestionnaire utilise l'enregistreur `LambdaLogger` fourni par l'objet de contexte. Pour les tests, l'application utilise une classe `TestLogger` personnalisée qui implémente l'interface `LambdaLogger` avec un logger Log4j 2. Il utilise SLF4J comme façade pour la compatibilité avec le kit SDK AWS. Les bibliothèques de journalisation sont exclues de la sortie de build pour limiter la taille du package de déploiement.

L'exemple d'application [blank-java](#) s'appuie sur la configuration de base avec la journalisation du kit AWS SDK et l'appender Lambda Log4j 2. Il utilise Log4j 2 dans Lambda avec un appender personnalisé qui ajoute l'ID de requête d'appel à chaque ligne.

Erreurs de fonction AWS Lambda dans Java

Lorsque votre fonction déclenche une erreur, Lambda renvoie des détails sur l'erreur à l'appelant. Le corps de la réponse renvoyée par Lambda contient un document JSON avec le nom de l'erreur, le type d'erreur et un tableau de cadres de pile. Le client ou le service qui a appelé la fonction peut gérer l'erreur ou la

transmettre à un utilisateur final. Vous pouvez utiliser des exceptions personnalisées pour renvoyer des informations utiles aux utilisateurs en cas d'erreurs client.

Example [src/main/java/example/HandlerDivide.java](#) – Exception d'exécution

```
import java.util.List;

// Handler value: example.HandlerDivide
public class HandlerDivide implements RequestHandler<List<Integer>, Integer>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public Integer handleRequest(List<Integer> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        // process event
        if ( event.size() != 2 )
        {
            throw new InputLengthException("Input must be an array that contains 2 numbers.");
        }
        int numerator = event.get(0);
        int denominator = event.get(1);
        logger.log("EVENT: " + gson.toJson(event));
        logger.log("EVENT TYPE: " + event.getClass().toString());
        return numerator/denominator;
    }
}
```

Lorsque la fonction lance `InputLengthException`, le moteur d'exécution Java le sérialise dans le document suivant.

Example document d'erreur (espace ajouté)

```
{
    "errorMessage": "Input must contain 2 numbers.",
    "errorType": "java.lang.InputLengthException",
    "stackTrace": [
        "example.HandlerDividehandleRequest(HandlerDivide.java:23)",
        "example.HandlerDividehandleRequest(HandlerDivide.java:14)"
    ]
}
```

Dans cet exemple, `InputLengthException` est un `RuntimeException`. L'[interface \(p. 374\)](#) `RequestHandler` n'autorise pas les exceptions vérifiées. L'interface `RequestStreamHandler` prend en charge le lancement d'erreurs `IOException`.

L'instruction `return` dans l'exemple ci-dessus peut également lancer une exception d'exécution.

```
    return numerator/denominator;
```

Ce code peut renvoyer une erreur arithmétique.

```
{"errorMessage": "/ by zero", "errorType": "java.lang.ArithmaticException", "stackTrace": [
    "example.HandlerDividehandleRequest(HandlerDivide.java:28)", "example.HandlerDivide.handleRequest(Hand
```

Sections

- [Affichage de la sortie d'erreur \(p. 386\)](#)
- [Présentation des types et sources d'erreurs \(p. 387\)](#)

- [Gestion des erreurs dans les clients \(p. 388\)](#)
- [Gestion des erreurs dans d'autres services AWS \(p. 389\)](#)
- [Gestion des erreurs dans les exemples d'applications \(p. 389\)](#)

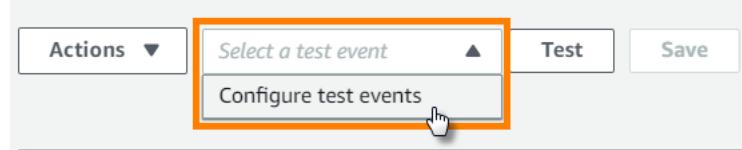
Affichage de la sortie d'erreur

Vous pouvez appeler votre fonction avec une charge utile de test et afficher la sortie d'erreur dans la console Lambda, à partir de la ligne de commande ou avec le kit AWS SDK. La sortie d'erreur est également capturée dans les journaux d'exécution de la fonction et, lorsque le [suivi \(p. 390\)](#) est activé, dans AWS X-Ray.

Pour afficher la sortie d'erreur dans la console Lambda,appelez-la avec un événement de test.

Pour appeler une fonction dans la console Lambda

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configurer les événements de test dans le menu déroulant en regard du bouton Tester.



4. Choisissez un modèle d'événement qui correspond aux événements traités par votre fonction.
5. Entrez un nom pour l'événement de test et modifiez la structure de l'événement si nécessaire.
6. Sélectionnez Créer.
7. Sélectionnez Test.

La console Lambda appelle votre fonction de [manière synchrone \(p. 92\)](#) et affiche le résultat. Pour afficher la réponse, les journaux et d'autres informations, développez la section Détails.

Lorsque vous appelez la fonction à partir de la ligne de commande, l'AWS CLI divise la réponse en deux documents. Pour indiquer une erreur de fonction, la réponse affichée dans le terminal comprend un champ `FunctionError`. La réponse ou l'erreur renvoyée par la fonction est écrite sur le fichier de sortie.

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Consultez le fichier de sortie pour accéder au document d'erreur.

```
$ cat out.json
{"errorMessage":"Input must contain 2
numbers.", "errorType":"java.lang.InputLengthException", "stackTrace":
["example.HandlerDivide.handleRequest(HandlerDivide.java:23)", "example.HandlerDivide.handleRequest(Han
```

Lambda enregistre également 256 Ko de l'objet d'erreur dans les journaux de la fonction. Pour afficher les journaux lorsque vous appelez la fonction à partir de la ligne de commande, utilisez l'option `--log-type` et décodez la chaîne en base64 de la réponse.

```
$ aws lambda invoke --function-name my-function --payload "[100,0]" out.json --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f Version: $LATEST
EVENT: [
    100,
    0
]EVENT TYPE: class java.util.ArrayList/ by zero: java.lang.ArithmeticException
java.lang.ArithmeticException: / by zero
    at example.HandlerDivide.handleRequest(HandlerDivide.java:28)
    at example.HandlerDivide.handleRequest(HandlerDivide.java:13)

END RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f
REPORT RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f Duration: 4.20 ms          Billed
Duration: 100 ms Memory Size: 512 MB      Max Memory Used: 95 MB
XRAY TraceId: 1-5e73162b-1919xmpl2592f4549e1c39be SegmentId: 3dadxmpl48126cb8
Sampled: true
```

Pour de plus amples informations sur les journaux, veuillez consulter [Journalisation des fonctions AWS Lambda dans Java \(p. 378\)](#).

Présentation des types et sources d'erreurs

Lorsque vous appelez une fonction, plusieurs sous-systèmes gèrent la demande, l'événement, la sortie et la réponse. Les erreurs peuvent provenir de votre service Lambda (erreurs d'appel) ou d'une instance de votre fonction. Les erreurs de fonction incluent les exceptions renvoyées par votre code de gestionnaire et les exceptions renvoyées par le moteur d'exécution Lambda.

Le service Lambda reçoit la demande d'appel et la valide. Il contrôle les autorisations, vérifie que le document d'événement est un JSON valide et contrôle les valeurs des paramètres. Si le service Lambda rencontre une erreur, il renvoie un type d'exception, un message et un code d'état HTTP qui indiquent la cause de l'erreur.

Note

Pour obtenir la liste complète des erreurs que l'opération d'API `Invoke` est susceptible de renvoyer, veuillez consulter [Appeler les erreurs \(p. 567\)](#) dans la référence de l'API Lambda.

Une erreur de type 4xx du service Lambda indique une erreur que l'appelant peut corriger en modifiant la demande, en demandant l'autorisation ou en réessayant. Une erreur de type 5xx indique un problème avec le service Lambda ou un problème avec la configuration ou les ressources de la fonction. Ces problèmes ne peuvent pas être résolus par l'appelant, mais le propriétaire de la fonction peut être en mesure de les résoudre.

Si une demande réussit la validation, Lambda l'envoie à une instance de la fonction. Le moteur d'exécution convertit le document d'événement en objet et le transmet à votre code de gestionnaire. Des erreurs peuvent survenir au cours de ce processus si, par exemple, le nom de votre méthode de gestionnaire ne correspond pas à la configuration de la fonction, ou si l'appel expire avant que votre code de gestionnaire renvoie une réponse. Les erreurs d'exécution Lambda sont formatées comme des erreurs que votre code renvoie, mais elles sont renvoyées par le moteur d'exécution.

Dans l'exemple suivant, le moteur d'exécution ne parvient pas à déserialiser l'événement dans un objet. L'entrée est un type JSON valide, mais elle ne correspond pas au type attendu par la méthode du gestionnaire.

Example Erreur d'exécution Lambda

```
{
    "errorMessage": "An error occurred during JSON parsing",
    "errorType": "java.lang.RuntimeException",
```

```

    "stackTrace": [],
    "cause": {
        "errorMessage": "com.fasterxml.jackson.databind.exc.InvalidFormatException: Can not
construct instance of java.lang.Integer from String value '1000,10': not a valid Integer
value\n at [Source: lambdainternal.util.NativeMemoryAsInputStream@35fc6dc4; line: 1,
column: 1] (through reference chain: java.lang.Object[0])",
        "errorType": "java.io.UncheckedIOException",
        "stackTrace": [],
        "cause": {
            "errorMessage": "Can not construct instance of java.lang.Integer
from String value '1000,10': not a valid Integer value\n at [Source:
lambdainternal.util.NativeMemoryAsInputStream@35fc6dc4; line: 1, column: 1] (through
reference chain: java.lang.Object[0])",
            "errorType": "com.fasterxml.jackson.databind.exc.InvalidFormatException",
            "stackTrace": [
                "com.fasterxml.jackson.databind.exc.InvalidFormatException.from(InvalidFormatException.java:55)",
                "com.fasterxml.jackson.databind.DeserializationContext.weirdStringException(DeserializationContext.jav
                ...
            ]
        }
    }
}

```

Pour les erreurs d'exécution Lambda et d'autres erreurs de fonction, le service Lambda ne renvoie pas de code d'erreur. Un code d'état de série 2xx indique que le service Lambda a accepté la demande. Au lieu d'un code d'erreur, Lambda indique l'erreur en incluant l'en-tête `X-Amz-Function-Error` dans la réponse.

Pour l'appel asynchrone, les événements sont mis en file d'attente avant d'être envoyés par Lambda à votre fonction. Pour les demandes valides, Lambda renvoie immédiatement une réponse de réussite et ajoute l'événement à la file d'attente. Lambda lit ensuite les événements de la file d'attente et appelle la fonction. Si une erreur se produit, Lambda réessaie avec un comportement qui varie en fonction du type d'erreur. Pour de plus amples informations, veuillez consulterz [Appel asynchrone \(p. 93\)](#).

Gestion des erreurs dans les clients

Les clients qui appellent des fonctions Lambda peuvent choisir de gérer les erreurs ou de les transmettre à l'utilisateur final. Le comportement correct de gestion des erreurs dépend du type d'application, de l'audience et de la source de l'erreur. Par exemple, si un appel échoue avec un code d'erreur 429 (trop de demandes), l'interface de ligne de commande AWS réessaie jusqu'à 4 fois avant d'afficher une erreur pour l'utilisateur.

```
$ aws lambda invoke --function-name my-function out.json
An error occurred (TooManyRequestsException) when calling the Invoke operation (reached max
retries: 4): Rate Exceeded.
```

Pour les autres erreurs d'appel, le comportement correct dépend du code de réponse. Les erreurs de type 5xx peuvent indiquer une condition temporaire susceptible d'être résolue sans aucune action de la part de l'utilisateur. Une nouvelle tentative peut réussir ou non. Les erreurs de type 4xx autres que 429 indiquent généralement une erreur de demande. Une nouvelle tentative n'est pas susceptible de réussir.

Pour les erreurs de fonction, le client peut traiter le document d'erreur et afficher le message d'erreur dans un format convivial. Une application basée sur un navigateur peut afficher le message et le type d'erreur, mais omettre la trace de la pile. L'interface de ligne de commande AWS enregistre l'objet d'erreur dans le fichier de sortie et affiche un document généré à partir des en-têtes de réponse.

```
$ aws lambda invoke --function-name my-function --payload '[1000]' out.json
```

```
{  
    "StatusCode": 200,  
    "FunctionError": "Unhandled",  
    "ExecutedVersion": "$LATEST"  
}
```

Example out.json

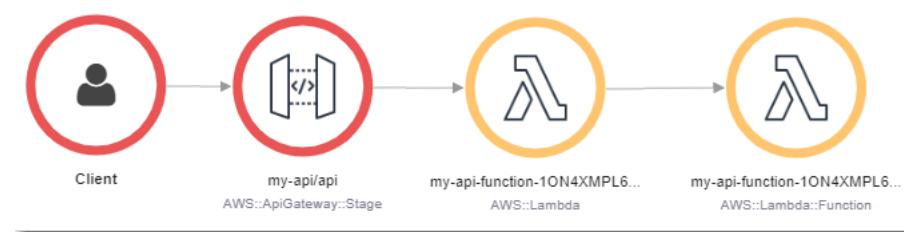
```
{"errorMessage":"Input must be an array that contains 2  
numbers.", "errorType":"example.InputLengthException", "stackTrace":  
[ "example.HandlerDivide.handleRequest(HandlerDivide.java:22)", "example.HandlerDivide.handleRequest(Hand
```

Gestion des erreurs dans d'autres services AWS

Lorsqu'un service AWS appelle votre fonction, le service choisit le type d'appel et le comportement pour la nouvelle tentative. Les services AWS peuvent appeler votre fonction selon un calendrier, en réponse à un événement de cycle de vie sur une ressource, ou pour répondre à une demande d'un utilisateur. Certains services appellent des fonctions de manière asynchrone et permettent à Lambda de gérer les erreurs, tandis que d'autres réessaient ou transmettent les erreurs à l'utilisateur.

Par exemple, API Gateway traite toutes les erreurs d'appel et de fonction comme des erreurs internes. Si l'API Lambda rejette la demande d'appel, API Gateway renvoie un code d'erreur 500. Si la fonction s'exécute mais renvoie une erreur, ou renvoie une réponse dans le mauvais format, API Gateway renvoie un code d'erreur 502. Pour personnaliser la réponse d'erreur, vous devez attraper les erreurs dans votre code et formater une réponse dans le format requis.

Pour déterminer la source d'une erreur et sa cause, utilisez AWS X-Ray. Avec X-Ray, vous pouvez savoir quel composant a rencontré une erreur et voir les détails sur les exceptions. L'exemple suivant montre une erreur de fonction qui a entraîné une réponse 502 de la part de API Gateway.



Commencez par X-Ray en [activant le suivi actif \(p. 390\)](#) de vos fonctions.

Pour de plus amples informations sur la façon dont les autres services gèrent les erreurs, veuillez consulter les rubriques du chapitre [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Gestion des erreurs dans les exemples d'applications

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation des erreurs. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle Modèle d'application sans serveur AWS (AWS SAM) et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.

- **java-events** – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- **java-events-v1sdk** – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- **s3-java** – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

La fonction `java-basic` inclut un gestionnaire (`HandlerDivide`) qui renvoie une exception d'exécution personnalisée. Le gestionnaire `HandlerStream` implémente `RequestStreamHandler` et peut lancer une exception vérifiée `IOException`.

Instrumentation du code Java dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle qu'elle parcourt les ressources de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.

3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

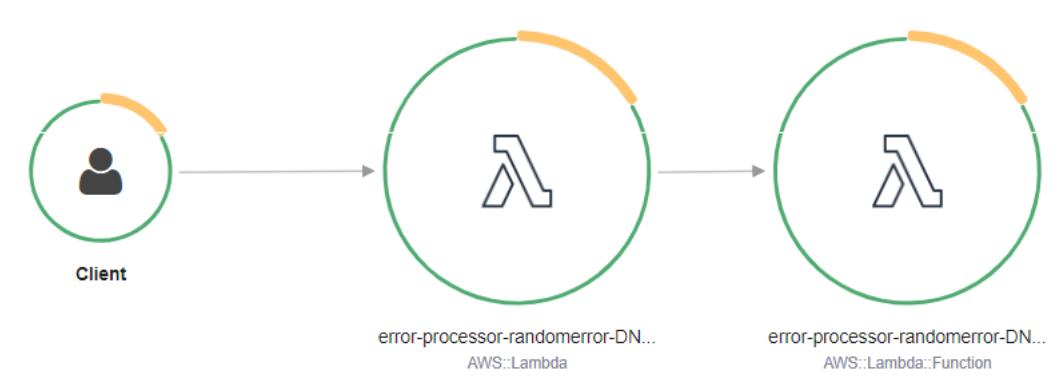
Tarification

X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au [rôle d'exécution \(p. 33\)](#) de votre fonction. Sinon, ajoutez la stratégie [AWSXRayDaemonWriteAccess](#) au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'environnement d'exécution (p. 18) de la fonction.

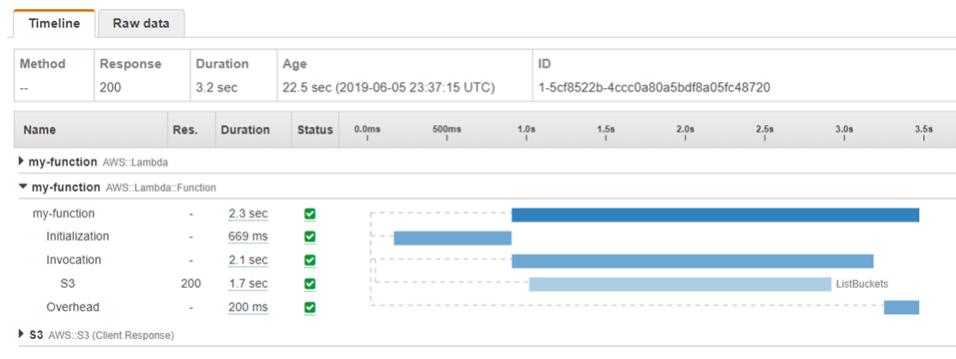


Pour enregistrer des détails sur les appels effectués par votre fonction à d'autres ressources et services, ajoutez le X-Ray SDK pour Java à votre configuration de build. L'exemple suivant montre une configuration de build Gradle qui inclut les bibliothèques permettant l'instrumentation automatique des clients AWS SDK pour Java 2.x.

Example [build.gradle](#) – Dépendances du suivi

```
dependencies {  
    implementation platform('software.amazon.awssdk:bom:2.10.73')  
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.4.0')  
    implementation 'software.amazon.awssdk:lambda'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-core'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'  
    ...  
}
```

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type `AWS::Lambda` et l'autre de type `AWS::Lambda::Function`. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour de plus amples informations, veuillez consulter [AWS X-Ray SDK pour Java](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 392\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 393\)](#)
- [Stockage des dépendances d'exécution dans une couche \(p. 393\)](#)
- [Suivi dans des exemples d'applications \(p. 394\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
    ...
```

Stockage des dépendances d'exécution dans une couche

Si vous utilisez le kit X-Ray SDK pour instrumentiser les clients du kit AWS SDK pour votre fonction, votre package de déploiement peut devenir assez volumineux. Pour éviter de charger des dépendances d'exécution chaque fois que vous mettez à jour votre code de fonction, empaquetez-les dans une [couche Lambda \(p. 76\)](#).

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le SDK pour Java et le kit X-Ray SDK pour Java.

Example [template.yml](#) – Couche de dépendances

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: build/distributions/blank-java.zip  
      Tracing: Active  
      Layers:  
        - !Ref libs  
      ...  
  libs:  
    Type: AWS::Serverless::LayerVersion  
    Properties:  
      LayerName: blank-java-lib  
      Description: Dependencies for the blank-java sample app.  
      ContentUri: build/blank-java-lib.zip  
      CompatibleRuntimes:
```

- java8

Avec cette configuration, vous ne mettez à jour les fichiers JAR de bibliothèque que si vous modifiez vos dépendances d'exécution. Le package de déploiement de fonction contient uniquement votre code. Lorsque vous mettez à jour votre code de fonction, le temps de téléchargement est beaucoup plus rapide que si vous incluez des dépendances dans le package de déploiement.

La création d'une couche pour les dépendances nécessite des modifications de configuration de génération pour générer l'archive des couches avant le déploiement. Consultez l'exemple d'application [java-basic](#) pour accéder à un exemple fonctionnel.

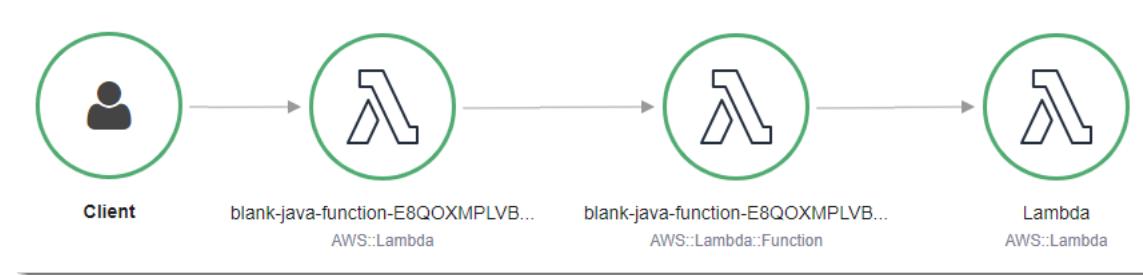
Suivi dans des exemples d'applications

Le référentiel GitHub de ce guide comprend des exemples d'applications qui démontrent l'utilisation du suivi. Chaque exemple d'application inclut des scripts facilitant le déploiement et le nettoyage, un modèle AWS SAM et des ressources de support.

Exemples d'applications Lambda en Java

- [blank-java](#) – Fonction Java qui montre l'utilisation des bibliothèques Java de Lambda, la journalisation, les variables d'environnement, les couches, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [java-basic](#) – Fonction Java minimale avec des tests unitaires et une configuration de journalisation variable.
- [java-events](#) – Fonction Java minimale qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements qui n'ont pas besoin du kit SDK AWS en tant que dépendance, par exemple Amazon API Gateway.
- [java-events-v1sdk](#) – Fonction Java qui utilise la bibliothèque [aws-lambda-java-events \(p. 366\)](#) avec des types d'événements nécessitant le kit SDK AWS en tant que dépendance (Amazon Simple Storage Service, Amazon DynamoDB et Amazon Kinesis).
- [s3-java](#) – Fonction Java qui traite les événements de notification de Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.

Tous les exemples d'applications ont un suivi actif activé pour les fonctions Lambda. L'application [blank-java](#) montre l'instrumentation automatique des clients AWS SDK pour Java 2.x, la gestion des segments pour les tests, les sous-segments personnalisés et l'utilisation de couches Lambda pour stocker les dépendances d'exécution.



Cet exemple issu de l'exemple d'application [blank-java](#) montre les nœuds du service Lambda, d'une fonction et de l'API Lambda. La fonction appelle l'API Lambda pour surveiller l'utilisation du stockage dans Lambda.

Création d'un package de déploiement à l'aide d'Eclipse

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.

Note

La boîte à outils Eclipse du kit AWS SDK fournit un plug-in Eclipse pour vous permettre de créer un package de déploiement et de le charger 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 obtenir un exemple d'utilisation de la boîte à outils pour la création des fonctions Lambda, consultez [Utilisation d'AWS Lambda avec AWS Toolkit pour Eclipse](#).

Rubriques

- [Prérequis \(p. 395\)](#)
- [Créer et développer un projet \(p. 395\)](#)

Prérequis

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.

Créer et développer 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 : 1.2.1

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-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 plus d'informations, consultez [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 : 3.2.2
- c. Générez le projet à nouveau.

Cette fois-ci, nous créerons 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 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](#).

Environnements d'exécution Go

| Nom | Identifiant | Système d'exploitation |
|--------|-------------|------------------------|
| Go 1.x | go1.x | Amazon Linux |

AWS Lambda fournit les bibliothèques suivantes pour Go :

- [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 \(p. 399\)](#).
- [github.com/aws/aws-lambda-go/lambdacontext](#) : assistants permettant d'accéder aux informations du contexte d'exécution depuis l'[objet de contexte \(p. 403\)](#).
- [github.com/aws/aws-lambda-go/events](#) : cette bibliothèque fournit les définitions de type pour les intégrations de sources d'événements communes.

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda en Go

- [blank-go](#) – Fonction Go qui montre l'utilisation des bibliothèques Go de Lambda, la journalisation, les variables d'environnement et le kit SDK AWS.

Rubriques

- [Package de déploiement AWS Lambda dans Go \(p. 398\)](#)
- [Gestionnaire de fonctions AWS Lambda dans Go \(p. 399\)](#)
- [Objet de contexte AWS Lambda dans Go \(p. 403\)](#)
- [Journalisation des fonctions AWS Lambda dans Go \(p. 404\)](#)
- [Erreurs de fonction AWS Lambda dans Go \(p. 408\)](#)
- [Instrumentation du code Go dans AWS Lambda \(p. 408\)](#)
- [Utilisation des variables d'environnement \(p. 412\)](#)

Package de déploiement AWS Lambda dans Go

Pour créer une fonction Lambda, vous devez d'abord concevoir un package de déploiement de la fonction Lambda, 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.

Téléchargez la bibliothèque Lambda pour Go avec go get, et compilez votre exécutable.

```
~/my-function$ go get github.com/aws/aws-lambda-go/lambda
~/my-function$ GOOS=linux go build main.go
```

Le fait de définir GOOS sur linux garantit que l'exécutable compilé est compatible avec le [runtime Go](#) (p. 122), même si vous le编译 dans un environnement Linux.

Créez un package de déploiement en empaquetant l'exécutable dans un fichier ZIP et utilisez le AWS CLI pour créer une fonction. Le paramètre du gestionnaire doit correspondre au nom de l'exécutable contenant votre gestionnaire.

```
~/my-function$ zip function.zip main
~/my-function$ aws lambda create-function --function-name my-function --runtime go1.x \
--zip-file file:///function.zip --handler main \
--role arn:aws:iam::123456789012:role/execution_role
```

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 -output main.zip main
```

Dans Powershell, exécutez la commande suivante :

```
$env:GOOS = "linux"
$env:CGO_ENABLED = "0"
$env:GOARCH = "amd64"
go build -o main main.go
~\Go\Bin\build-lambda-zip.exe -output 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 \(p. 403\)](#).
 - `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 via [Objet de contexte AWS Lambda dans Go \(p. 403\)](#).
 - `name MyEvent` : un type d'entrée avec un nom de variable `name` dont la valeur est renvoyée dans l'instruction `return`.
 - chaîne, erreur : Renvoie deux valeurs : chaîne de succès et informations d'`erreur` standard. Pour plus d'informations sur la gestion des erreurs personnalisées, consultez [Erreurs de fonction AWS Lambda dans Go \(p. 408\)](#).
 - `return fmt.Sprintf("Hello %s!", name), nil` : renvoie simplement une salutation formatée « Hello » avec le nom que vous avez fourni dans l'événement d'entrée. `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. 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 fonctions Lambda à l'aide de 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 être exportées, les noms de champs de la structure d'événement doivent être en majuscules. 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 \(p. 408\)](#).

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.

```
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.Println(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 \(p. 399\)](#). 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` – Nom de la fonction Lambda.
- `FunctionVersion` – [Version \(p. 70\)](#) de la fonction.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `LogGroupName` – Groupe de journaux de la fonction.
- `LogStreamName` – Flux de journal pour l'instance de la fonction.

Méthodes de contexte

- `Deadline` – Renvoie la date d'expiration de l'exécution, exprimée en millisecondes au format horaire Unix.

Propriétés du contexte

- `InvokedFunctionArn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `AwsRequestId` – Identifiant de la demande d'appel.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.

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.CognitoIdentityPoolID)
}

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.CognitoIdentityPoolID)` affiche ces informations, dans ce cas, le `CognitoPoolID`.

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 est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur [le package fmt](#) ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant utilise [le package de journal](#).

Example `main.go` – journalisation

```
func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
    // event
    eventJson, _ := json.MarshalIndent(event, "", "    ")
    log.Printf("EVENT: %s", eventJson)
    // environment variables
    log.Printf("REGION: %s", os.Getenv("AWS_REGION"))
    log.Println("ALL ENV VARS:")
    for _, element := range os.Environ() {
        log.Println(element)
    }
}
```

Example format des journaux

```
START RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Version: $LATEST
2020/03/27 03:40:05 EVENT: {
    "Records": [
        {
            "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            "md5OfBody": "7b27xmplb47ff90a553787216d55d91d",
            "md5OfMessageAttributes": "",
            "attributes": {
                "ApproximateFirstReceiveTimestamp": "1523232000001",
                "ApproximateReceiveCount": "1",
                "SenderId": "123456789012",
                "SentTimestamp": "1523232000000"
            },
            ...
        }
    ]
}
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMP6XBC
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$LATEST
2020/03/27 03:40:05 AWS_EXECUTION_ENV=AWS_Lambda_go1.x
END RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71
REPORT RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Duration: 38.66 ms Billed Duration:
100 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled: true
```

L'environnement d'exécution Go enregistre les lignes START, END et REPORT pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.

- XRAY Traceld – Pour les demandes suivies, l'[ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 406\)](#)
- [Utilisation de l'AWS CLI \(p. 406\)](#)
- [Suppression de journaux \(p. 408\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instruisez votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
```

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
  "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    }
  ]
}
```

```
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration:
26.73 ms\tBilled Duration: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

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 les informations suivantes :

```
{
    "errorMessage": "something went wrong!",
    "errorType": "errorString"
}
```

Instrumentation du code Go dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle parcourt les ressources de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

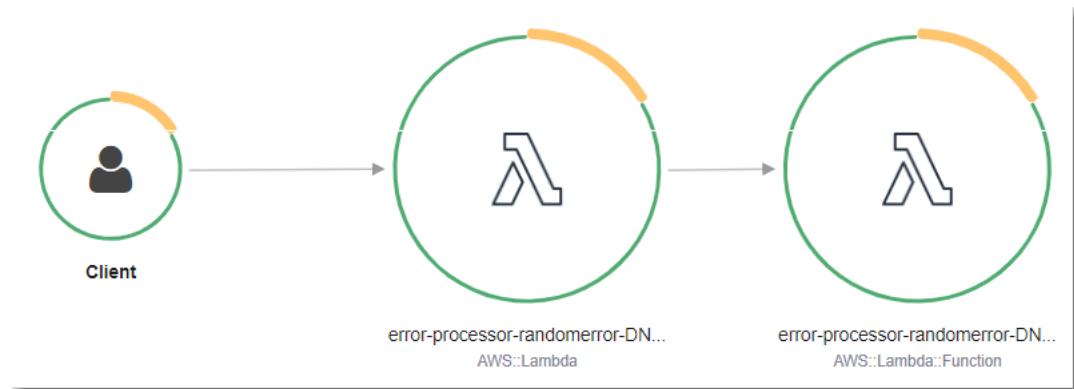
Tarification

X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au [rôle d'exécution \(p. 33\)](#) de votre fonction. Sinon, ajoutez la stratégie `AWSXRayDaemonWriteAccess` au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'[environnement d'exécution \(p. 18\)](#) de la fonction.



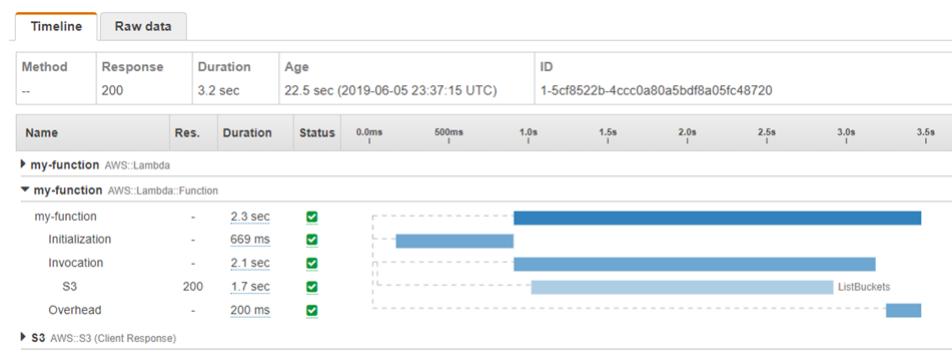
Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer des détails sur les appels que le gestionnaire effectue vers d'autres ressources et services, utilisez le kit SDK X-Ray pour Go. Téléchargez le kit de développement logiciel (SDK) à partir de son [référentiel GitHub](#) à l'aide de l'instruction go get :

```
$ go get github.com/aws/aws-xray-sdk-go
```

Pour instrumenter les clients AWS SDK, transmettez le client vers la méthode `xray.AWS()`.

```
xray.AWS(s3.Client)
```

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type `AWS::Lambda` et l'autre de type `AWS::Function`. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez [Kit SDK X-Ray pour Go](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 411\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 411\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...

```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...

```

Utilisation des variables d'environnement

Pour accéder aux [variables d'environnement \(p. 55\)](#) 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 %. years old\n", os.Getenv("NAME"), os.Getenv("AGE"))
}
```

Pour obtenir la liste des variables d'environnement définies par le moteur d'exécution Lambda, consultez [Variables d'environnement d'exécution \(p. 57\)](#).

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#.

AWS Lambda fournit les bibliothèques suivantes pour les fonctions C# :

- 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 en C# \(p. 424\)](#)) 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](#).

Environnements d'exécution .NET

| Nom | Identifiant | Système d'exploitation | |
|---------------|----------------------------|------------------------|--|
| .NET Core 3.1 | <code>dotnetcore3.1</code> | Amazon Linux 2 | |
| .NET Core 2.1 | <code>dotnetcore2.1</code> | Amazon Linux | |

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda en C#

- [blank-csharp](#) – Fonction C # qui montre l'utilisation des bibliothèques .NET de Lambda, la journalisation, les variables d'environnement, le suivi AWS X-Ray, les tests unitaires et le kit SDK AWS.
- [ec2-spot](#) – Fonction qui gère les demandes d'instance Spot dans Amazon EC2.

Rubriques

- [Package de déploiement AWS Lambda dans C# \(p. 414\)](#)
- [Gestionnaire de fonctions AWS Lambda en C# \(p. 419\)](#)
- [Objet de contexte AWS Lambda en C# \(p. 424\)](#)
- [Journalisation des fonctions AWS Lambda en C# \(p. 424\)](#)
- [Erreurs de fonction AWS Lambda en C# \(p. 428\)](#)
- [Instrumentation du code C # dans AWS Lambda \(p. 430\)](#)

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.

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 \(p. 417\)](#) ou [Interface de ligne de commande .NET Core \(p. 414\)](#). 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.

Rubriques

- [Interface de ligne de commande .NET Core \(p. 414\)](#)
- [AWS Toolkit for Visual Studio \(p. 417\)](#)

Interface de ligne de commande .NET Core

L'interface de ligne de commande .NET Core 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. Cela est utile si vous souhaitez créer un projet hors 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 exécutez la commande suivante :

| 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 | | | |

...

Exemples:

```
dotnet new mvc --auth Individual
dotnet new viewstart
dotnet new --help
```

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 examiner les détails relatifs à un modèle, utilisez l'option d'aide.

```
dotnet new lambda.EmptyFunction --help
```

Le modèle `lambda.EmptyFunction` prend en charge les options suivantes.

- `--name` – Nom de la fonction.
- `--profile` – Nom d'un profil dans votre [fichier d'informations d'identification Kit AWS SDK pour .NET](#).
- `--region` – La région AWS où créer la fonction.

Ces options sont enregistrées dans un fichier nommé `aws-lambda-tools-defaults.json`.

Créez un projet de fonction avec le modèle `lambda.EmptyFunction`.

```
dotnet new lambda.EmptyFunction --name 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" : "default",
"region" : "us-east-2",
"configuration" : "Release",
"framework" : "netcoreapp2.1",
"function-runtime": "dotnetcore3.1",
"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 de plus amples informations sur les conditions de sérialisation et les options, veuillez consulter [Sérialisation des fonctions Lambda \(p. 422\)](#). 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
```

AWS Toolkit for Visual Studio

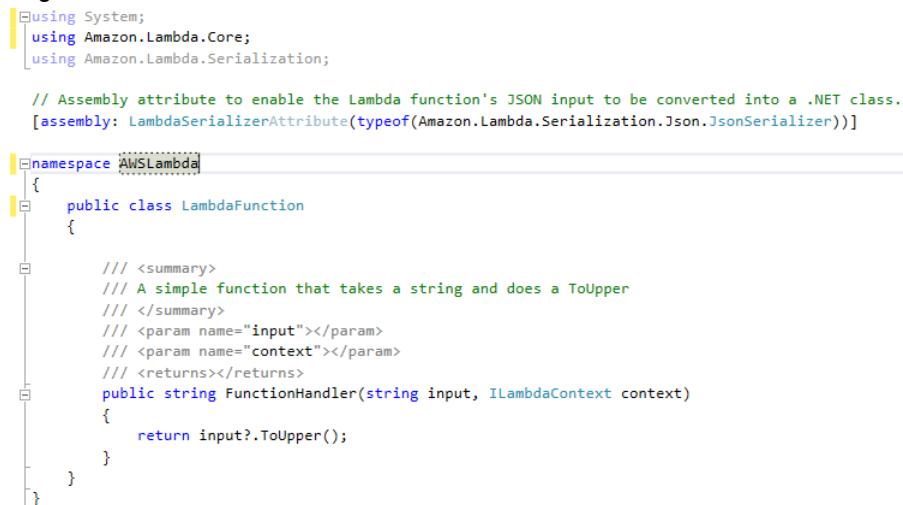
Vous pouvez créer des applications Lambda basées sur .NET à l'aide du plug-in Lambda d'[AWS Toolkit for Visual Studio](#). Le kit d'outils est disponible sous la forme d'une [extension Visual Studio](#).

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 de bout en bout, choisissez Empty Function (Fonction vide), puis Finish (Finir).
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 :
 - `profile` – Nom d'un profil dans votre [fichier d'informations d'identification Kit AWS SDK pour .NET](#).
 - `function-handler` – Emplacement où `function handler` est spécifié, ce qui explique que vous n'avez pas à le définir dans l'Assistant. Toutefois, 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` file.

```
{
    "profile": "default",
    "region" : "us-east-2",
    "configuration" : "Release",
    "framework" : "netcoreapp2.1",
    "function-runtime": "dotnetcore3.1",
    "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 (Détail des fonctions avancées), configurez les options suivantes :

- Role Name (Nom de rôle) (obligatoire) – [Rôle IAM \(p. 33\)](#) que AWS Lambda endosse lors de l'exécution de votre fonction.
 - Environment (Environnement) – Paires clé-valeur que Lambda définit dans l'environnement d'exécution. [Utilisez des variables d'environnement \(p. 55\)](#) pour étendre la configuration de votre fonction en dehors du code.
 - Memory (Mémoire) – Quantité de mémoire disponible pour la fonction pendant son exécution. Choisissez un volume compris [entre 128 Mo et 3,008 MB \(p. 30\)](#) par incrément de 64 Mo.
 - Timeout (Expiration) – Temps autorisé par Lambda pour l'exécution d'une fonction avant de l'arrêter. Le durée par défaut est 3 secondes. La valeur maximale autorisée est 900 secondes.
 - VPC – Si votre fonction a besoin d'un accès réseau à des ressources qui ne sont pas disponibles sur Internet, [configurez-la pour qu'elle se connecte à un VPC \(p. 81\)](#).
 - DLQ – Si votre fonction est appelée de façon asynchrone, [choisissez une file d'attente ou une rubrique \(p. 99\)](#) pour recevoir les appels ayant échoué.
 - Enable active tracing (Activer le traçage actif) – Exemples de demandes entrantes et [suivi des exemples de demandes avec AWS X-Ray \(p. 296\)](#).
7. 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 en 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`.

- `ILambdaContext context` – Le second argument de la signature du gestionnaire est facultatif. Il donne accès à l'[objet de contexte \(p. 424\)](#) qui dispose d'informations sur la fonction et la demande.

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 de plus amples informations, veuillez consulter [Utilisation des fonctions asynchrones en C# avec AWS Lambda \(p. 423\)](#).

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 de plus amples informations, veuillez consulter [Package de déploiement AWS Lambda dans C# \(p. 414\)](#).

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 de plus amples informations, veuillez consulter [Interface de ligne de commande .NET Core \(p. 414\)](#). 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 (surchargées).

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 :

- Utilisez le package `Amazon.Lambda.Serialization.Json`. Cette bibliothèque utilise JSON.NET pour gérer la sérialisation.
- Créez 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 \(p. 420\)](#).

Restrictions liées au gestionnaire de fonctions 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 plus d'informations, 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 en 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`. Lorsque vous utilisez cette signature, Lambda exécute la fonction de façon synchrone et attend qu'elle renvoie une réponse ou que l'exécution [expire](#) (p. 53).

```
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 prend 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 en C#

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au [gestionnaire \(p. 419\)](#). Cet objet fournit les propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `FunctionName` – Nom de la fonction Lambda.
- `FunctionVersion` – [Version \(p. 70\)](#) de la fonction.
- `InvokedFunctionArn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `AwsRequestId` – Identifiant de la demande d'appel.
- `LogGroupName` – Groupe de journaux de la fonction.
- `LogStreamName` – Flux de journal pour l'instance de la fonction.
- `RemainingTime (TimeSpan)` – Nombre de millisecondes restant avant l'expiration de l'exécution.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
- `Logger` L'[objet enregistreur d'événements \(p. 424\)](#) 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 en C#

Votre fonction Lambda est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur [la classe de console](#) ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout`

ou `stderr`. L'exemple suivant utilise la classe `LambdaLogger` provenant de la bibliothèque [Amazon.Lambda.Core \(p. 413\)](#).

Example [src/blank-csharp/Function.cs](#) – Journalisation

```
public async Task<AccountUsage> FunctionHandler(SQSEvent invocationEvent, ILambdaContext context)
{
    GetAccountSettingsResponse accountSettings;
    try
    {
        accountSettings = await callLambda();
    }
    catch (AmazonLambdaException ex)
    {
        throw ex;
    }
    AccountUsage accountUsage = accountSettings.AccountUsage;
    LambdaLogger.Log("ENVIRONMENT VARIABLES: " +
JsonConvert.SerializeObject(System.Environment.GetEnvironmentVariables()));
    LambdaLogger.Log("CONTEXT: " + JsonConvert.SerializeObject(context));
    LambdaLogger.Log("EVENT: " + JsonConvert.SerializeObject(invocationEvent));
    return accountUsage;
}
```

Example Format des journaux

```
START RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "AWS_EXECUTION_ENV": "AWS_Lambda_dotnetcore2.1",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "256",
    "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/blank-csharp-function-WU56XMPLV2XA",
    "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
    "AWS_LAMBDA_LOG_STREAM_NAME": "2020/03/27/[$LATEST]5296xmpl084f411d9fb73b258393f30c",
    "AWS_LAMBDA_FUNCTION_NAME": "blank-csharp-function-WU56XMPLV2XA",
    ...
}
EVENT:
{
    "Records": [
        {
            "MessageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
            "ReceiptHandle": "MessageReceiptHandle",
            "Body": "Hello from SQS!",
            "Md5OfBody": "7b270e59b47ff90a553787216d55d91d",
            "Md5OfMessageAttributes": null,
            "EventSourceArn": "arn:aws:sqs:us-west-2:123456789012:MyQueue",
            "EventSource": "aws:sqs",
            "AwsRegion": "us-west-2",
            "Attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1523232000000",
                "SenderId": "123456789012",
                "ApproximateFirstReceiveTimestamp": "1523232000001"
            },
            ...
        }
    ]
}
END RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d
REPORT RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d Duration: 4157.16 ms Billed Duration: 4200 ms Memory Size: 256 MB Max Memory Used: 99 MB Init Duration: 841.60 ms XRAY TraceId: 1-5e7e8131-7ff0xmpl32bfb31045d0a3bb SegmentId: 0152xmpl6016310f Sampled: true
```

L'environnement d'exécution .NET enregistre les lignes `START`, `END` et `REPORT` pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY Traceld – Pour les demandes suivies, [l'ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 426\)](#)
- [Utilisation de l'AWS CLI \(p. 426\)](#)
- [Suppression de journaux \(p. 428\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
```

```
    "LogResult":  
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d  
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST  
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-  
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",  
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed  
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash  
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out  
sed -i'' -e 's/"/\n/g' out  
sleep 15  
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat  
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh  
{  
    "StatusCode": 200,  
    "ExecutedVersion": "$LATEST"  
}  
{  
    "events": [  
        {  
            "timestamp": 1559763003171,  
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:  
$LATEST\n",  
            "ingestionTime": 1559763003309  
        },  
        {  
            "timestamp": 1559763003173,  
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf  
\tINFO\tENVIRONMENT VARIABLES\r{\r\t\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",  
            "ingestionTime": 1559763018353  
        },  
        {  
            "timestamp": 1559763003173,  
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf  
\tINFO\tEVENT\r{\r\t\t\"key\": \"value\"\r}\n",  
        }  
    ]  
}
```

```
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

Erreurs de fonction AWS Lambda en 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 fonctions Lambda \(p. 422\)](#)), 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.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.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 de plus amples informations, veuillez consulter [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).

Instrumentation du code C # dans AWS Lambda

Lambda s'intègre à AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle parcourt les ressources

de votre application, de l'API frontale au stockage et à la base de données sur le backend. En ajoutant simplement la bibliothèque SDK X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et la latence pour tous les appels que votre fonction adresse à un service AWS.

La cartographie du service X-Ray indique le flux des demandes dans votre application. L'exemple suivant de l'exemple d'application [processeur d'erreurs \(p. 308\)](#) montre une application avec deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La seconde fonction traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le kit SDK AWS pour appeler X-Ray, Amazon S3 et Amazon CloudWatch Logs.



Pour effectuer un suivi des requêtes qui n'ont pas d'en-tête de suivi, activez le suivi actif dans la configuration de votre fonction.

Activer le suivi actif

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Sous AWS X-Ray, choisissez Active tracing (Suivi actif).
4. Choisissez Enregistrer.

Tarification

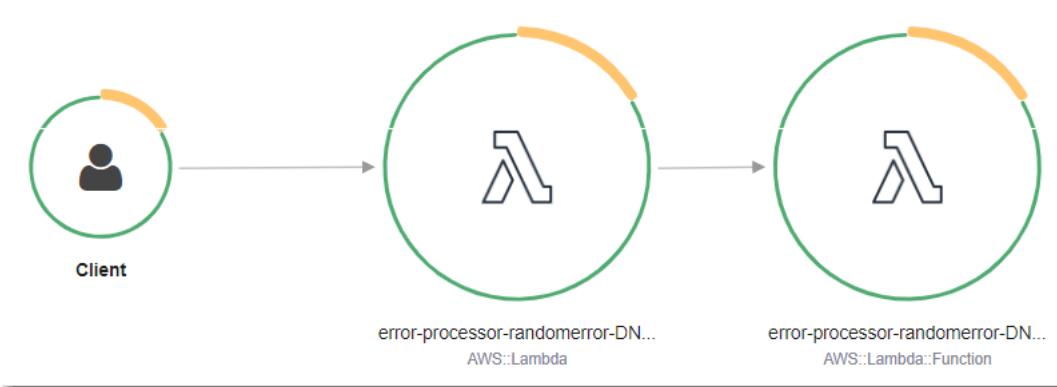
X-Ray propose une offre gratuite perpétuelle. Au-delà du seuil de niveau gratuit, X-Ray facture le stockage et la récupération du suivi. Pour de plus amples informations, consultez [Tarification AWS X-Ray](#).

Votre fonction a besoin d'une autorisation pour télécharger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, ce dernier ajoute les autorisations requises au [rôle d'exécution \(p. 33\)](#) de votre fonction. Sinon, ajoutez la stratégie [AWSXRayDaemonWriteAccess](#) au rôle d'exécution.

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. La règle d'échantillonnage par défaut est 1 demande par seconde et 5 % de demandes supplémentaires.

Lorsque le suivi actif est activé, Lambda enregistre un suivi pour un sous-ensemble d'appels. Lambda enregistre deux segments, ce qui crée deux nœuds sur la cartographie des services. Le premier nœud

représente le service Lambda qui reçoit la demande d'appel. Le deuxième nœud est enregistré par l'environnement d'exécution (p. 18) de la fonction.



Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer des détails sur les appels que le gestionnaire effectue vers d'autres ressources et services, utilisez le kit SDK X-Ray pour .NET. Pour obtenir ce kit SDK, ajoutez les packages `AWSXRayRecorder` à votre fichier de projet.

Example [src/blank-csharp/blank-csharp.csproj](#)

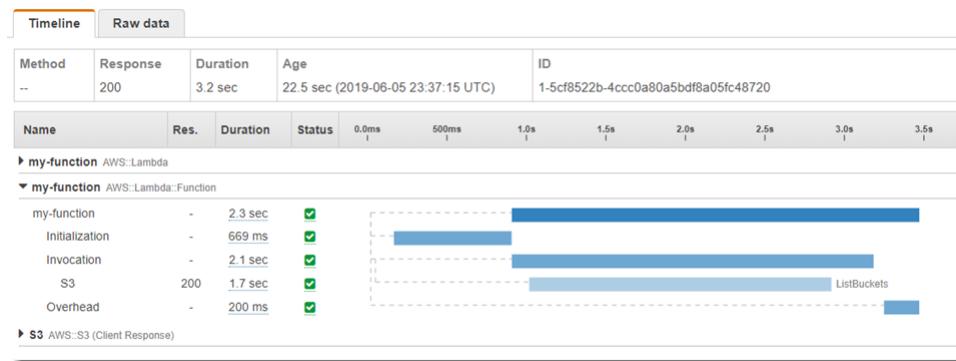
```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
</PropertyGroup>
<ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="12.0.3" />
    <PackageReference Include="Amazon.Lambda.Core" Version="1.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="1.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.7.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.3.104.38" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.3.108.11" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.6.2" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.7.2" />
</ItemGroup>
</Project>
```

Pour instrumenter les clients AWS SDK, appelez la méthode `RegisterXRayForAllServices` dans le code d'initialisation.

Example [src/blank-csharp/Function.cs](#) – Initialiser X-Ray

```
static async void initialize() {
    AWSSDKHandler.RegisterXRayForAllServices();
    lambdaClient = new AmazonLambdaClient();
    await callLambda();
}
```

L'exemple suivant illustre une trace avec 2 segments. Les deux sont nommés my-function, mais l'un est de type `AWS::Lambda` et l'autre de type `AWS::Lambda::Function`. Le segment de fonction est développé pour afficher ses sous-segments.



Le premier segment représente la demande d'appel traitée par le service Lambda. Le deuxième segment enregistre le travail effectué par votre fonction. Le segment de fonction comporte 3 sous-segments.

- Initialization (Initialisation) – Représente le temps passé à charger votre fonction et à exécuter le [code d'initialisation \(p. 19\)](#). Ce sous-segment n'apparaît que pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le travail effectué par votre code de gestionnaire. En instrumentant votre code, vous pouvez étendre ce sous-segment avec des sous-segments supplémentaires.
- Overhead (Travail supplémentaire) – Représente le travail effectué par l'environnement d'exécution Lambda pour préparer le traitement de l'événement suivant.

Vous pouvez également utiliser des clients HTTP, enregistrer des requêtes SQL et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez [Kit SDK X-Ray pour .NET](#) dans le Manuel du développeur AWS X-Ray.

Sections

- [Activation du suivi actif avec l'API Lambda \(p. 433\)](#)
- [Activation du suivi actif avec AWS CloudFormation \(p. 434\)](#)

Activation du suivi actif avec l'API Lambda

Pour gérer la configuration de suivi à l'aide de l'interface de ligne de commande AWS ou du kit AWS SDK, utilisez les opérations d'API suivantes :

- [UpdateFunctionConfiguration \(p. 643\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [CreateFunction \(p. 504\)](#)

L'exemple de commande AWS CLI suivant active le suivi actif sur une fonction nommée my-function.

```
$ aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique à la version qui est verrouillée lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi actif d'une ressource `AWS::Lambda::Function` dans un modèle AWS CloudFormation, utilisez la propriété `TracingConfig`.

Example [function-inline.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Pour une ressource Modèle d'application sans serveur AWS (AWS SAM) `AWS::Serverless::Function`, utilisez la propriété `Tracing`.

Example [template.yml](#) – Configuration du suivi

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

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.

Environnements d'exécution .NET

| Nom | Identifiant | Système d'exploitation | |
|---------------|---------------|------------------------|--|
| .NET Core 3.1 | dotnetcore3.1 | Amazon Linux 2 | |
| .NET Core 2.1 | dotnetcore2.1 | Amazon Linux | |

Note

Pour commencer à développer des applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le référentiel GitHub de ce guide.

Exemples d'applications Lambda dans PowerShell

- [blank-powershell](#) – Fonction PowerShell qui montre l'utilisation de la journalisation, des variables d'environnement et du kit SDK AWS.

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 \(p. 435\)](#).

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 [Package de déploiement AWS Lambda dans PowerShell \(p. 436\)](#).

Rubriques

- [Configuration d'un environnement de développement PowerShell \(p. 435\)](#)
- [Package de déploiement AWS Lambda dans PowerShell \(p. 436\)](#)
- [Gestionnaire de fonctions AWS Lambda dans PowerShell \(p. 438\)](#)
- [Objet de contexte AWS Lambda dans PowerShell \(p. 439\)](#)
- [Journalisation des fonctions AWS Lambda dans PowerShell \(p. 439\)](#)
- [Erreurs de fonction AWS Lambda dans PowerShell \(p. 443\)](#)

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 de PowerShell par Lambda 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 3.1 – Étant donné que PowerShell Core repose sur .NET Core, la prise en charge de PowerShell par Lambda utilise le même environnement d'exécution Lambda .NET Core 3.1 pour les fonctions Lambda .NET Core et PowerShell. Le kit SDK .NET Core 3.1 est utilisé par les nouvelles applets de commande de publication Lambda PowerShell pour créer le package de déploiement Lambda. Le kit SDK .NET Core 3.1 est disponible dans les [téléchargements .NET](#) sur le site web de Microsoft. Veillez à installer le kit SDK et non l'environnement d'exécution (runtime).
3. Installez le module AWSLambdaPSCore – Vous pouvez l'installer à partir de la [PowerShell Gallery](#) ou à l'aide de la commande d'interpréteur PowerShell Core suivante :

```
Install-Module AWSLambdaPSCore -Scope CurrentUser
```

4. (Facultatif) Installez les Outils AWS pour PowerShell – Vous pouvez installer la version modulaire [AWS.Tools](#) ou [AWSPowerShell.NetCore](#) dans PowerShell Core 6.0 pour utiliser l'API Lambda dans votre environnement PowerShell. Pour obtenir des instructions, veuillez consulter [Installation des outils AWS pour PowerShell](#)

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.

Le module AWSLambdaPSCore possède les nouvelles applets de commande suivantes pour vous aider à créer et publier des fonctions Lambda PowerShell.

Cmdlets AWSLambdaPSCore

- Get-AWSPowerShellLambdaTemplate – Renvoie une liste de modèles de mise en route.
- New-AWSPowerShellLambda – Crée un script PowerShell basée sur un modèle.
- Publish-AWSPowerShellLambda – Publie un script PowerShell pour Lambda.
- New-AWSPowerShellLambdaPackage – Crée un package de déploiement Lambda qui peut être utilisé dans un système de CI/CD 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. 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 |

...

- 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='3.3.618.0'}  
  
# Uncomment to send the input to CloudWatch Logs  
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

- 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
```

- 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-2
```

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.

- Appelez votre fonction avec la commande de l'interface de ligne de commande AWS `invoke`.

```
> aws lambda invoke --function-name MyFirstPSScript out
```

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='3.3.618.0'}
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 \(À propos de Requires\)](#). Pour de plus amples informations sur les packages de déploiement PowerShell, veuillez consulter [Package de déploiement AWS Lambda dans PowerShell \(p. 436\)](#).

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 mettant une variable \$LambdaContext à la disposition du [gestionnaire \(p. 438\)](#). 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` – Nom de la fonction Lambda.
- `FunctionVersion` – [Version \(p. 70\)](#) de la fonction.
- `InvokedFunctionArn` – Amazon Resource Name (ARN) qui est utilisé pour appeler cette fonction. Indique si le mécanisme d'appel a spécifié un numéro de version ou un alias.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `AwsRequestId` – Identifiant de la demande d'appel.
- `LogGroupName` – Groupe de journaux de la fonction.
- `LogStreamName` – Flux de journal pour l'instance de la fonction.
- `RemainingTime` – Nombre de millisecondes restant avant l'expiration de l'exécution.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
- `Logger` – L'[objet Logger \(p. 439\)](#) 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='3.3.618.0'}
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 est fournie avec un groupe de journaux CloudWatch Logs, avec un flux de journal pour chaque instance de votre fonction. L'environnement d'exécution envoie des détails sur chaque appel au flux de journal et relaie les journaux et autres sorties provenant du code de votre fonction.

Pour produire des journaux à partir de votre code de fonction, vous pouvez utiliser des cmdlets sur [Microsoft.PowerShell.Utility](#), ou tout module de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant utilise `Write-Host`.

Example [function/Handler.ps1](#) – Journalisation

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
```

```
Write-Host `## Environment variables
Write-Host AWS_LAMBDA_FUNCTION_VERSION=$Env:AWS_LAMBDA_FUNCTION_VERSION
Write-Host AWS_LAMBDA_LOG_GROUP_NAME=$Env:AWS_LAMBDA_LOG_GROUP_NAME
Write-Host AWS_LAMBDA_LOG_STREAM_NAME=$Env:AWS_LAMBDA_LOG_STREAM_NAME
Write-Host AWS_EXECUTION_ENV=$Env:AWS_EXECUTION_ENV
Write-Host AWS_LAMBDA_FUNCTION_NAME=$Env:AWS_LAMBDA_FUNCTION_NAME
Write-Host PATH=$Env:PATH
Write-Host `## Event
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 3)
```

Example format des journaux

```
START RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Version: $LATEST
Importing module ./Modules/AWSPowerShell.NetCore/3.3.618.0/AWSPowerShell.NetCore.ps1
[Information] - ## Environment variables
[Information] - AWS_LAMBDA_FUNCTION_VERSION=$LATEST
[Information] - AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/blank-powershell-
function-18CIXMPLHFAJJ
[Information] - AWS_LAMBDA_LOG_STREAM_NAME=2020/04/01/
[$LATEST]53c5xmpl52d64ed3a744724d9c201089
[Information] - AWS_EXECUTION_ENV=AWS_Lambda_dotnetcore2.1_powershell_1.0.0
[Information] - AWS_LAMBDA_FUNCTION_NAME=blank-powershell-function-18CIXMPLHFAJJ
[Information] - PATH=/var/lang/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin
[Information] - ## Event
[Information] -
{
    "Records": [
        {
            "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1523232000000",
                "SenderId": "123456789012",
                "ApproximateFirstReceiveTimestamp": "1523232000001"
            },
            ...
    ]
}
END RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed
REPORT RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Duration: 3906.38 ms Billed
Duration: 4000 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19 ms
XRAY TraceId: 1-5e843da6-733cxmpl7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled: true
```

L'environnement d'exécution .NET enregistre les lignes START, END et REPORT pour chaque appel. La ligne de rapport fournit les détails suivants.

Journal des rapports

- RequestID – L'ID de demande unique pour l'appel.
- Durée– Temps nécessaire pour le traitement de l'événement par la méthode de gestion de votre fonction.
- Durée facturée – Durée facturée pour l'appel.
- Taille de la mémoire – Quantité de mémoire allouée à la fonction.
- Mémoire maximale utilisée – La quantité de mémoire utilisée par la fonction.
- Durée d'initialisation – Pour la première demande servie, le temps qu'il a fallu au moteur d'exécution pour charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY Traceld – Pour les demandes suivies, [l'ID de suivi AWS X-Ray \(p. 296\)](#).
- SegmentId – Pour les demandes suivies, l'ID du segment X-Ray.
- Échantillonné– Pour les demandes suivies, le résultat de l'échantillonnage.

Vous pouvez afficher les journaux dans la console Lambda, dans la console CloudWatch Logs ou à partir de l'interface de ligne de commande.

Sections

- [Affichage des journaux dans la AWS Management Console \(p. 441\)](#)
- [Utilisation de l'AWS CLI \(p. 441\)](#)
- [Suppression de journaux \(p. 443\)](#)

Affichage des journaux dans la AWS Management Console

La console Lambda affiche la sortie de journal lorsque vous testez une fonction sur la page de configuration de la fonction. Pour afficher les journaux pour tous les appels, utilisez la console CloudWatch Logs.

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.

Chaque flux de journal correspond à une [instance de votre fonction \(p. 124\)](#). De nouveaux flux apparaissent lorsque vous mettez à jour votre fonction et lorsque des instances supplémentaires sont créées pour traiter plusieurs appels simultanés. Pour trouver les journaux associés à des appels spécifiques, vous pouvez instrumenter votre fonction avec X-Ray et enregistrer les détails sur la demande et le flux de journal dans le suivi. Pour obtenir un exemple d'application qui met en corrélation les journaux et les suivis avec X-Ray, veuillez consulter [Exemple d'application du processeur d'erreurs pour AWS Lambda \(p. 308\)](#).

Utilisation de l'AWS CLI

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 my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez utiliser l'utilitaire `base64` pour décoder les journaux.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
    "AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"", ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, MacOS et [Ubuntu sous Windows](#). Pour MacOS, la commande est `base64 -D`.

Pour obtenir des événements de journaux complets à partir de la ligne de commande, vous pouvez inclure le nom du flux de journaux dans la sortie de votre fonction, comme illustré dans l'exemple précédent. L'exemple de script suivant appelle une fonction nommée `my-function` et télécharge les cinq derniers événement de journaux.

Example Script get-logs.sh

Cet exemple nécessite que `my-function` renvoie un ID de flux de journal.

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"/\n/g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

```
$ ./get-logs.sh
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
```

```
}
```

Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement lorsque vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez les groupes de journaux ou [configurez une période de rétention](#) après laquelle les journaux seront automatiquement supprimés.

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 vous appelez 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"
}
```

Surveillance et dépannage des applications Lambda

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. Il publie également les mesures CloudWatch associées. Vous pouvez tirer parti de ces métriques pour configurer des alarmes CloudWatch personnalisées.

La console Lambda fournit un [tableau de bord de surveillance \(p. 444\)](#) intégré pour chacune de vos fonctions et applications.

Pour surveiller une fonction

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Surveillance.

Tarification

CloudWatch propose une offre gratuite perpétuelle. Au-delà du seuil de cette offre gratuite, CloudWatch facture les métriques, les tableaux de bord, les alarmes, les journaux et les informations. Pour plus d'informations, consultez [Tarification CloudWatch](#).

Chaque fois que votre fonction est appelée, Lambda enregistre les [métriques \(p. 445\)](#) de la demande, la réponse de la fonction et l'état général de cette dernière. Vous pouvez utiliser des métriques pour définir des alarmes qui se déclenchent lorsque les performances de la fonction se dégradent ou lorsque vous rapprochez des limites de simultanéité dans la région active.

Pour déboguer votre code et vérifier qu'il fonctionne comme prévu, vous pouvez générer des journaux avec la fonctionnalité de journalisation standard pour votre langage de programmation. L'environnement d'exécution Lambda charge la sortie du journal de votre fonction dans CloudWatch Logs. Vous pouvez [afficher les journaux \(p. 447\)](#) dans la console CloudWatch Logs, dans la console Lambda ou à partir de l'interface de ligne de commande.

En plus de surveiller les journaux et les métriques dans CloudWatch, vous pouvez utiliser AWS X-Ray pour suivre et déboguer les demandes servies par votre application. Pour plus d'informations, consultez [Utilisation d'AWS Lambda avec AWS X-Ray \(p. 296\)](#).

Sections

- [Surveillance des fonctions dans la console AWS Lambda \(p. 444\)](#)
- [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#)
- [Accès aux journaux Amazon CloudWatch pour AWS Lambda \(p. 447\)](#)

Surveillance des fonctions dans la console AWS Lambda

AWS Lambda surveille les fonctions en votre nom et envoie les métriques à Amazon CloudWatch. Ces métriques incluent le nombre total de demandes, la durée et les taux d'erreur. La console Lambda crée les graphiques pour ces métriques et les affiche sur la page Monitoring de chaque fonction.

Pour accéder à la console de surveillance

1. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez Surveillance.

La console fournit les graphiques suivants.

Surveillance des graphiques Lambda

- Appels : – Le nombre de fois que la fonction a été appelée par période de 5 minutes.
- Durée : – Le temps d'exécution moyen, minimum et maximum.
- Nombre d'erreurs et taux de réussite (%) : – Le nombre d'erreurs et le pourcentage d'exécutions terminées sans erreur.
- Limitations : – Le nombre de fois où l'exécution a échoué en raison de limites de simultanéité.
- IteratorAge : – Pour les sources d'événements de flux, l'âge du dernier élément du lot lorsque Lambda le reçoit et appelle la fonction.
- Échecs de remise asynchrone – Nombre d'erreurs survenues quand Lambda a tenté d'écrire dans une file d'attente de destination ou de lettres mortes.
- Exécutions simultanées – Nombre d'instances de fonction qui traitent des événements.

Pour consulter la définition d'un graphique dans CloudWatch, choisissez Afficher dans les métriques dans le menu en haut à droite du graphique. Pour de plus amples informations sur les métriques que Lambda enregistre, veuillez consulter [Utilisation des métriques de fonction AWS Lambda \(p. 445\)](#).

La console montre également les rapports d'Insights CloudWatch Logs qui sont compilés à partir des informations des journaux de votre fonction. Vous pouvez ajouter ces rapports à un tableau de bord personnalisé dans la console CloudWatch Logs. Utilisez ces requêtes comme point de départ pour vos propres rapports.

Pour afficher une demande, choisissez View dans CloudWatch Logs Insights dans le menu en haut à droite du rapport.

Utilisation des métriques de fonction AWS Lambda

Lorsque votre fonction termine le traitement d'un événement, Lambda envoie des métriques concernant l'appel à Amazon CloudWatch. Vous pouvez générer des graphiques et des tableaux de bord avec ces métriques dans la console CloudWatch et définir des alarmes pour répondre aux modifications des taux d'utilisation, de performances et d'erreur. Utilisez les dimensions pour filtrer et trier les métriques de fonction par nom, alias ou version de fonction.

Pour afficher les métriques sur la console CloudWatch

1. Ouvrez la [page Métriques de la console Amazon CloudWatch](#) (espace de noms AWS/Lambda).
2. Choisissez une dimension.
 - Par nom de fonction (`FunctionName`) – Affichez les métriques d'agrégation pour toutes les versions et tous les alias d'une fonction.
 - Par ressource (`Resource`) – Affichez les métriques pour une version ou un alias d'une fonction.
 - By Executed Version (Par version exécutée) (`ExecutedVersion`) – Affichez les métriques pour une combinaison d'alias et de version. Utilisez la dimension `ExecutedVersion` pour comparer les taux d'erreur pour deux versions d'une fonction qui sont les cibles d'un [alias pondéré \(p. 72\)](#).

- Sur toutes les fonctions (aucun) – Affichez les métriques agrégées pour toutes les fonctions figurant dans la région AWS actuelle.
3. Choisissez des métriques pour les ajouter au graphique.

Par défaut, les graphiques utilisent la statistique `Sum` pour toutes les métriques. Pour choisir une autre statistique et personnaliser le graphique, utilisez les options de l'onglet Graphique des métriques.

L'horodatage d'une métrique indique le moment où la fonction a été appelée. En fonction de la durée de l'exécution, il peut se rapporter à plusieurs minutes avant l'émission de la métrique. Si, par exemple, le délai d'expiration de votre fonction est de 10 minutes, effectuez une recherche plus de 10 minutes dans le passé pour obtenir des métriques précises.

Pour plus d'informations sur CloudWatch, consultez le [Guide de l'utilisateur Amazon CloudWatch](#).

Sections

- [Utilisation des métriques d'appel \(p. 446\)](#)
- [Utilisation des métriques de performances \(p. 447\)](#)
- [Utilisation des métriques de simultanéité \(p. 447\)](#)

Utilisation des métriques d'appel

Les métriques d'appel sont des indicateurs binaires du résultat d'un appel. Par exemple, si la fonction renvoie une erreur, Lambda envoie la métrique `Errors` avec une valeur de 1. Pour obtenir le nombre d'erreurs de fonction qui se sont produites chaque minute, examinez la somme (`Sum`) de la métrique `Errors` avec une période d'une minute.

Vous devez visualiser les métriques suivantes avec la statistique `Sum`.

Métriques d'appel

- **Invocations** – Nombre de fois où votre code de fonction est exécuté, y compris les exécutions réussies et les exécutions qui entraînent une erreur de fonction. Les appels ne sont pas enregistrés si la demande d'appel est limitée ou si elle entraîne une [erreur d'appel \(p. 567\)](#). Ceci est égal au nombre de demandes facturées.
- **Errors** – Nombre d'appels qui entraînent une erreur de fonction. Les erreurs de fonction incluent les exceptions levées par votre code et les exceptions levées par le runtime Lambda. Le runtime renvoie des erreurs pour des problèmes tels que les expirations de délai et les erreurs de configuration. Pour calculer le taux d'erreur, divisez la valeur `Errors` par la valeur `Invocations`.
- **DeadLetterErrors** – Pour un [appel asynchrone \(p. 93\)](#), nombre de fois où Lambda tente d'envoyer un événement à une file d'attente de lettres mortes mais échoue. Des erreurs de lettre morte peuvent survenir en raison d'erreurs d'autorisations, de ressources mal configurées ou de limites de taille.
- **DestinationDeliveryFailures** – Pour un appel asynchrone, nombre de fois où Lambda tente d'envoyer un événement à une [destination \(p. 26\)](#) mais échoue. Des erreurs de remise peuvent se produire en raison d'erreurs d'autorisations, de ressources mal configurées ou de limites de taille.
- **Throttles** – Nombre de demandes d'appel qui sont soumises à une limitation. Lorsque toutes les instances de fonction traitent des demandes et qu'aucune simultanéité n'est disponible pour la mise à l'échelle, Lambda rejette les demandes supplémentaires avec [TooManyRequestsException \(p. 567\)](#). Les demandes limitées et les autres erreurs d'appel ne comptent pas comme `Invocations` ou `Errors`.
- **ProvisionedConcurrencyInvocations** – Nombre de fois où votre code de fonction est exécuté sur la [simultanéité provisionnée \(p. 61\)](#).
- **ProvisionedConcurrencySpilloverInvocations** – Nombre de fois où votre code de fonction est exécuté sur la simultanéité standard lorsque toute la simultanéité provisionnée est en cours d'utilisation.

Utilisation des métriques de performances

Les métriques de performance fournissent des détails sur les performances d'un appel individuel. Par exemple, la métrique `Duration` indique la durée en millisecondes que votre fonction consacre au traitement d'un événement. Pour avoir une idée de la rapidité avec laquelle votre fonction traite les événements, affichez ces métriques avec la statistique `Average` ou `Max`.

Métriques de performances

- `Duration` – Durée de traitement d'un événement par votre code de fonction. Pour le premier événement traité par une instance de votre fonction, elle inclut le [temps d'initialisation \(p. 19\)](#). La durée facturée pour un appel est la valeur `Duration` arrondie aux 100 millisecondes les plus proches.
- `IteratorAge` – Pour les [mappages de source d'événement \(p. 101\)](#) qui lisent à partir des flux, âge du dernier enregistrement dans l'événement. Cet âge correspond au temps écoulé entre le moment où le flux reçoit l'enregistrement et le moment où le mappage de source d'événement envoie l'événement à la fonction.

`Duration` prend également en charge les [statistiques sur les centiles](#). Utilisez les centiles pour exclure les valeurs aberrantes qui faussent les statistiques moyennes et maximales. Par exemple, la statistique P95 indique la durée maximale de 95 % des exécutions, en excluant les 5 % les plus lentes.

Utilisation des métriques de simultanéité

Lambda signale les métriques de simultanéité sous la forme d'un nombre agrégé du nombre d'instances traitant des événements au sein d'une fonction, d'une version, d'un alias ou d'une région AWS. Pour voir la distance qui vous sépare des limites de simultanéité, affichez ces métriques à l'aide de la statistique `Max`.

Métriques de simultanéité

- `ConcurrentExecutions` – Nombre d'instances de fonction qui traitent des événements. Si ce nombre atteint votre [limite d'exécutions simultanées \(p. 30\)](#) pour la région ou la [limite de simultanéité réservée \(p. 61\)](#) que vous avez configurée sur la fonction, les demandes d'appel supplémentaires sont limitées.
- `ProvisionedConcurrentExecutions` – Nombre d'instances de fonction qui traitent des événements sur la [simultanéité provisionnée \(p. 61\)](#). Pour chaque appel d'un alias ou d'une version avec simultanéité provisionnée, Lambda émet le nombre actuel.
- `ProvisionedConcurrencyUtilization` – Pour une version ou un alias, valeur `ProvisionedConcurrentExecutions` divisée par le montant total de la simultanéité provisionnée allouée. Par exemple, .5 indique que 50 % de la simultanéité provisionnée allouée est en cours d'utilisation.
- `UnreservedConcurrentExecutions` – Pour une région AWS, nombre d'événements qui sont traités par des fonctions qui n'ont pas de simultanéité réservée.

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 plus d'informations sur les groupes de journaux et pour découvrir comment y accéder via la console CloudWatch, consultez [Surveillance des fichiers journaux système, d'applications et personnalisés](#) dans le Guide de l'utilisateur Amazon CloudWatch.

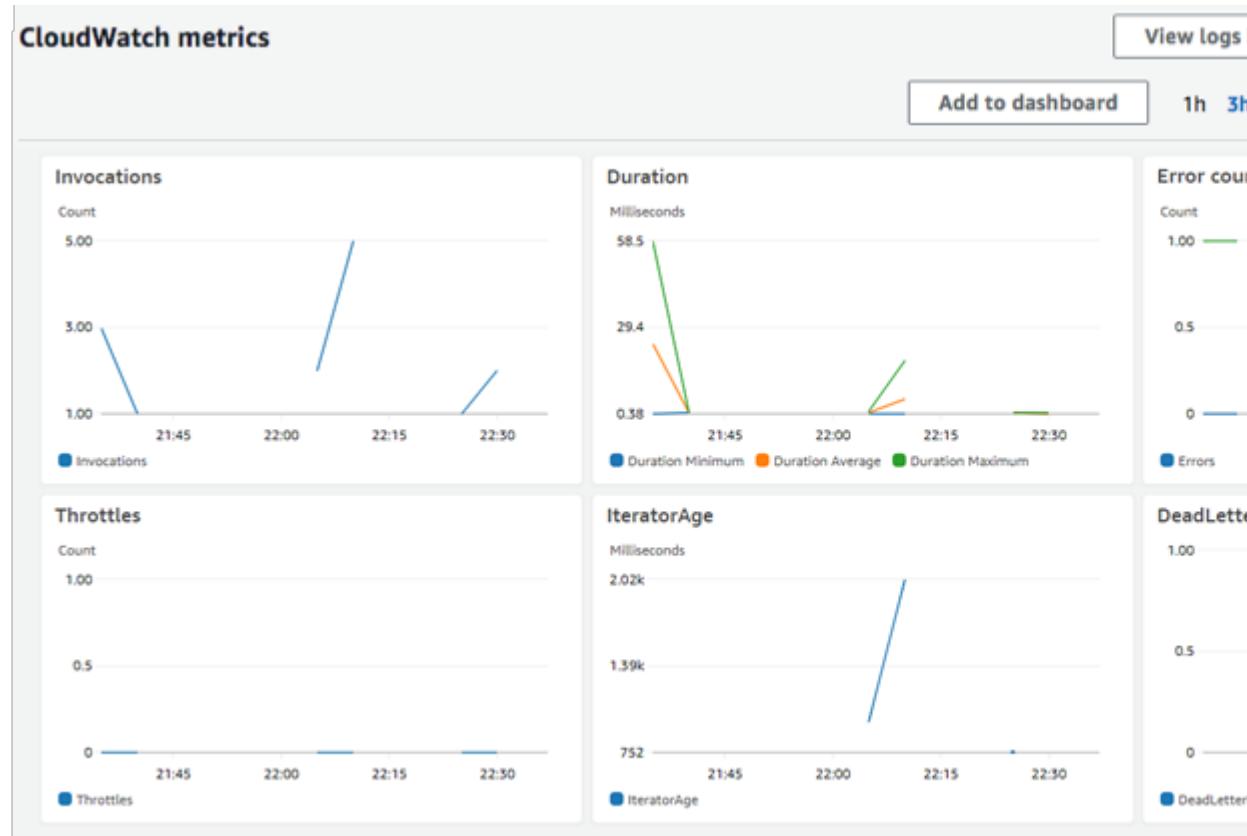
Pour afficher les journaux correspondant à Lambda, utilisez la console Lambda, la console CloudWatch, l'AWS CLI ou l'API CloudWatch. La procédure suivante vous montre comment afficher les journaux via la console Lambda.

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. Ouvrez la [page Fonctions](#) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Surveillance.



Une représentation graphique des métriques de la fonction Lambda s'affiche.

4. Choisissez Afficher les journaux dans CloudWatch.

Lambda utilise les autorisations de votre fonction pour charger les journaux sur CloudWatch Logs. Si vous ne voyez pas les journaux dans la console, vérifiez vos [autorisations de rôle d'exécution \(p. 33\)](#).

Sécurité dans AWS Lambda

Chez AWS, la sécurité dans le cloud est notre priorité numéro 1. En tant que client AWS, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses en termes de sécurité.

La sécurité est une responsabilité partagée entre AWS et vous-même. Le [modèle de responsabilité partagée](#) décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- La sécurité du cloud – AWS est responsable de la protection de l'infrastructure qui exécute des services AWS dans le cloud AWS. AWS vous fournit également les services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des [programmes de conformité AWS](#). Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS Lambda, consultez [Services AWS concernés par le programme de conformité](#).
- Sécurité dans le cloud – Votre responsabilité est déterminée par le service AWS que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre entreprise, et la législation et la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de Lambda. Les rubriques suivantes vous montrent comment configurer Lambda pour répondre à vos objectifs de sécurité et de conformité. Vous pouvez également apprendre à utiliser d'autres services AWS qui vous permettent de surveiller et de sécuriser vos ressources Lambda.

Rubriques

- [Protection des données dans AWS Lambda \(p. 449\)](#)
- [Identity and Access Management pour AWS Lambda \(p. 451\)](#)
- [Validation de la conformité pour AWS Lambda \(p. 459\)](#)
- [Résilience dans AWS Lambda \(p. 460\)](#)
- [Sécurité de l'infrastructure dans AWS Lambda \(p. 460\)](#)
- [Configuration et analyse des vulnérabilités dans AWS Lambda \(p. 461\)](#)

Protection des données dans AWS Lambda

AWS Lambda se conforme au [modèle de responsabilité partagée](#) AWS, qui comprend des directives et des régulations pour la protection des données. AWS est responsable de la protection de l'infrastructure globale qui exécute tous les services AWS. AWS conserve le contrôle des données hébergées sur cette infrastructure, y compris les contrôles de configuration de sécurité pour traiter le contenu et les données personnelles des clients. Les clients AWS et les partenaires d'APN, qu'ils agissent en tant que contrôleur ou responsables du traitement des données, sont responsables de toutes les données personnelles qu'ils mettent sur le cloud AWS.

Pour des raisons de protection des données, nous vous recommandons de protéger vos identifiants de compte AWS et de configurer des comptes utilisateurs individuels avec AWS Identity and Access Management (IAM), afin que chaque utilisateur reçoive uniquement les permissions nécessaires pour accomplir ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multi-facteurs (MFA) avec chaque compte.
- Utilisez SSL/TLS pour communiquer avec des ressources AWS.

- Configurez l'API et la consignation des activités utilisateur avec AWS CloudTrail.
- Utilisez des solutions de chiffrement AWS, ainsi que tous les contrôles de sécurité par défaut au sein des services AWS.
- Utilisez des services de sécurité gérés comme Amazon Macie, qui contribue à la découverte et à la sécurisation des données personnelles stockées dans Amazon S3.

Nous vous recommandons vivement de ne jamais placer d'informations identifiables sensibles, telles que les numéros de compte de vos clients, dans des champs de formulaire ou des métadonnées comme les noms de fonction et les balises. Cela vaut lorsque vous utilisez Lambda ou d'autres services AWS à l'aide de la console, d'API, de AWS CLI ou de kits AWS SDK. Toutes les données que vous entrez dans les métadonnées peuvent être récupérées pour être insérées dans les journaux de diagnostic. Lorsque vous fournissez une URL à un serveur externe, n'incluez pas les informations d'identification non chiffrées dans l'URL pour valider votre demande adressée au serveur.

Pour de plus amples informations sur la protection des données, veuillez consulter le billet de blog [Modèle de responsabilité partagée AWS et RGPD](#) à la page Blog sur la sécurité AWS.

Sections

- [Chiffrement en transit \(p. 450\)](#)
- [Chiffrement au repos \(p. 450\)](#)

Chiffrement en transit

Lambda Les points de terminaison d'API ne prennent en charge que des connexions sécurisées sur HTTPS. Lorsque vous gérez des ressources Lambda avec AWS Management Console, le kit SDK AWS ou l'API Lambda, toutes les communications sont chiffrées à l'aide du protocole TLS (Transport Layer Security).

Pour une liste complète des points de terminaison des API, consultez [Régions AWS et points de terminaison](#) dans le AWS General Reference.

Chiffrement au repos

Vous pouvez utiliser des variables d'environnement pour stocker des codes secrets à utiliser en toute sécurité avec les fonctions Lambda. Lambda chiffre toujours chiffre les variables d'environnement au repos.

De plus, vous pouvez utiliser les fonctionnalités suivantes pour personnaliser la façon dont les variables d'environnement sont chiffrées.

- Configuration de clé– Pour chaque fonction, vous pouvez configurer Lambda pour utiliser une clé de chiffrement que vous créez et gérez dans AWS Key Management Service. Elles sont appelées clés CMK gérées par le client ou clés gérées par le client. Si vous ne configurez pas une clé gérée par le client, Lambda utilise une clé CMK gérée par AWS nommée `aws/lambda`, que Lambda crée dans votre compte.
- Assistants de chiffrement– La console Lambda vous permet de chiffrer les valeurs des variables d'environnement côté client avant de les envoyer à Lambda. Cela renforce la sécurité supplémentaire en empêchant des codes secrets non chiffrés d'être affichés dans la console Lambda ou dans la configuration de la fonction renvoyée par l'API Lambda. La console fournit également un exemple de code que vous pouvez adapter pour déchiffrer les valeurs dans votre gestionnaire de fonctions.

Pour plus d'informations, consultez [Utilisation des variables d'environnement AWS Lambda \(p. 55\)](#).

Lambda chiffre toujours les fichiers que vous chargez sur Lambda, y compris les [packages de déploiement \(p. 21\)](#) et les [archives de couches \(p. 76\)](#).

Amazon CloudWatch Logs et AWS X-Ray chiffrent également les données par défaut et peuvent être configurées de façon à utiliser une clé gérée par le client. Pour plus d'informations, consultez [Chiffrement des données de journal CloudWatch Logs](#) et [Protection des données dans AWS X-Ray](#).

Identity and Access Management pour AWS Lambda

AWS Identity and Access Management (IAM) est un service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux ressources AWS. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (disposant des autorisations) à utiliser les ressources Lambda. IAM est un service AWS que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- [Public ciblé \(p. 451\)](#)
- [Authentification avec des identités \(p. 451\)](#)
- [Gestion de l'accès à l'aide de stratégies \(p. 453\)](#)
- [Fonctionnement de AWS Lambda avec IAM \(p. 455\)](#)
- [Exemples de stratégies AWS Lambda basées sur l'identité \(p. 455\)](#)
- [Résolution des problèmes d'identité et d'accès avec AWS Lambda \(p. 457\)](#)

Public ciblé

Votre utilisation d'AWS Identity and Access Management (IAM) évolue selon la tâche que vous réalisez dans Lambda.

Utilisateur du service – Si vous utilisez le service Lambda pour effectuer votre tâche, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Plus vous utiliserez de fonctionnalités Lambda pour effectuer votre travail, plus vous pourrez avoir besoin d'autorisations supplémentaires. Comprendre la gestion des accès peut vous aider à demander à votre administrateur les autorisations appropriées. Si vous ne pouvez pas accéder à une fonctionnalité dans Lambda, consultez [Résolution des problèmes d'identité et d'accès avec AWS Lambda \(p. 457\)](#).

Administrateur du service – Si vous êtes le responsable des ressources Lambda de votre entreprise, vous bénéficiez probablement d'un accès total à Lambda. C'est à vous de déterminer les fonctionnalités et les ressources Lambda auxquelles vos employés pourront accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec Lambda, consultez [Fonctionnement de AWS Lambda avec IAM \(p. 455\)](#).

Administrateur IAM – Si vous êtes un administrateur IAM, vous souhaiterez peut-être obtenir des détails sur la façon dont vous pouvez écrire des stratégies pour gérer l'accès à Lambda. Pour obtenir des exemples de stratégies Lambda basées sur l'identité que vous pouvez utiliser dans IAM, consultez [Exemples de stratégies AWS Lambda basées sur l'identité \(p. 455\)](#).

Authentification avec des identités

L'authentification correspond au processus par lequel vous vous connectez à AWS via vos informations d'identification. Pour plus d'informations sur la signature à l'aide d'AWS Management Console, consultez [La console et la page de connexion IAM](#) dans le IAM Guide de l'utilisateur.

Vous devez être authentifié (connecté à AWS) en tant que Utilisateur racine d'un compte AWS ou utilisateur IAM, ou en assumant un rôle IAM. Vous pouvez également utiliser l'authentification de connexion unique de votre entreprise ou vous connecter via Google ou Facebook. Dans ce cas, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS avec des informations d'identification d'une autre entreprise, vous assumez indirectement un rôle.

Pour vous connecter directement à la [AWS Management Console](#), utilisez votre mot de passe avec votre e-mail utilisateur racine ou votre nom d'utilisateur IAM. Vous pouvez accéder à AWS par programmation avec vos clés d'accès utilisateur racine ou IAM. AWS fournit un kit de développement logiciel (SDK) et des outils de ligne de commande pour signer de manière cryptographique votre requête avec vos informations d'identification. Si vous n'utilisez pas les outils AWS, vous devez signer la demande vous-même. Pour ce faire, utilisez Signature Version 4, un protocole permettant d'authentifier les demandes d'API entrantes. Pour en savoir plus sur l'authentification des demandes, consultez [Processus de signature Signature Version 4](#) dans la documentation AWS General Reference.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être également fournir des informations de sécurité supplémentaires. Par exemple, AWS vous recommande d'utiliser l'authentification multi-facteurs (MFA) pour améliorer la sécurité de votre compte. Pour en savoir plus, consultez [Utilisation de Multi-Factor Authentication \(MFA\) dans AWS](#) dans le IAM Guide de l'utilisateur.

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 la 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 de l'utilisateur racine et utilisez-les pour effectuer uniquement certaines tâches de gestion des comptes et des services.

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité dans votre compte AWS qui dispose d'autorisations spécifiques pour une seule personne ou application. Un utilisateur IAM peut disposer d'informations d'identification à long terme comme un nom d'utilisateur et un mot de passe ou un ensemble de clés d'accès. Pour savoir comment générer des clés d'accès, consultez [Gestion des clés d'accès pour les utilisateurs IAM](#) dans le IAM Guide de l'utilisateur. Lorsque vous générez des clés d'accès pour un utilisateur IAM, veillez à afficher et enregistrer la paire de clés de manière sécurisée. Vous ne pourrez plus récupérer la clé d'accès secrète à l'avenir. Au lieu de cela, vous devrez générer une nouvelle paire de clés d'accès.

Un [groupe IAM](#) est une identité qui spécifie un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé Admins IAM et accorder à ce groupe les autorisations leur permettant d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour plus de détails, consultez [Quand créer un utilisateur IAM \(au lieu d'un rôle\)](#) dans le IAM Guide de l'utilisateur.

Rôles IAM

Un [rôle IAM](#) est une entité au sein de votre compte AWS qui dispose d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais un rôle n'est pas associé à une personne en particulier.

Vous pouvez temporairement endosser un rôle IAM dans l'AWS Management Console grâce au [changement de rôle](#). Vous pouvez obtenir un rôle en appelant une opération d'API AWS CLI ou AWS à l'aide d'une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez [Utilisation de rôles IAM dans le IAM Guide de l'utilisateur](#).

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Autorisations utilisateur IAM temporaires – Un utilisateur IAM peut endosser un rôle IAM pour accepter différentes autorisations temporaires concernant une tâche spécifique.
- Accès d'utilisateurs fédérés – Au lieu de créer un utilisateur IAM, vous pouvez utiliser des identités d'utilisateur préexistantes provenant d'AWS Directory Service, de l'annuaire d'utilisateurs de votre entreprise ou d'un fournisseur d'identité web. On parle alors d'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 IAM Guide de l'utilisateur.
- Accès entre comptes – Vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (mandataire de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès entre plusieurs comptes. Toutefois, certains services AWS vous permettent d'attacher une stratégie directement à une ressource (au lieu d'utiliser un rôle en tant que proxy). Pour en savoir plus sur la différence entre les rôles et les stratégies basées sur les ressources pour l'accès entre comptes, consultez [Différence entre les rôles IAM et les stratégies basées sur les ressources](#) dans le IAM Guide de l'utilisateur.
- Accès à un service AWS –Un rôle de service est un rôle IAM qu'un service assume pour effectuer des actions dans votre compte en votre nom. Lorsque vous configurez certains environnements de services AWS, vous devez définir un rôle que ce service devra assumer. Ce rôle de service doit comprendre toutes les autorisations nécessaires pour que le service puisse accéder aux ressources AWS dont il a besoin. Les rôles de service varient d'un service à un service, mais nombre d'entre eux vous permettent de choisir vos autorisations, tant que vous respectez les exigences documentées pour le service en question. Les rôles de service fournissent un accès uniquement au sein de votre compte et ne peuvent pas être utilisés pour accorder l'accès à des services dans d'autres comptes. Vous créez, modifiez et supprimez un rôle de service à partir d'IAM. 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 ce compartiment dans un cluster Amazon Redshift. Pour plus d'informations, consultez [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 CLI ou 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.

Pour savoir si vous devez utiliser ces rôles IAM ou non, consultez [Quand créer un rôle IAM \(au lieu d'un utilisateur\)](#) dans le IAM Guide de l'utilisateur.

Gestion de l'accès à l'aide de stratégies

Vous contrôlez les accès dans AWS en créant des stratégies et en les attachant à des identités IAM ou à des ressources AWS. Une stratégie est un objet dans AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit les autorisations de ces dernières. AWS évalue ces stratégies lorsqu'une entité (utilisateur racine, utilisateur IAM ou rôle IAM) envoie une demande. Les autorisations dans les stratégies déterminent si la demande est autorisée ou refusée. La plupart des stratégies sont stockées dans AWS en tant que documents JSON. Pour plus d'informations sur la structure et le contenu du document de stratégie JSON, consultez [Présentation des stratégies JSON](#) dans le IAM Guide de l'utilisateur.

Les stratégies permettent à un administrateur IAM de spécifier qui a accès aux ressources AWS et quelles actions ces personnes peuvent exécuter sur ces ressources. Chaque entité IAM (utilisateur ou rôle) démarre sans autorisation. En d'autres termes, par défaut, les utilisateurs ne peuvent rien faire, pas même changer leurs propres mots de passe. Pour autoriser un utilisateur à effectuer une opération, un administrateur doit associer une stratégie d'autorisations à ce dernier. Il peut également ajouter l'utilisateur à un groupe disposant des autorisations prévues. Lorsqu'un administrateur accorde des autorisations à un groupe, tous les utilisateurs de ce groupe se voient octroyer ces autorisations.

Les stratégies IAM définissent les autorisations d'une action quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposez d'une stratégie qui autorise l'action `iam:GetRole`. Un utilisateur avec cette stratégie peut obtenir des informations utilisateur à partir de l'AWS Management Console, de l'AWS CLI ou de l'API AWS.

Stratégies basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un rôle ou un groupe IAM. Ces stratégies contrôlent les actions que peut exécuter cette identité, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une stratégie basée sur l'identité, consultez [Création de stratégies IAM](#) dans le IAM Guide de l'utilisateur.

Les stratégies basées sur l'identité peuvent être classées comme étant des stratégies en ligne ou des stratégies gérées. Les stratégies en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les stratégies gérées sont des stratégies autonomes que vous pouvez lier à plusieurs utilisateurs, groupes et rôles de votre compte AWS. Les stratégies gérées incluent les stratégies gérées par AWS et les stratégies gérées par le client. Pour découvrir comment choisir entre une politique gérée ou une politique en ligne, consultez [Choix entre les stratégies gérées et les stratégies en ligne](#) dans le IAM Guide de l'utilisateur.

Stratégies basées sur les ressources

Les stratégies basées sur les ressources sont des documents de stratégie JSON que vous attachez à une ressource, telle qu'un compartiment Amazon S3. Les administrateurs de service peuvent utiliser ces stratégies pour définir les actions qu'un principal (membre de compte, utilisateur ou rôle) spécifié peut effectuer sur cette ressource et dans quelles conditions. Les stratégies basées sur les ressources sont des stratégies en ligne. Il ne s'agit pas de stratégies gérées basées sur les ressources.

Listes de contrôle d'accès (ACL)

Les listes de contrôle d'accès (ACL) constituent un type de stratégie permettant de définir les mandataires (membres de compte, utilisateurs ou rôles) ayant l'autorisation d'accéder à une ressource. Les listes de contrôle d'accès sont semblables aux stratégies basées sur les ressources, bien qu'elles n'utilisent pas le format de document de stratégie JSON. Amazon S3, AWS WAF et Amazon VPC sont des exemples de services prenant en charge les listes de contrôle d'accès. Pour en savoir plus sur les listes de contrôle d'accès, consultez [Présentation de la liste de contrôle d'accès \(ACL\)](#) dans le Manuel du développeur Amazon Simple Storage Service.

Autres types de stratégie

AWS prend en charge d'autres types de stratégies moins courantes. Ces types de stratégies peuvent définir le nombre maximal d'autorisations qui vous sont accordées par des types de stratégies plus courants.

- Limite d'autorisations – Une limite d'autorisations est une fonctionnalité avancée dans laquelle vous définissez les autorisations maximales qu'une stratégie basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations obtenues représentent la combinaison des stratégies basées sur l'identité de l'entité et de ses limites d'autorisations. Les stratégies basées sur les ressources qui spécifient l'utilisateur ou le rôle

dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces stratégies remplace l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez [Limites d'autorisations pour des entités IAM](#) dans le IAM Guide de l'utilisateur.

- Stratégies de contrôle de service (SCP) – Les SCP sont des stratégies JSON qui spécifient le nombre maximal d'autorisations pour une organisation ou une unité d'organisation (OU) dans AWS Organizations. AWS Organizations est un service qui vous permet de regrouper et de gérer de façon centralisée plusieurs comptes AWS détenus par votre entreprise. Si vous activez toutes les fonctions d'une organisation, vous pouvez appliquer les stratégies de contrôle de service (SCP) à l'un ou à l'ensemble de vos comptes. La SCP limite les autorisations pour les entités dans les comptes membres, y compris dans chaque Utilisateur racine d'un compte AWS. Pour plus d'informations sur les Organisations et les SCP, consultez [Fonctionnement des stratégies de contrôle de service](#) dans le Manuel de l'utilisateur AWS Organizations.
- Stratégies de session – Les stratégies de session sont des stratégies avancées que vous transmettez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de la session obtenue sont une combinaison des stratégies basées sur l'identité de l'utilisateur ou du rôle et des stratégies de session. Les autorisations peuvent également provenir d'une stratégie basée sur les ressources. Un refus explicite dans l'une de ces stratégies remplace l'autorisation. Pour de plus amples informations, veuillez consulter [Stratégies de session](#) dans le IAM Guide de l'utilisateur.

Plusieurs types de stratégie

Lorsque plusieurs types de stratégies s'appliquent à la requête, les autorisations obtenues sont plus compliquées à comprendre. Pour découvrir la façon dont AWS détermine s'il convient d'autoriser une demande en présence de plusieurs types de stratégies, consultez [Logique d'évaluation de stratégies](#) dans le IAM Guide de l'utilisateur.

Fonctionnement de AWS Lambda avec IAM

Avant d'utiliser IAM pour gérer l'accès à Lambda, vous devez comprendre quelles sont les fonctions IAM disponibles à utiliser avec Lambda. Pour obtenir une vue d'ensemble de la façon dont Lambda et d'autres services AWS fonctionnent avec IAM, consultez [Services AWS qui fonctionnent avec IAM](#) dans le IAM Guide de l'utilisateur.

Pour une vue d'ensemble des autorisations, des stratégies et des rôles tels qu'ils sont utilisés par Lambda, consultez [Autorisations AWS Lambda \(p. 32\)](#).

Exemples de stratégies AWS Lambda basées sur l'identité

Par défaut, les utilisateurs et les rôles IAM ne sont pas autorisés à créer ou modifier les ressources Lambda. Ils ne peuvent pas non plus exécuter des tâches à l'aide de AWS Management Console, AWS CLI ou de l'API AWS. Un administrateur IAM doit créer des stratégies IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées dont ils ont besoin. Il doit ensuite attacher ces stratégies aux utilisateurs ou aux groupes IAM ayant besoin de ces autorisations.

Pour apprendre à créer une stratégie basée sur l'identité IAM à l'aide de ces exemples de documents de stratégie JSON, veuillez consulter [Création de stratégies dans l'onglet JSON](#) dans le IAM Guide de l'utilisateur.

Rubriques

- [Bonne pratique en matière de stratégies \(p. 456\)](#)
- [Utilisation de la console Lambda \(p. 456\)](#)
- [Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations \(p. 456\)](#)

Bonnes pratiques en matière de stratégies

Les stratégies basées sur l'identité sont très puissantes. Elles déterminent si une personne peut créer, consulter ou supprimer des ressources Lambda dans votre compte. Ces actions peuvent entraîner des frais pour votre compte AWS. Lorsque vous créez ou modifiez des stratégies basées sur l'identité, suivez ces instructions et recommandations :

- Commencer à utiliser des stratégies gérées AWS – Pour commencer à utiliser Lambda rapidement, utilisez les politiques gérées AWS pour accorder à vos employés les autorisations dont ils ont besoin. Ces stratégies sont déjà disponibles dans votre compte et sont gérées et mises à jour par AWS. Pour plus d'informations, consultez la section [Mise en route avec les autorisations à l'aide des stratégies gérées AWS](#) dans le IAM Guide de l'utilisateur.
- Accorder le privilège le plus faible – Lorsque vous créez des stratégies personnalisées, accordez uniquement les autorisations requises pour exécuter une seule tâche. Commencez avec un minimum d'autorisations et accordez-en d'autres si nécessaire. Cette méthode est plus sûre que de commencer avec des autorisations trop permissives et d'essayer de les restreindre plus tard. Pour plus d'informations, consultez [Accorder le privilège le plus faible](#) dans le IAM Guide de l'utilisateur.
- Activer MFA pour les opérations sensibles – Pour plus de sécurité, obligez les utilisateurs IAM à utiliser l'authentification multi-facteurs (MFA) pour accéder à des ressources ou à des opérations d'API sensibles. Pour plus d'informations, consultez [Utilisation de Multi-Factor Authentication \(MFA\) dans AWS](#) dans le IAM Guide de l'utilisateur.
- Utiliser des conditions de stratégie pour une plus grande sécurité – Tant que cela reste pratique pour vous, définissez les conditions dans lesquelles vos stratégies basées sur l'identité autorisent l'accès à une ressource. Par exemple, vous pouvez rédiger les conditions pour spécifier une plage d'adresses IP autorisées d'où peut provenir une demande. Vous pouvez également écrire des conditions pour autoriser les requêtes uniquement à une date ou dans une plage de temps spécifiée, ou pour imposer l'utilisation de SSL ou de MFA. Pour plus d'informations, consultez [Éléments de stratégie JSON IAM : condition](#) dans le IAM Guide de l'utilisateur.

Utilisation de la console Lambda

Pour accéder à la console AWS Lambda, vous devez disposer d'un ensemble minimum d'autorisations. Ces autorisations doivent vous permettre de répertorier et de consulter les informations relatives aux ressources Lambda de votre compte AWS. Si vous créez une stratégie basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs et rôles IAM) tributaires de cette stratégie.

Pour obtenir un exemple de stratégie qui accorde un accès minimal pour le développement de fonctions, veuillez consulter [Développement des fonctions \(p. 41\)](#). En plus des API Lambda, la console Lambda utilise d'autres services pour afficher la configuration des déclencheurs et vous permettre d'ajouter de nouveaux déclencheurs. Si vos utilisateurs utilisent Lambda avec d'autres services, ils ont également besoin d'accéder à ces services. Pour plus d'informations sur la configuration d'autres services Lambda, reportez-vous à la section [Utilisation de AWS Lambda avec d'autres services \(p. 155\)](#).

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une stratégie qui permet aux utilisateurs IAM d'afficher les stratégies en ligne et gérées attachées à leur identité d'utilisateur. Cette stratégie inclut les autorisations nécessaires pour réaliser cette action sur la console ou par programmation à l'aide de l'AWS CLI ou de l'API AWS.

```
{  
    "Version": "2012-10-17",  
    "Statement": [
```

```
{  
    "Sid": "ViewOwnUserInfo",  
    "Effect": "Allow",  
    "Action": [  
        "iam:GetUserPolicy",  
        "iam>ListGroupsForUser",  
        "iam>ListAttachedUserPolicies",  
        "iam>ListUserPolicies",  
        "iam GetUser"  
    ],  
    "Resource": ["arn:aws:iam::*:user/${aws:username}"]  
},  
{  
    "Sid": "NavigateInConsole",  
    "Effect": "Allow",  
    "Action": [  

```

Résolution des problèmes d'identité et d'accès avec AWS Lambda

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec Lambda et IAM.

Rubriques

- [Je ne suis pas autorisé à effectuer une action dans Lambda \(p. 457\)](#)
- [Je ne suis pas autorisé à exécuter : iam:PassRole \(p. 458\)](#)
- [Je veux afficher mes clés d'accès \(p. 458\)](#)
- [Je suis un administrateur et je souhaite autoriser d'autres utilisateurs à accéder à Lambda \(p. 458\)](#)
- [Je souhaite permettre à des personnes extérieures à mon compte AWS d'accéder à mes ressources Lambda \(p. 459\)](#)

Je ne suis pas autorisé à effectuer une action dans Lambda

Si AWS Management Console indique que vous n'êtes pas autorisé à exécuter une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe.

L'exemple d'erreur suivant se produit lorsque l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées concernant une fonction mais ne dispose pas des autorisations `lambda:GetFunction` nécessaires.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
lambda:GetFunction on resource: my-function
```

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses stratégies pour lui permettre d'accéder à la ressource `my-function` à l'aide de l'action `lambda:GetFunction`.

Je ne suis pas autorisé à exécuter : `iam:PassRole`

Si vous recevez un message d'erreur selon lequel vous n'êtes pas autorisé à exécuter l'action `iam:PassRole`, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe. Demandez à cette personne de mettre à jour vos stratégies pour vous permettre de transmettre un rôle à Lambda.

Certains services AWS vous permettent de transmettre un rôle existant à ce service, au lieu de créer un nouveau rôle de service ou rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans Lambda. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

Dans ce cas, Mary demande à son administrateur de mettre à jour ses stratégies pour lui permettre d'exécuter l'action `iam:PassRole`.

Je veux afficher mes clés d'accès

Une fois que vous avez créé vos clés d'accès utilisateur IAM, vous pouvez afficher votre ID de clé d'accès à tout moment. Toutefois, vous ne pouvez pas afficher à nouveau votre clé d'accès secrète. Si vous perdez votre clé d'accès secrète, vous devez créer une nouvelle paire de clés.

Les clés d'accès se composent de deux parties : un ID de clé d'accès (par exemple, `AKIAIOSFODNN7EXAMPLE`) et une clé d'accès secrète (par exemple, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). À l'instar d'un nom d'utilisateur et un mot de passe, vous devez utiliser à la fois l'ID de clé d'accès et la clé d'accès secrète pour authentifier vos demandes. Gérez vos clés d'accès de manière aussi sécurisée que votre nom d'utilisateur et votre mot de passe.

Important

Ne communiquez pas vos clés d'accès à un tiers, même pour qu'il vous aide à [trouver votre ID utilisateur canonique](#). En effet, vous lui accorderiez ainsi un accès permanent à votre compte.

Lorsque vous créez une paire de clé d'accès, enregistrez l'ID de clé d'accès et la clé d'accès secrète dans un emplacement sécurisé. La clé d'accès secrète est accessible uniquement au moment de sa création. Si vous perdez votre clé d'accès secrète, vous devez ajouter de nouvelles clés d'accès pour votre utilisateur IAM. Vous pouvez avoir un maximum de deux clés d'accès. Si vous en avez déjà deux, vous devez supprimer une paire de clés avant d'en créer une nouvelle. Pour afficher les instructions, consultez [Gestion des clés d'accès](#) dans le IAM Guide de l'utilisateur.

Je suis un administrateur et je souhaite autoriser d'autres utilisateurs à accéder à Lambda

Pour permettre à d'autres utilisateurs d'accéder à Lambda, vous devez créer une entité IAM (utilisateur ou rôle) pour la personne ou l'application qui a besoin de l'accès. Ils utiliseront les informations d'identification de cette entité pour accéder à AWS. Vous devez ensuite associer une stratégie à l'entité qui leur accorde les autorisations appropriées dans Lambda.

Pour démarrer immédiatement, consultez [Création de votre premier groupe et utilisateur délégué IAM](#) dans le IAM Guide de l'utilisateur.

Je souhaite permettre à des personnes extérieures à mon compte AWS d'accéder à mes ressources Lambda

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation peuvent utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est approuvé pour assumer le rôle. Pour les services qui prennent en charge les stratégies basées sur les ressources ou les listes de contrôle d'accès (ACL), vous pouvez utiliser ces stratégies pour accorder aux personnes l'accès à vos ressources.

Pour en savoir plus, consultez les éléments suivants :

- Pour savoir si Lambda prend en charge ces fonctionnalités, consultez [Fonctionnement de AWS Lambda avec IAM \(p. 455\)](#).
- Pour savoir comment fournir un accès à vos ressources sur les comptes AWS que vous détenez, consultez [Octroi à un utilisateur IAM de l'autorisation d'accès à un autre compte AWS vous appartenant](#) dans le IAM Guide de l'utilisateur.
- Pour savoir comment fournir l'accès à vos ressources à des comptes AWS tiers, consultez [Octroi d'un accès à des comptes AWS appartenant à des tiers](#) dans le IAM Guide de l'utilisateur.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez [Octroi d'accès à des utilisateurs authentifiés en externe \(fédération d'identité\)](#) dans le IAM Guide de l'utilisateur.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des stratégies basées sur les ressources pour l'accès entre comptes, consultez [Différence entre les rôles IAM et les stratégies basées sur les ressources](#) dans le IAM Guide de l'utilisateur.

Validation de la conformité pour AWS Lambda

Les auditeurs tiers évaluent la sécurité et la conformité de AWS Lambda dans le cadre de plusieurs programmes de conformité AWS. Il s'agit notamment des certifications SOC, PCI, FedRAMP, HIPAA et autres.

Pour obtenir la liste des services AWS associés à des programmes de conformité spécifiques, consultez [Services AWS concernés par le programme de conformité](#). Pour obtenir des informations générales, veuillez consulter [Programmes de conformité AWS](#).

Vous pouvez télécharger les rapports de l'audit externe avec AWS Artifact. Pour plus d'informations, consultez [Téléchargement des rapports dans AWS Artifact](#).

Votre responsabilité de conformité lors de l'utilisation de Lambda est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise, ainsi que la législation et la réglementation applicables. AWS fournit les ressources suivantes pour faciliter le respect de la conformité :

- [Guides de démarrage rapide de la sécurité et de la conformité](#) – Ces guides de déploiement proposent des considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de référence centrés sur la sécurité et la conformité dans AWS.
- [Livre blanc sur l'architecture pour la sécurité et la conformité HIPAA](#) – Le livre blanc décrit comment les entreprises peuvent utiliser AWS pour créer des applications conformes à la loi HIPAA.
- [Ressources de conformité AWS](#) – Cet ensemble de manuels et de guides peut s'appliquer à votre secteur et à votre emplacement.
- [AWS Config](#) – Ce service AWS permet d'évaluer le degré de conformité de vos configurations de ressources par rapport aux pratiques internes, aux normes et aux directives industrielles.
- [AWS Security Hub](#) – Ce service AWS fournit une vue complète de votre état de sécurité au sein d'AWS qui vous permet de vérifier votre conformité aux normes du secteur et aux bonnes pratiques de sécurité.

Résilience dans AWS Lambda

L'infrastructure mondiale dAWS repose sur des régions et des zones de disponibilité AWS. Les régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à débit élevé et à forte redondance. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur les régions et les zones de disponibilité AWS, consultez [Infrastructure mondiale AWS](#).

Outre l'infrastructure globale AWS, Lambda propose plusieurs fonctionnalités qui contribuent à la prise en charge des vos besoins en matière de résilience et de sauvegarde de données.

- Gestion des versions – Vous pouvez utiliser la gestion des versions dans Lambda pour enregistrer le code de votre fonction et la configuration que vous développez. Avec les alias, vous pouvez utiliser la gestion des versions pour effectuer des déploiements bleu/vert et de roulement. Pour plus d'informations, consultez [Versions de fonction AWS Lambda \(p. 70\)](#).
- Mise à l'échelle – Lorsque votre fonction reçoit une demande tandis qu'elle traite une demande précédente, Lambda lance une autre instance de votre fonction pour traiter la charge accrue. Lambda met automatiquement à l'échelle pour traiter 1,000 exécutions simultanées par région, une [limite \(p. 30\)](#) qui peut être augmentée si nécessaire. Pour plus d'informations, veuillez consulter [Dimensionnement d'une fonction AWS Lambda \(p. 106\)](#).
- Haute disponibilité – Lambda exécute votre fonction dans plusieurs zones de disponibilité afin de vous assurer qu'elle est disponible pour traiter les événements en cas d'interruption de service dans une seule zone. Si vous configurez votre fonction pour vous connecter à un cloud privé virtuel (VPC) de votre compte, spécifiez les sous-réseaux dans plusieurs zones de disponibilité, pour une garantie de haute disponibilité. Pour plus d'informations, consultez [Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC \(p. 81\)](#).
- Simultanéité réservée – Pour vous assurer que votre fonction peut toujours mettre à l'échelle afin de répondre à des demandes supplémentaires, vous pouvez réserver la simultanéité. La définition de la simultanéité réservée pour une fonction permet de s'assurer qu'elle peut être mise à l'échelle, sans le dépasser, un certain nombre spécifié d'appels simultanés. Cela garantit que vous ne perdez pas les demandes en raison d'autres fonctions utilisant toute la simultanéité disponible. Pour plus d'informations, consultez [Gestion de la simultanéité pour une fonction Lambda \(p. 61\)](#).
- Nouvelles tentatives – Pour les appels asynchrones et un sous-ensemble d'appels déclenchés par d'autres services, Lambda retente automatiquement en cas cas d'erreur avec retards entre les tentatives. Les autres clients et les services AWS qui appellent des fonctions de manière synchrone sont responsables de l'exécution des nouvelles tentatives. Pour plus d'informations, veuillez consulter [Gestion des erreurs et tentatives automatiques dans AWS Lambda \(p. 111\)](#).
- Files d'attente de lettres mortes – Pour les appels asynchrones, vous pouvez configurer Lambda pour envoyer des demandes à une file d'attente de lettres mortes si toutes les tentatives échouent. Une file d'attente de lettres mortes est une rubrique Amazon SNS ou une file d'attente Amazon SQS qui reçoit les événements pour dépannage ou retraitement. Pour plus d'informations, consultez [Files d'attente de lettres mortes de fonction AWS Lambda \(p. 99\)](#).

Sécurité de l'infrastructure dans AWS Lambda

En tant que service géré, AWS Lambda est protégé par les procédures de sécurité du réseau mondial AWS qui sont décrites dans le livre blanc [Amazon Web Services : Présentation des procédures de sécurité](#).

Vous utilisez les appels d'API publiés AWS pour accéder à Lambda via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.0 ou version ultérieure. Nous recommandons TLS 1.2 ou version ultérieure. Les clients doivent également prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La plupart des systèmes modernes telles que Java 7 et versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un mandataire IAM. Vous pouvez également utiliser [AWS Security Token Service](#) (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Configuration et analyse des vulnérabilités dans AWS Lambda

AWS Lambda fournit des [runtimes](#) (p. 122) qui exécutent le code de votre fonction dans un environnement d'exécution basé sur Amazon Linux. Lambda est responsable du maintien à jour des logiciels du runtime et de l'environnement d'exécution, de la publication des nouveaux runtime pour les nouveaux langages et les nouvelles infrastructures, et pour rendre obsolètes les runtimes lorsque le logiciel sous-jacent n'est plus pris en charge.

Si vous utilisez des bibliothèques supplémentaires avec votre fonction, vous êtes responsable de la mise à jour des bibliothèques. Vous pouvez inclure des bibliothèques supplémentaires dans le [package de déploiement](#) (p. 21), ou dans les [couches](#) (p. 76) que vous attachez à votre fonction. Vous pouvez également créer des [runtimes personnalisés](#) (p. 126) et utiliser des couches pour les partager avec d'autres comptes.

Lambda déclare obsolètes les runtimes lorsque le logiciel du runtime ou de son environnement d'exécution atteint sa fin de vie. Lorsque Lambda rend obsolète un runtime, vous êtes responsable de la migration de vos fonctions vers un runtime pris en charge pour le même langage ou la même infrastructure. Pour plus d'informations, consultez [Stratégie de prise en charge des environnements d'exécution](#) (p. 124).

Résolution des problèmes dans AWS Lambda

Les rubriques suivantes fournissent des conseils de dépannage pour les erreurs et les problèmes que vous pouvez rencontrer lors de l'utilisation de l'API Lambda, de la console ou des outils. Si vous rencontrez un problème qui n'est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Pour plus de conseils de dépannage et de réponses aux questions courantes de support, visitez le [Centre de connaissances AWS](#).

Rubriques

- [Résolution des problèmes de déploiement dans AWS Lambda \(p. 462\)](#)
- [Résolution des problèmes d'appel dans AWS Lambda \(p. 464\)](#)
- [Résolution des problèmes d'exécution dans AWS Lambda \(p. 466\)](#)
- [Résolution des problèmes de mise en réseau dans AWS Lambda \(p. 467\)](#)

Résolution des problèmes de déploiement dans AWS Lambda

Lorsque vous mettez à jour votre fonction, Lambda déploie la modification en lançant de nouvelles instances de cette fonction avec le code ou les paramètres mis à jour. Les erreurs de déploiement empêchent l'utilisation de la nouvelle version et peuvent résulter de problèmes liés au package de déploiement, au code, à vos autorisations ou à vos outils.

Lorsque vous déployez des mises à jour de votre fonction directement avec l'API Lambda ou avec un client tel que l'AWS CLI, les erreurs Lambda sont directement visibles dans la sortie. Si vous utilisez des services comme AWS CloudFormation, AWS CodeDeploy ou AWS CodePipeline, recherchez la réponse de Lambda dans les journaux ou le flux d'événements de ce service.

Erreur : EACCES : autorisation refusée, ouvrir '/var/task/index.js'

Erreur : impossible de charger ce fichier - cette fonction

Erreur : [Errno 13] Autorisation refusée : '/var/task/function.py'

L'environnement d'exécution Lambda a besoin d'une autorisation pour lire les fichiers de votre package de déploiement. Vous pouvez utiliser la commande `chmod` pour modifier le mode fichier. Grâce aux exemples de commandes suivants, tous les fichiers et dossiers du répertoire actuel sont lisibles par n'importe quel utilisateur.

```
my-function$ chmod 644 $(find . -type f)
my-function$ chmod 755 $(find . -type d)
```

Erreur : une erreur s'est produite (RequestEntityTooLargeException) lors de l'appel de l'opération `UpdateFunctionCode`

Lorsque vous chargez un package de déploiement ou une archive de couche directement dans Lambda, la taille du fichier ZIP est limitée à 50 Mo. Pour charger un fichier plus volumineux, stockez-le dans Amazon S3 et utilisez les paramètres [S3Bucket et S3Key \(p. 637\)](#).

Note

Lorsque vous chargez un fichier directement avec l'AWS CLI, le kit AWS SDK ou autre, le fichier ZIP binaire est converti en base64, ce qui augmente sa taille d'environ 30 %. Pour prendre cela en compte, et de la taille des autres paramètres de la demande, la limite de taille réelle de la demande appliquée par Lambda est supérieure. Pour cette raison, la limite de 50 Mo est approximative.

Erreur : une erreur s'est produite pendant GetObject. Code d'erreur S3 : PermanentRedirect. Message d'erreur S3 : Le compartiment se trouve dans cette région : us-east-2. Veuillez utiliser cette région pour renouveler la demande

Lorsque vous téléchargez le package de déploiement d'une fonction à partir d'un compartiment Amazon S3, le compartiment doit se trouver dans la même région que la fonction. Ce problème peut se produire lorsque vous spécifiez un objet Amazon S3 dans un appel à [UpdateFunctionCode \(p. 636\)](#), ou que vous utilisez le package et déployez des commandes dans l'interface de ligne de commande AWS ou dans l'interface de ligne de commande AWS SAM. Créez un compartiment d'artefact de déploiement pour chaque région dans laquelle vous développez des applications.

Erreur : module 'function' introuvable

Erreur : impossible de charger ce fichier - cette fonction

Erreur : impossible d'importer le module 'function'

Erreur : Classe introuvable : Function.handler

Erreur : fork/exec /var/task/function : aucun fichier ou répertoire de ce type

Erreur : impossible de charger le type 'Function.Handler' à partir de l'assemblage 'Function'.

Le nom du fichier ou de la classe dans la configuration du gestionnaire de votre fonction ne correspond pas à votre code. Reportez-vous à l'entrée suivante pour plus d'informations.

Erreur : index.handler n'est pas défini ou n'est pas exporté

Erreur : gestionnaire 'handler' manquant dans le module 'function'

Erreur : méthode non définie 'handler' pour #<lambdaHandler:0x000055B76CceBF98>

Erreur : aucune méthode publique nommée handleRequest avec la signature de méthode appropriée trouvée dans la classe class function.Handler

Erreur : méthode 'handleRequest' introuvable dans le type 'Function.Handler' de l'assemblage 'Function'

Le nom de la méthode du gestionnaire dans la configuration du gestionnaire de votre fonction ne correspond pas à votre code. Chaque exécution définit une convention d'affectation de noms pour les gestionnaires, tels que le **nom de fichier.nom de la méthode**. Le gestionnaire correspond à la méthode dans le code de votre fonction que l'environnement d'exécution exécute lorsque la fonction est appelée.

Pour certaines langues, Lambda fournit une bibliothèque avec une interface pour laquelle une méthode de gestionnaire doit avoir un nom spécifique. Pour plus d'informations sur l'attribution de noms de gestionnaire pour chaque langue, consultez les rubriques suivantes.

- [Création de fonctions Lambda avec Node.js \(p. 312\)](#)

- [Création de fonctions Lambda avec Python \(p. 329\)](#)
- [Création de fonctions Lambda avec Ruby \(p. 347\)](#)
- [Création de fonctions Lambda avec Java \(p. 363\)](#)
- [Création de fonctions Lambda avec Go \(p. 398\)](#)
- [Création de fonctions Lambda avec C# \(p. 413\)](#)
- [Création de fonctions Lambda avec PowerShell \(p. 435\)](#)

Erreur : `InvalidParameterValueException` : Lambda n'a pas pu configurer vos variables d'environnement car les variables d'environnement que vous avez fournies dépassaient la limite de 4 Ko. Chaîne mesurée : `{"A1": "uSFeY5cyPiPn7AtnX5BsM..."}`

Erreur : `RequestEntityToolargeException` : La demande doit être inférieure à 5 120 octets pour l'opération `UpdateFunctionConfiguration`

La taille maximale de l'objet variable stocké dans la configuration de la fonction ne doit pas dépasser 4096 octets. Cela inclut les noms de clés, les valeurs, les guillemets, les virgules et les crochets. La taille totale du corps de requête HTTP est également limitée.

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "nodejs12.x",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Environment": {  
        "Variables": {  
            "BUCKET": "my-bucket",  
            "KEY": "file.txt"  
        }  
    },  
    ...  
}
```

Dans cet exemple, l'objet comporte 39 caractères et utilise 39 octets lorsqu'il est stocké (sans espace) comme chaîne `{"BUCKET": "my-bucket", "KEY": "file.txt"}`. Les caractères ASCII standard dans les valeurs des variables d'environnement utilisent un octet chacun. Les caractères ASCII et Unicode étendus peuvent utiliser entre 2 et 4 octets par caractère.

Erreur : `InvalidParameterValueException` : Lambda n'a pas pu configurer vos variables d'environnement car les variables d'environnement que vous avez fournies contiennent des clés réservées qui ne sont actuellement pas prises en charge pour la modification.

Lambda réserve certaines clés de variables d'environnement pour une utilisation interne. Par exemple, `AWS_REGION` est utilisé par l'environnement d'exécution pour déterminer la région actuelle et ne peut pas être remplacé. D'autres variables, comme `PATH`, sont utilisées par l'environnement d'exécution, mais peuvent être étendues dans la configuration de votre fonction. Pour obtenir une liste complète, veuillez consulter [Variables d'environnement d'exécution \(p. 57\)](#).

Résolution des problèmes d'appel dans AWS Lambda

Lorsque vous appelez une fonction Lambda, Lambda valide la demande et vérifie la capacité de dimensionnement avant d'envoyer l'événement à votre fonction ou, pour un appel asynchrone, à la file d'attente d'événements. Les erreurs d'appel peuvent être causées par des problèmes de paramètres de

demande, de structure d'événement, de paramètres de fonction, d'autorisations utilisateur, d'autorisations de ressources ou de limites.

Si vous appelez la fonction directement, les erreurs d'appel sont visibles dans la réponse d'Lambda. Si vousappelez la fonction de manière asynchrone avec un mappage de source d'événement ou via un autre service, vous pouvez trouver des erreurs dans les journaux, une file d'attente de lettres mortes ou une destination en cas d'échec. Les options de gestion des erreurs et le comportement des nouvelles tentatives varient en fonction de la façon dont vous appelez la fonction et du type d'erreur.

Pour obtenir la liste des types d'erreur qui peuvent être renvoyés par l'opération `Invoke`, consultez [Invoke \(p. 565\)](#).

Erreur : Utilisateur : arn : aws : iam::123456789012 : l'utilisateur/le développeur n'est pas autorisé à réaliser : lambda : InvokeFunction sur la ressource : my-function

Votre utilisateur IAM, ou le rôle que vous assumez, a besoin d'une autorisation pour appeler une fonction. Cette exigence s'applique également aux fonctions Lambda et autres ressources de calcul qui appellent des fonctions. Ajoutez la stratégie gérée AWSLambdaRole ou une stratégie personnalisée qui autorise l'action `lambda:InvokeFunction` sur la fonction cible, à votre utilisateur IAM.

Note

Contrairement aux autres actions d'API dans Lambda, le nom de l'action dans IAM (`lambda:InvokeFunction`) ne correspond pas au nom de l'action d'API (`Invoke`) pour appeler une fonction.

Pour de plus amples informations, veuillez consulter [Autorisations AWS Lambda \(p. 32\)](#).

Erreur :ResourceConflictException : l'opération ne peut pas être effectuée pour le moment. La fonction est actuellement dans l'état suivant : En attente

Lorsque vous connectez une fonction à un VPC au moment de la création, la fonction passe à l'état `Pending` pendant que Lambda crée des interfaces réseau Elastic. Pendant ce temps, vous ne pouvez pas appeler ni modifier votre fonction. Si vous connectez votre fonction à un VPC après la création, vous pouvez l'appeler pendant que la mise à jour est en attente, mais vous ne pouvez pas modifier son code ni sa configuration.

Pour de plus amples informations, veuillez consulter [Surveillance de l'état d'une fonction avec l'API Lambda \(p. 105\)](#).

Erreur : Une fonction est bloquée à l'état `Pending` depuis plusieurs minutes.

Si une fonction est bloquée à l'état `Pending` depuis plus de six minutes,appelez l'une des opérations d'API suivantes pour la débloquer.

- [UpdateFunctionCode \(p. 636\)](#)
- [UpdateFunctionConfiguration \(p. 643\)](#)
- [PublishVersion \(p. 601\)](#)

Lambda annule l'opération en attente et place la fonction à l'état `Failed`. Vous pouvez ensuite supprimer la fonction et la recréer, ou tenter une autre mise à jour.

Problème : une fonction utilise toute la simultanéité disponible, limitant ainsi d'autres fonctions.

Pour diviser la simultanéité disponible dans une région en groupes, utilisez la [simultanéité réservée \(p. 61\)](#). La simultanéité réservée garantit qu'une fonction s'adapte toujours à la simultanéité qui lui est assignée et, parallèlement, qu'elle n'utilise pas plus de simultanéité que ce qui lui est attribué.

Problème : vous pouvez appeler la fonction directement, mais elle ne s'exécute pas lorsqu'elle est appelée par un autre service ou compte.

Vous accordez à [d'autres services \(p. 155\)](#) et comptes l'autorisation d'appeler une fonction dans la [stratégie basée sur les ressources \(p. 36\)](#) de la fonction. Si l'appelant se trouve dans un autre compte, cet utilisateur a également besoin de [l'autorisation d'appeler des fonctions \(p. 41\)](#).

Problème : La fonction est invoquée en continu dans une boucle.

Cela se produit généralement lorsque votre fonction gère des ressources dans le même service AWS qui la déclenche. Par exemple, on peut créer une fonction qui stocke un objet dans un compartiment Amazon S3 configuré avec une [notification qui appelle à nouveau la fonction \(p. 262\)](#). Pour arrêter l'exécution de la fonction, choisissez Limitations dans la [page de configuration de la fonction \(p. 53\)](#). Identifiez ensuite le chemin de code ou l'erreur de configuration qui a provoqué l'invocation récursive.

Erreur : KMSDisabledException : Lambda n'a pas pu déchiffrer les variables d'environnement car la clé KMS utilisée est désactivée. Veuillez vérifier les paramètres de clé KMS de la fonction.

Cette erreur peut se produire si votre clé KMS est désactivée ou si l'octroi qui permet à Lambda d'utiliser la clé est révoqué. Si l'octroi est manquant, configurez la fonction pour utiliser une autre clé. Ensuite, réaffectez la clé personnalisée pour recréer l'octroi.

Résolution des problèmes d'exécution dans AWS Lambda

Lorsque le moteur d'exécution Lambda exécute le code de fonction, l'événement peut être traité sur une instance de la fonction qui traite déjà les événements depuis un certain temps. Sinon, l'initialisation d'une nouvelle instance peut être nécessaire. Des erreurs peuvent se produire lors de l'initialisation de la fonction, lorsque le code de gestionnaire traite l'événement ou lorsque la fonction renvoie (ou ne parvient pas à renvoyer) une réponse.

Les erreurs d'exécution de fonction peuvent être causées par des problèmes de code, de configuration de fonction, de ressources en aval ou d'autorisations. Si vous appelez la fonction directement, les erreurs de fonction sont visibles dans la réponse d'Lambda. Si vousappelez la fonction de manière asynchrone, avec un mappage de source d'événement ou via un autre service, vous pouvez trouver les erreurs dans les journaux, une file d'attente de lettres mortes ou une destination réservées aux échecs. Les options de gestion des erreurs et le comportement des nouvelles tentatives varient en fonction de la façon dont vousappelez la fonction et du type d'erreur.

Lorsque le code de fonction ou le moteur d'exécution Lambda renvoie une erreur, le code d'état indiqué dans la réponse d'Lambda est 200 OK. La présence d'une erreur dans la réponse est indiquée par un en-tête nommé `X-Amz-Function-Error`. Les codes d'état des séries 400 et 500 sont réservés aux [erreurs d'appel \(p. 464\)](#).

Problème : l'exécution de la fonction prend trop de temps.

Si l'exécution de votre code est beaucoup plus longue dans Lambda que sur votre ordinateur local, cela peut être dû à une limite au niveau de la mémoire ou de la puissance de traitement disponible pour la fonction. [Configurez la fonction avec de la mémoire supplémentaire \(p. 53\)](#) pour augmenter à la fois la mémoire et la capacité d'UC.

Problème : les journaux n'apparaissent pas dans CloudWatch Logs.

Problème : les traces n'apparaissent pas dans AWS X-Ray.

Votre fonction doit être autorisée à appeler CloudWatch Logs et X-Ray. Mettez à jour son [rôle d'exécution \(p. 33\)](#) pour lui accorder l'autorisation. Ajoutez les stratégies gérées suivantes pour activer les journaux et le suivi.

- AWSLambdaBasicExecutionRole

- AWSXRayDaemonWriteAccess

Lorsque vous ajoutez des autorisations à votre fonction, mettez également à jour son code ou sa configuration. Cela force l'arrêt et le remplacement des instances en cours d'exécution de votre fonction, qui ont des informations d'identification obsolètes.

Problème : (Node.js) la fonction renvoie un objet avant la fin de l'exécution du code

De nombreuses bibliothèques, y compris le kit SDK AWS, fonctionnent de manière asynchrone. Lorsque vous effectuez un appel réseau ou toute autre opération nécessitant l'attente d'une réponse, les bibliothèques renvoient un objet appelé promesse, qui suit la progression de l'opération en arrière-plan.

Pour attendre que la promesse soit résolue en réponse, utilisez le mot-clé `await`. Celui-ci empêche le code de gestionnaire de s'exécuter jusqu'à ce que la promesse soit résolue en objet contenant la réponse. Si vous n'avez pas besoin d'utiliser les données de la réponse dans votre code, vous pouvez retourner la promesse directement à l'environnement d'exécution.

Certaines bibliothèques ne retournent pas de promesses, mais peuvent être enveloppées dans du code qui le fait. Pour de plus amples informations, veuillez consulter [Gestionnaire de fonctions AWS Lambda dans Node.js \(p. 314\)](#).

Problème : le kit SDK AWS inclus dans le moteur d'exécution ne correspond pas à la dernière version

Problème : le kit SDK AWS inclus dans l'environnement d'exécution se met à jour automatiquement

Les environnements d'exécution pour les langages de script incluent le kit SDK AWS et sont régulièrement mis à jour afin d'installer la dernière version. La version actuelle de chaque moteur d'exécution est répertoriée sur la [page d'exécution \(p. 122\)](#). Pour utiliser une version plus récente du kit SDK AWS ou pour imposer une version spécifique à vos fonctions, vous pouvez regrouper la bibliothèque avec votre code de fonction ou [créer une couche Lambda \(p. 76\)](#). Pour plus d'informations sur la création d'un package de déploiement avec des dépendances, consultez les rubriques suivantes :

- [Package de déploiement AWS Lambda dans Node.js \(p. 316\)](#)
- [Package de déploiement AWS Lambda dans Python \(p. 332\)](#)
- [Package de déploiement AWS Lambda en Ruby \(p. 350\)](#)
- [Package de déploiement AWS Lambda en Java \(p. 366\)](#)
- [Package de déploiement AWS Lambda dans Go \(p. 398\)](#)
- [Package de déploiement AWS Lambda dans C# \(p. 414\)](#)
- [Package de déploiement AWS Lambda dans PowerShell \(p. 436\)](#)

Problème : (Python) certaines bibliothèques ne se chargent pas correctement à partir du package de déploiement

Les bibliothèques avec des modules d'extension écrits en C ou C++ doivent être compilées dans un environnement avec la même architecture de processeur qu'Lambda (Amazon Linux). Pour de plus amples informations, veuillez consulter [Package de déploiement AWS Lambda dans Python \(p. 332\)](#).

Résolution des problèmes de mise en réseau dans AWS Lambda

Par défaut, Lambda exécute les fonctions dans un Virtual Private Cloud (VPC) interne avec connectivité aux services AWS et à Internet. Pour accéder aux ressources réseau en local, vous pouvez configurer

votre fonction afin qu'elle se connecte à un VPC de votre compte ([p. 81](#)). Lorsque vous utilisez cette fonctionnalité, vous gérez l'accès Internet et la connectivité réseau de la fonction avec les ressources VPC.

Les erreurs de connectivité réseau peuvent résulter de problèmes au niveau de la configuration du routage, des règles de groupe de sécurité, des autorisations de rôle, de la traduction d'adresses réseau ou de la disponibilité des ressources telles que les adresses IP ou les interfaces réseau. Ces problèmes peuvent générer une erreur spécifique ou, si une demande n'atteint pas sa destination, l'expiration de la tentative.

Problème : la fonction perd l'accès Internet après la connexion à un VPC

Erreur : Error: connect ETIMEDOUT 176.32.98.189:443

Erreur : Error: Task timed out after 10.00 seconds

Lorsque vous connectez une fonction à un VPC, toutes les demandes sortantes passent par votre VPC. Pour vous connecter à Internet, configurez votre VPC pour envoyer le trafic sortant depuis le sous-réseau de la fonction vers une passerelle NAT dans un sous-réseau public. Pour obtenir plus d'informations et des exemples de configurations VPC, consultez [Accès à Internet et aux services pour des fonctions connectées à un VPC \(p. 83\)](#).

Problème : la fonction doit accéder aux services AWS sans utiliser Internet

Pour vous connecter aux services AWS à partir d'un sous-réseau privé sans accès Internet, utilisez les points de terminaison du VPC. Pour obtenir un exemple de modèle avec des points de terminaison de VPC pour DynamoDB et Amazon S3, consultez [??? \(p. 84\)](#).

Erreur : ENILimitReachedException: The elastic network interface limit was reached for the function's VPC.

Lorsque vous connectez une fonction à un VPC, Lambda crée une interface réseau Elastic pour chaque combinaison de sous-réseau et de groupe de sécurité attachés à cette fonction. Ces interfaces réseau sont limitées à 250 pour chaque VPC, mais cette limite peut être augmentée. Pour demander une augmentation, utilisez la [console Support Center](#).

Versions 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 |
|--|---|---------------------|
| Exemples d'applications CDK AWS dans la console Lambda | La console Lambda inclut désormais des exemples d'applications qui utilisent kit AWS Cloud Development Kit (AWS CDK) pour TypeScript. Le kit AWS CDK est un framework qui vous permet de définir vos ressources d'application en TypeScript, Python, Java ou .NET. Pour obtenir un didacticiel sur la création d'applications, consultez Création d'une application avec distribution continue dans la console Lambda . | June 1, 2020 |
| Prise en charge de l'environnement d'exécution .NET Core 3.1.0 dans AWS Lambda | AWS Lambda prend maintenant en charge l'exécution de .NET Core 3.1.0. Pour de plus amples informations, veuillez consulter Interface de ligne de commande .NET Core . | March 31, 2020 |
| Prise en charge des API HTTP API Gateway | Documentation mise à jour et étendue pour l'utilisation de Lambda avec API Gateway, y compris la prise en charge des API HTTP. Ajout d'un exemple d'application qui crée une API et une fonction avec AWS CloudFormation. Pour de plus amples informations, veuillez consulter Utilisation d'AWS Lambda avec Amazon API Gateway . | March 23, 2020 |
| Ruby 2.7 | Un nouveau runtime est disponible pour Ruby 2.7, ruby2.7, qui est le premier runtime Ruby à utiliser Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Ruby . | February 19, 2020 |
| Métriques de simultanéité | AWS Lambda signale maintenant la métrique <code>ConcurrentExecutions</code> | February 18, 2020 |

pour l'ensemble des fonctions, des alias et des versions. Vous pouvez afficher un graphique pour cette métrique dans la page de surveillance de votre fonction. Auparavant, `ConcurrentExecutions` était seulement signalé au niveau du compte et pour les fonctions utilisant la simultanéité réservée. Pour plus d'informations, consultez [Métriques des fonctions AWS Lambda](#).

Mise à jour des états de fonction

Les états de fonction sont désormais appliqués pour toutes les fonctions par défaut. Lorsque vous connectez une fonction à un VPC, Lambda crée des interfaces réseau Elastic partagées. Cela permet à votre fonction d'évoluer sans créer d'interfaces réseau supplémentaires. Pendant ce temps, vous ne pouvez pas effectuer d'opérations supplémentaires sur la fonction, y compris la mise à jour de sa configuration et de ses versions de publication. Dans certains cas, l'appel est également affecté. Les détails sur l'état actuel d'une fonction sont disponibles à partir de l'API Lambda.

January 24, 2020

Cette mise à jour est publiée en plusieurs phases. Pour de plus amples informations, veuillez consulter [Updated Lambda states lifecycle for VPC networking](#) sur le blog AWS Compute. Pour plus d'informations sur les états, consultez [États de fonctions AWS Lambda](#).

Mises à jour de la sortie de l'API de configuration de fonction

Ajout de codes de motif à `StateReasonCode` (`InvalidSubnet`, `InvalidSecurityGroup`) et `LastUpdateStatusReasonCode` (`SubnetOutOfRangeIPAddresses`, `InvalidSubnet`, `InvalidSecurityGroup`) pour les fonctions qui se connectent à un VPC. Pour plus d'informations sur les états, consultez [États de fonctions AWS Lambda](#).

January 20, 2020

| | | |
|---|--|-------------------|
| Simultanéité provisionnée | <p>Vous pouvez désormais allouer la simultanéité provisionnée pour une version de fonction ou un alias. La simultanéité provisionnée permet à une fonction d'évoluer sans fluctuations de latence. Pour plus d'informations, consultez Gestion de la simultanéité pour une fonction Lambda.</p> | December 3, 2019 |
| Créer un proxy de base de données | <p>Vous pouvez désormais utiliser la console Lambda pour créer un proxy de base de données pour une fonction Lambda. Un proxy de base de données permet à une fonction d'atteindre des niveaux de simultanéité élevés sans épuiser les connexions de base de données. Pour plus d'informations, consultez Configuration de l'accès à une base de données pour une fonction Lambda.</p> | December 3, 2019 |
| Prise en charge des percentiles pour la métrique de durée | <p>Vous pouvez désormais filtrer la métrique de durée en fonction des percentiles. Pour plus d'informations, consultez Métriques AWS Lambda.</p> | November 26, 2019 |
| États de fonction | <p>Lorsque vous créez ou mettez à jour une fonction, celle-ci prend un état en attente pendant que Lambda provisionne les ressources pour sa prise en charge. Si vous connectez votre fonction à un VPC, Lambda peut créer immédiatement une interface réseau Elastic partagée, plutôt que de créer des interfaces réseau lorsque votre fonction est appelée. Cela se traduit par de meilleures performances pour les fonctions connectées au VPC, mais peut nécessiter une mise à jour de votre automatisation. Pour plus d'informations, consultez États de fonction AWS Lambda.</p> | November 25, 2019 |

| | | |
|--|--|-------------------|
| Options de gestion des erreurs pour un appel asynchrone | De nouvelles options de configuration sont disponibles pour l'appel asynchrone. Vous pouvez configurer Lambda de manière à limiter les nouvelles tentatives et à définir un âge d'événement maximal. Pour plus d'informations, consultez Configuration de la gestion des erreurs pour les appels asynchrones . | November 25, 2019 |
| Gestion des erreurs pour les sources d'événement de flux | De nouvelles options de configuration sont disponibles pour les mappages de source d'événement qui lisent depuis les flux. Vous pouvez configurer des mappages de source d'événement de flux DynamoDB et de flux Kinesis pour limiter les nouvelles tentatives et définir un âge d'enregistrement maximal. Lorsque des erreurs se produisent, vous pouvez configurer le mappage de source d'événement pour fractionner les lots avant d'effectuer des nouvelles tentatives et d'envoyer des enregistrements d'appel pour les lots ayant échoué vers une file d'attente ou une rubrique. Pour plus d'informations, consultez Mappage de source d'événement AWS Lambda . | November 25, 2019 |
| Destinations d'appel asynchrone | Vous pouvez désormais configurer Lambda pour envoyer des enregistrements d'appel asynchrone à un autre service. Les enregistrements d'appel contiennent des détails sur l'événement, le contexte et la réponse de la fonction. Vous pouvez envoyer des enregistrements d'appel à une file d'attente SQS, une rubrique SNS, une fonction Lambda ou un bus d'événements EventBridge. Pour plus d'informations, consultez Configuration des destinations pour les appels asynchrones . | November 25, 2019 |

| | | |
|--|--|-------------------|
| Augmentation de la simultanéité pour des sources d'événement de flux | Une nouvelle option pour les mappages de source d'événement de flux DynamoDB et de flux Kinesis vous permet de traiter plusieurs lots à la fois à partir de chaque partition. Lorsque vous augmentez le nombre de lots simultanés par partition, la simultanéité de votre fonction peut atteindre dix fois le nombre de partitions dans votre flux. Pour plus d'informations, consultez Mappage de source d'événement AWS Lambda . | November 25, 2019 |
| Prise en charge des files d'attente FIFO pour les sources d'événement Amazon SQS | Vous pouvez désormais créer un mappage de source d'événement qui lit à partir d'une file d'attente FIFO (premier entré, premier sorti). Auparavant, seules les files d'attente standard étaient prises en charge. Pour plus d'informations, consultez Utilisation d'AWS Lambda avec Amazon SQS . | November 18, 2019 |
| Nouveaux environnements d'exécution pour Node.js, Python et Java | De nouveaux environnements d'exécution sont disponibles pour Node.js 12, Python 3.8 et Java 11. Pour plus d'informations, consultez Environnements d'exécution AWS Lambda . | November 18, 2019 |
| Créer des applications dans la console Lambda | La création d'applications dans la console Lambda est désormais disponible au niveau mondial. Pour obtenir des instructions, consultez Création d'une application avec distribution continue dans la console Lambda . | October 31, 2019 |

| | | |
|---|---|-------------------|
| Créer des applications dans la console Lambda (bêta) | <p>Vous pouvez désormais créer une application Lambda avec un pipeline de distribution continue intégré dans la console Lambda. La console fournit des exemples d'applications que vous pouvez utiliser comme point de départ pour votre propre projet. Choisissez entre AWS CodeCommit et GitHub pour le contrôle source. Chaque fois que vous transmettez des modifications à votre référentiel, le pipeline inclus les construit et les déploie automatiquement. Pour obtenir des instructions, consultez Création d'une application avec distribution continue dans la console Lambda.</p> | October 3, 2019 |
| Améliorations des performances pour les fonctions connectées à un VPC | <p>Lambda utilise désormais un nouveau type d'interface réseau Elastic qui est partagé par toutes les fonctions d'un sous-réseau Virtual Private Cloud (VPC). Lorsque vous connectez une fonction à un VPC, Lambda crée une interface réseau pour chaque combinaison de groupe de sécurité et de sous-réseau que vous choisissez. Lorsque les interfaces réseau partagées sont disponibles, la fonction n'a plus besoin de créer d'interfaces réseau supplémentaires au fur et à mesure de sa mise à l'échelle ascendante. Ainsi, les temps de démarrage sont considérablement améliorés. Pour plus d'informations, consultez Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC.</p> | September 3, 2019 |

| | | |
|--|---|-----------------|
| Paramètres de traitement par lots de flux | <p>Vous pouvez désormais configurer une fenêtre de traitement par lots pour les mappages de source d'événement Amazon DynamoDB et Amazon Kinesis. Configurez une fenêtre de traitement par lots de cinq minutes maximum pour mettre en mémoire tampon les enregistrements entrants jusqu'à ce qu'un lot complet soit disponible. Cela réduit le nombre de fois où votre fonction est appelée lorsque le flux est moins actif.</p> | August 29, 2019 |
| Intégration des informations CloudWatch Logs | <p>La page de surveillance de la console Lambda inclut désormais les rapports d'informations Amazon CloudWatch Logs. Pour plus d'informations, consultez Surveillance des fonctions dans la console AWS Lambda.</p> | June 18, 2019 |
| Amazon Linux 2018.03 | <p>L'environnement d'exécution Lambda a été mis à jour pour utiliser Amazon Linux 2018.03. Pour plus d'informations, consultez Environnement d'exécution.</p> | May 21, 2019 |
| Node.js 10 | <p>Un nouvel environnement d'exécution est disponible pour Node.js 10, nodejs10.x. Cet environnement d'exécution utilise Node.js 10.15 et sera mis à jour avec la dernière version de Node.js 10 régulièrement. Node.js 10 est également le premier environnement d'exécution pour utiliser Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js.</p> | May 13, 2019 |

| | | |
|--|--|-------------------|
| GetLayerVersionByArn API | Utilisez l'API GetLayerVersionByArn pour télécharger des informations de version d'une couche avec l'ARN de version comme entrée. Comparé à GetLayerVersion, GetLayerVersionByArn vous permet d'utiliser l'ARN directement au lieu de l'analyser pour obtenir le nom de la couche et le numéro de version. | April 25, 2019 |
| Ruby | AWS Lambda prend désormais en charge Ruby 2.5 avec un nouvel environnement d'exécution. Pour plus d'informations, consultez Création de fonctions Lambda avec Ruby . | November 29, 2018 |
| Environnements d'exécution personnalisés | Créez un runtime personnalisé pour exécuter des fonctions Lambda dans votre langage de programmation favori. Pour plus d'informations, consultez Environnements d'exécution AWS Lambda personnalisés . | November 29, 2018 |
| Déclencheurs des équilibreurs de charge d'applications | Elastic Load Balancing prend désormais en charge les fonctions Lambda en tant que cibles pour les Application Load Balancers. Pour plus d'informations, consultez Utilisation de Lambda avec des équilibreurs de charge d'applications . | November 29, 2018 |
| Couches | 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. Pour plus d'informations, consultez Couches AWS Lambda . | November 29, 2018 |

| | | |
|--|--|-------------------|
| Utilisation des consommateurs de flux Kinesis HTTP/2 comme déclencheur | 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. Pour de plus amples informations, veuillez consulter la section Utilisation d'AWS Lambda avec Kinesis . | November 19, 2018 |
| Python 3.7 | AWS Lambda prend désormais en charge Python 3.7 avec un nouveau runtime. Pour plus d'informations, consultez Création de fonctions Lambda avec Python . | November 19, 2018 |
| Augmentation de la limite de la charge utile d'appel de fonction asynchrone | 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. Pour plus d'informations, consultez Limites AWS Lambda . | November 16, 2018 |
| Région AWS GovCloud (USA Est) | AWS Lambda est désormais disponible dans la région AWS GovCloud (USA Est). Pour en savoir plus, consultez la page 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 | 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>Vous pouvez afficher le statut de vos applications Lambda sur 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.</p> | October 11, 2018 |
| Délai d'attente d'exécution de la fonction | <p>Pour permettre d'utiliser des fonctions de longue durée, le délai d'exécution maximal configurable est passé de 5 minutes à 15 minutes. Pour plus d'informations, consultez Limites AWS Lambda.</p> | October 10, 2018 |
| Prise en charge du langage PowerShell Core dans AWS Lambda | <p>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.</p> | September 11, 2018 |
| Prise en charge de l'exécution .NET Core 2.1.0 dans AWS Lambda | <p>AWS Lambda prend maintenant en charge l'exécution de .NET Core 2.1.0. Pour en savoir plus, consultez la page Interface de ligne de commande .NET Core.</p> | July 9, 2018 |
| Mises à jour disponibles sur RSS | <p>Vous pouvez maintenant vous abonner à un flux RSS pour suivre les versions de ce guide.</p> | July 5, 2018 |
| Région Chine (Ningxia) | <p>AWS Lambda est désormais disponible dans la région Région Chine (Ningxia). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le AWS General Reference.</p> | June 28, 2018 |
| Prise en charge pour Amazon SQS en tant que source d'événement | <p>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.</p> | 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 l'environnement d'exécution Node.js version 8.10 Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js (p. 312) . | 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 |
| Prise en charge de l'environnement d'exécution Go et .NET 2.0 | La prise en charge de l'environnement d'exécution pour Go et .NET 2.0 a été ajoutée pour AWS Lambda. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Go (p. 398) et Création de fonctions Lambda avec C# (p. 413) . | 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 de plus amples informations, veuillez consulter Création de fonctions à l'aide de l'éditeur de la console AWS Lambda (p. 6) . | 30 novembre 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 de plus amples informations, veuillez consulter Gestion de la simultanéité pour une fonction Lambda (p. 61) . | 30 novembre 2017 |
| Déplacement du trafic avec des alias | AWS Lambda prend désormais en charge le déplacement du trafic avec des alias. Pour de plus amples informations, veuillez consulter Déploiements propagés pour les fonctions Lambda (p. 148) . | 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'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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 l'environnement d'exécution Lambda. Pour plus | 11 août 2017 |

| Modification | Description | Date |
|---|--|------------------|
| | d'informations, consultez Test et débogage d'applications sans serveur . | |
| Région Canada (Centre) | AWS Lambda est désormais disponible dans la région Région Canada (Centre). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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 de plus amples informations, veuillez consulter Utilisation d'AWS Lambda avec AWS X-Ray (p. 296) . | 19 avril 2017 |
| Région Asie-Pacifique (Mumbai) | AWS Lambda est désormais disponible dans la région Région Asie-Pacifique (Mumbai). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le AWS General Reference. | 28 mars 2017 |
| AWS Lambda prend désormais en charge l'environnement d'exécution Node.js v6.10 | AWS Lambda prend désormais en charge l'environnement d'exécution Node.js v6.10. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js (p. 312) . | 22 mars 2017 |
| Région Europe (Londres) | AWS Lambda est désormais disponible dans la région Région Europe (Londres). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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. | AWS Lambda a ajouté la prise en charge de C#. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec C# (p. 413) . Lambda@Edge vous permet d'exécuter des fonctions Lambda au niveau des emplacements périphériques AWS en réponse à des événements CloudFront. Pour de plus amples informations, veuillez consulter Utilisation de AWS Lambda avec CloudFront Lambda@Edge (p. 197) . | 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 de plus amples informations, veuillez consulter Utilisation de AWS Lambda avec Amazon Lex (p. 255) . | 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'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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>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 de plus amples informations, veuillez consulter Applications AWS Lambda (p. 136).</p> <p>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 de plus amples informations, veuillez consulter Utilisation des variables d'environnement AWS Lambda (p. 55).</p> | 18 novembre 2016 |
| Région Asie-Pacifique (Séoul) | AWS Lambda est désormais disponible dans la région Région Asie-Pacifique (Séoul). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le AWS General Reference. | 29 août 2016 |
| Région Asie-Pacifique (Sydney) | Lambda est désormais disponible dans la région Région Asie-Pacifique (Sydney). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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 de plus amples informations, veuillez consulter Créer une fonction Lambda avec la console (p. 3) . | 23 juin 2016 |
| AWS Lambda prend désormais en charge l'environnement d'exécution Node.js v4.3 | AWS Lambda prend désormais en charge l'environnement d'exécution Node.js v4.3. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js (p. 312) . | 07 avril 2016 |
| Europe (Francfort) région | Lambda est désormais disponible dans la région Europe (Francfort). Pour plus d'informations sur les régions et points de terminaison Lambda, consultez Régions et points de terminaison dans le 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 de plus amples informations, veuillez consulter Configuration d'une fonction Lambda pour accéder aux ressources d'un VPC (p. 81) . | 11 février 2016 |
| L'environnement d'exécution AWS Lambda a été mis à jour. | L'environnement d'exécution (p. 122) a été mis à jour. | 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>Vous pouvez désormais développer le code de la fonction Lambda à l'aide du langage Python. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Python (p. 329).</p> <p>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 de plus amples informations, veuillez consulter Versions de fonction AWS Lambda (p. 70).</p> <p>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, veuillez consulter Utilisation de AWS Lambda avec Amazon CloudWatch Events (p. 188).</p> <p>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.</p> | 08 octobre 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 de procédure, veuillez consulter Didacticiel : Utilisation d'AWS Lambda avec les flux Amazon DynamoDB (p. 214) . | 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 voir un exemple d'exercice de mise en route, veuillez consulter Utilisation de AWS Lambda avec Amazon API Gateway (p. 157).</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 de plus amples informations, veuillez consulter Mise en route avec AWS Lambda (p. 3) . | 09 juillet 2015 |
| 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 Création de fonctions Lambda avec Java (p. 363) . | 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 style="margin-left: 2em;">Mise en route avec AWS Lambda (p. 3)</p> <p style="margin-left: 2em;">AWS Lambda</p> | 9 avril 2015 |
| Version préliminaire | Version préliminaire du Manuel du développeur AWS Lambda. | 13 novembre 2014 |

Référence d'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 le Référence générale d'Amazon Web Services.

Pour obtenir une présentation du service, consultez la section [Présentation d'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 \(p. 485\)](#)
- [Data Types \(p. 655\)](#)

Actions

The following actions are supported:

- [AddLayerVersionPermission \(p. 487\)](#)
- [AddPermission \(p. 490\)](#)
- [CreateAlias \(p. 494\)](#)
- [CreateEventSourceMapping \(p. 498\)](#)
- [CreateFunction \(p. 504\)](#)
- [DeleteAlias \(p. 513\)](#)
- [DeleteEventSourceMapping \(p. 515\)](#)
- [DeleteFunction \(p. 519\)](#)
- [DeleteFunctionConcurrency \(p. 521\)](#)
- [DeleteFunctionEventInvokeConfig \(p. 523\)](#)
- [DeleteLayerVersion \(p. 525\)](#)
- [DeleteProvisionedConcurrencyConfig \(p. 527\)](#)
- [GetAccountSettings \(p. 529\)](#)
- [GetAlias \(p. 531\)](#)
- [GetEventSourceMapping \(p. 534\)](#)
- [GetFunction \(p. 538\)](#)
- [GetFunctionConcurrency \(p. 541\)](#)
- [GetFunctionConfiguration \(p. 543\)](#)
- [GetFunctionEventInvokeConfig \(p. 549\)](#)
- [GetLayerVersion \(p. 552\)](#)
- [GetLayerVersionByArn \(p. 555\)](#)
- [GetLayerVersionPolicy \(p. 558\)](#)
- [GetPolicy \(p. 560\)](#)
- [GetProvisionedConcurrencyConfig \(p. 562\)](#)
- [Invoke \(p. 565\)](#)

- [InvokeAsync \(p. 570\)](#)
- [ListAliases \(p. 572\)](#)
- [ListEventSourceMappings \(p. 575\)](#)
- [ListFunctionEventInvokeConfigs \(p. 578\)](#)
- [ListFunctions \(p. 581\)](#)
- [ListLayers \(p. 584\)](#)
- [ListLayerVersions \(p. 586\)](#)
- [ListProvisionedConcurrencyConfigs \(p. 589\)](#)
- [ListTags \(p. 592\)](#)
- [ListVersionsByFunction \(p. 594\)](#)
- [PublishLayerVersion \(p. 597\)](#)
- [PublishVersion \(p. 601\)](#)
- [PutFunctionConcurrency \(p. 608\)](#)
- [PutFunctionEventInvokeConfig \(p. 611\)](#)
- [PutProvisionedConcurrencyConfig \(p. 615\)](#)
- [RemoveLayerVersionPermission \(p. 618\)](#)
- [RemovePermission \(p. 620\)](#)
- [TagResource \(p. 622\)](#)
- [UntagResource \(p. 624\)](#)
- [UpdateAlias \(p. 626\)](#)
- [UpdateEventSourceMapping \(p. 630\)](#)
- [UpdateFunctionCode \(p. 636\)](#)
- [UpdateFunctionConfiguration \(p. 643\)](#)
- [UpdateFunctionEventInvokeConfig \(p. 652\)](#)

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 \(p. 618\)](#) 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 \(p. 487\)](#)

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 \(p. 487\)](#)

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 \(p. 487\)](#)

The version number.

Request Body

The request accepts the following data in JSON format.

[Action \(p. 487\)](#)

The API action that grants access to the layer. For example, `lambda:GetLayerVersion`.

Type: String

Pattern: `lambda:GetLayerVersion`

Required: Yes

[OrganizationId \(p. 487\)](#)

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 \(p. 487\)](#)

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 \(p. 487\)](#)

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 \(p. 488\)](#)

A unique identifier for the current revision of the policy.

Type: String

[Statement \(p. 488\)](#)

The permission statement.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400
`PolicyLengthExceededException`

The permissions policy for the resource is too large. [Learn more](#)

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, or another operation is in progress.

HTTP Status Code: 409
`ResourceNotFoundException`

The resource 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`

The request throughput limit was 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 V3](#)

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 of the associated resource as the `SourceArn`. 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 permissions 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. 490)

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. 490)

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 \(p. 490\)](#)

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 \(p. 490\)](#)

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 \(p. 490\)](#)

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 \(p. 490\)](#)

Only update the policy if the revision ID matches the ID that's specified. Use this option to avoid modifying a policy that has changed since you last read it.

Type: String

Required: No

[SourceAccount \(p. 490\)](#)

For Amazon S3, the ID of the account that owns the resource. Use this together with `SourceArn` to ensure that the resource is owned by the specified account. It is possible for an Amazon S3 bucket to be deleted by its owner and recreated by another account.

Type: String

Pattern: \d{12}

Required: No

[SourceArn \(p. 490\)](#)

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 \(p. 490\)](#)

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 \(p. 492\)](#)

The permission statement that's added to the function policy.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

PolicyLengthExceededException

The permissions policy for the resource is too large. [Learn more](#)

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, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 494)

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. 494)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 494\)](#)

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 \(p. 494\)](#)

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 \(p. 494\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 661\)](#) 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 \(p. 495\)](#)

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 \(p. 495\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 495\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 495\)](#)

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 \(p. 495\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 495\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 661\)](#) object

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource 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

The request throughput limit was 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 V3](#)

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 DynamoDB](#)
- [Using AWS Lambda with Amazon Kinesis](#)
- [Using AWS Lambda with Amazon SQS](#)

The following error handling options are only available for stream sources (DynamoDB and Kinesis):

- `BisectBatchOnFunctionError` - If the function returns an error, split the batch in two and retry.
- `DestinationConfig` - Send discarded records to an Amazon SQS queue or Amazon SNS topic.
- `MaximumRecordAgeInSeconds` - Discard records older than the specified age.
- `MaximumRetryAttempts` - Discard records after the specified number of retries.
- `ParallelizationFactor` - Process multiple batches from each shard concurrently.

Request Syntax

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "Enabled": boolean,
    "EventSourceArn": "string",
    "FunctionName": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "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 \(p. 498\)](#)

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

[BisectBatchOnFunctionError \(p. 498\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

[DestinationConfig \(p. 498\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

Required: No

[Enabled \(p. 498\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

[EventSourceArn \(p. 498\)](#)

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 \(p. 498\)](#)

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

[MaximumBatchingWindowInSeconds \(p. 498\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

[MaximumRecordAgeInSeconds \(p. 498\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

[MaximumRetryAttempts \(p. 498\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

[ParallelizationFactor \(p. 498\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

[StartingPosition \(p. 498\)](#)

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 \(p. 498\)](#)

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,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "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 \(p. 501\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 501\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 501\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

[EventSourceArn \(p. 501\)](#)

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 \(p. 501\)](#)

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 \(p. 501\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 501\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 501\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 501\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 501\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 501\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 501\)](#)

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 \(p. 501\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String
[UUID \(p. 501\)](#)

The identifier of the event source mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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.

When you create a function, Lambda provisions an instance of the function and its supporting resources. If your function connects to a VPC, this process can take a minute or so. During this time, you can't invoke or modify the function. The `State`, `StateReason`, and `StateReasonCode` fields in the response from [GetFunctionConfiguration \(p. 543\)](#) indicate when the function is ready to invoke. For more information, see [Function States](#).

A function has an unpublished version, and can have published versions and aliases. The unpublished version changes when you update your function's code and configuration. A published version is a snapshot of your function code and configuration that can't 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 \(p. 643\)](#). Function-level settings apply to both the unpublished and published versions of the function, and include tags ([TagResource \(p. 622\)](#)) and per-function concurrency limits ([PutFunctionConcurrency \(p. 608\)](#)).

If another account or an AWS service invokes your function, use [AddPermission \(p. 490\)](#) 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 \(p. 565\)](#). To invoke your function in response to events in other AWS services, create an event source mapping ([CreateEventSourceMapping \(p. 498\)](#)), 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",
  "Timeout": number
}
```

```
"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 \(p. 504\)](#)

The code for the function.

Type: [FunctionCode \(p. 671\)](#) object

Required: Yes

[DeadLetterConfig \(p. 504\)](#)

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 \(p. 663\)](#) object

Required: No

[Description \(p. 504\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 504\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 665\)](#) object

Required: No

[FunctionName \(p. 504\)](#)

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 \(p. 504\)](#)

The name of the method within your code that Lambda calls to execute your function. The format includes the file name. It 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 \(p. 504\)](#)

The ARN of the AWS Key Management Service (AWS KMS) key that's used to encrypt your function's environment variables. If it's not provided, AWS Lambda uses a default service key.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.]+|()`

Required: No

[Layers \(p. 504\)](#)

A list of [function layers](#) to add to the function's execution environment. Specify each layer by its 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 \(p. 504\)](#)

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 \(p. 504\)](#)

Set to true to publish the first version of the function during creation.

Type: Boolean

Required: No

[Role \(p. 504\)](#)

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 \(p. 504\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Required: Yes

[Tags \(p. 504\)](#)

A list of [tags](#) to apply to the function.

Type: String to string map

Required: No

[Timeout \(p. 504\)](#)

The amount of time that Lambda allows a function to run before stopping it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 504\)](#)

Set `Mode` to `Active` to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 690\)](#) object

Required: No

[VpcConfig \(p. 504\)](#)

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 \(p. 692\)](#) 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",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "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 \(p. 508\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 508\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 508\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 663\)](#) object

[Description \(p. 508\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 508\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 667\)](#) object

[FunctionArn \(p. 508\)](#)

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 \(p. 508\)](#)

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 \(p. 508\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 508\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-z0-9-.]+:[.*])|()`

[LastModified \(p. 508\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 508\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 508\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 508\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 508\)](#)

The function's `layers`.

Type: Array of [Layer \(p. 680\)](#) objects

[MasterArn \(p. 508\)](#)

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 \(p. 508\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 508\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 508\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+`
[Runtime \(p. 508\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs10.x` | `nodejs12.x` | `java8` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `dotnetcore2.1` | `dotnetcore3.1` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided`

[State \(p. 508\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 508\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 508\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Timeout \(p. 508\)](#)

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 508\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) object

[Version \(p. 508\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 508\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 693\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 513)

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. 513)

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/1.1 204
```

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.

- HTTP Status Code: 400
ResourceConflictException

The resource already exists, or another operation is in progress.
- HTTP Status Code: 409
ServiceException

The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was 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 V3](#)

DeleteEventSourceMapping

Deletes an [event source mapping](#). You can get the identifier of a mapping from the output of [ListEventSourceMappings](#) (p. 575).

When you delete an event source mapping, it enters a `Deleting` state and might not be completely deleted for several seconds.

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. 515)

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,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "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 \(p. 515\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 515\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 515\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

[EventSourceArn \(p. 515\)](#)

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 \(p. 515\)](#)

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 \(p. 515\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 515\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 515\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 515\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 515\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 515\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 515\)](#)

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 \(p. 515\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 515\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

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 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](#)

The request throughput limit was 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 V3](#)

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 \(p. 515\)](#). 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 \(p. 519\)](#)

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 \(p. 519\)](#)

Specify a version to delete. You can't delete a version that's 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.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 521)

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.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

DeleteFunctionEventInvokeConfig

Deletes the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 611\)](#).

Request Syntax

```
DELETE /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 523\)](#)

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 \(p. 523\)](#)

A version number or alias name.

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.

HTTP Status Code: 400
ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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 \(p. 525\)](#)

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 \(p. 525\)](#)

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

The request throughput limit was 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 V3](#)

DeleteProvisionedConcurrencyConfig

Deletes the provisioned concurrency configuration for a function.

Request Syntax

```
DELETE /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 527)

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-_]+))?`

[Qualifier](#) (p. 527)

The version number or alias name.

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.

- HTTP Status Code: 400
`ResourceConflictException`

The resource already exists, or another operation is in progress.
- HTTP Status Code: 409
`ResourceNotFoundException`

The resource 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`

The request throughput limit was 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 V3](#)

GetAccountSettings

Retrieves details about your account's [limits](#) and usage in an AWS 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. 529)

Limits that are related to concurrency and code storage.

Type: [AccountLimit](#) (p. 657) object

[AccountUsage](#) (p. 529)

The number of functions and amount of storage in use.

Type: [AccountUsage](#) (p. 658) object

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was 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 V3](#)

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. 531)

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. 531)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-])^[\w-]{1,128}$`

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. 531)

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. 531)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion](#) (p. 531)

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. 531)

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. 531)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig](#) (p. 531)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration](#) (p. 661) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

GetEventSourceMapping

Returns details about an event source mapping. You can get the identifier of a mapping from the output of [ListEventSourceMappings \(p. 575\)](#).

Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 534\)](#)

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,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "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 \(p. 534\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 534\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 534\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

[EventSourceArn \(p. 534\)](#)

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-zA-Z]+\d{1})?:(\d{12})?:(.*?)

[FunctionArn \(p. 534\)](#)

The ARN of the Lambda 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-_]+))?

[LastModified \(p. 534\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 534\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 534\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 534\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 534\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 534\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 534\)](#)

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 \(p. 534\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 534\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource 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](#)

The request throughput limit was 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 V3](#)

GetFunction

Returns information about the 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 that are 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. 538)

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. 538)

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",
            "LastUpdateStatus": "string",
            "LastUpdateStatusReason": "string",
            "LastUpdateStatusReasonCode": "string",
            "Layers": [
                {
                    "Arn": "string",
                    "CodeSize": number
                }
            ],
            "MasterArn": "string",
            "MemorySize": number,
            "RevisionId": "string",
            "Role": "string",
            "Runtime": "string",
            "State": "string",
            "StateReason": "string",
            "StateReasonCode": "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 \(p. 538\)](#)

The deployment package of the function or version.

Type: [FunctionCodeLocation \(p. 672\)](#) object
[Concurrency \(p. 538\)](#)

The function's reserved concurrency.

Type: [Concurrency \(p. 662\)](#) object
[Configuration \(p. 538\)](#)

The configuration of the function or version.

Type: [FunctionConfiguration \(p. 673\)](#) object
[Tags \(p. 538\)](#)

The function's tags.

Type: String to string map

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid.

HTTP Status Code: 400

`ResourceNotFoundException`

The resource 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`

The request throughput limit was 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 V3](#)

GetFunctionConcurrency

Returns details about the reserved concurrency configuration for a function. To set a concurrency limit for a function, use [PutFunctionConcurrency \(p. 608\)](#).

Request Syntax

```
GET /2019-09-30/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 541\)](#)

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/1.1 200
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 \(p. 541\)](#)

The number of simultaneous executions that are reserved for the function.

Type: Integer

Valid Range: Minimum value of 0.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

GetFunctionConfiguration

Returns 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 \(p. 643\)](#).

To get all of a function's details, including function-level settings, use [GetFunction \(p. 538\)](#).

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 \(p. 543\)](#)

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 \(p. 543\)](#)

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",
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "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. 543)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize](#) (p. 543)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig](#) (p. 543)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 663\)](#) object

[Description \(p. 543\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 543\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 667\)](#) object

[FunctionArn \(p. 543\)](#)

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 \(p. 543\)](#)

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 \(p. 543\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

[KMSKeyArn \(p. 543\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.:*)|()

[LastModified \(p. 543\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 543\)](#)

The status of the last update that was performed on the function. This is first set to [Successful](#) after function creation completes.

Type: String

Valid Values: Successful | Failed | InProgress

[LastUpdateStatusReason \(p. 543\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 543\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup

[Layers \(p. 543\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 680\)](#) objects

[MasterArn \(p. 543\)](#)

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 \(p. 543\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 543\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 543\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/_-]+

[Runtime \(p. 543\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 543\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 543\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 543\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Timeout \(p. 543\)](#)

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 543\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) object

[Version \(p. 543\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 543\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 693\)](#) object

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

GetFunctionEventInvokeConfig

Retrieves the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 611\)](#).

Request Syntax

```
GET /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 549)

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. 549)

A version number or alias name.

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

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
}
```

```
    },
    "FunctionArn": "string",
    "LastModified": number,
    "MaximumEventAgeInSeconds": number,
    "MaximumRetryAttempts": 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.

[DestinationConfig \(p. 549\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

[FunctionArn \(p. 549\)](#)

The Amazon Resource Name (ARN) of the 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 \(p. 549\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 549\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 549\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidOperationException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 552)

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. 552)

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 \(p. 552\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 552\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 683\)](#) object

[CreatedDate \(p. 552\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 552\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 552\)](#)

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 \(p. 552\)](#)

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 \(p. 552\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 552\)](#)

The version number.

Type: Long

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

GetLayerVersionByArn

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?find=LayerVersion&Arn=Arn HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Arn](#) (p. 555)

The ARN of the layer version.

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]+

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. 555)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 555\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 683\)](#) object

[CreatedDate \(p. 555\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 555\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 555\)](#)

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 \(p. 555\)](#)

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 \(p. 555\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 555\)](#)

The version number.

Type: Long

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

GetLayerVersionPolicy

Returns the permission policy for a version of an [AWS Lambda layer](#). For more information, see [AddLayerVersionPermission \(p. 487\)](#).

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 \(p. 558\)](#)

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 \(p. 558\)](#)

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 \(p. 558\)](#)

The policy document.

Type: String

[RevisionId \(p. 558\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 560)

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. 560)

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 \(p. 560\)](#)

The resource-based policy.

Type: String

[RevisionId \(p. 560\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource 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](#)

The request throughput limit was 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 V3](#)

GetProvisionedConcurrencyConfig

Retrieves the provisioned concurrency configuration for a function's alias or version.

Request Syntax

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 562)

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-_]+))?

[Qualifier](#) (p. 562)

The version number or alias name.

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

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "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.

[AllocatedProvisionedConcurrentExecutions \(p. 562\)](#)

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

[AvailableProvisionedConcurrentExecutions \(p. 562\)](#)

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

[LastModified \(p. 562\)](#)

The date and time that a user last updated the configuration, in [ISO 8601 format](#).

Type: String

[RequestedProvisionedConcurrentExecutions \(p. 562\)](#)

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

[Status \(p. 562\)](#)

The status of the allocation process.

Type: String

Valid Values: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 562\)](#)

For failed allocations, the reason that provisioned concurrency could not be allocated.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ProvisionedConcurrencyConfigNotFoundException](#)

The specified configuration does not exist.

HTTP Status Code: 404

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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](#).

When an error occurs, your function may be invoked multiple times. Retry behavior varies by error type, client, event source, and invocation type. For example, if you invoke a function asynchronously and it returns an error, Lambda executes the function up to two more times. For more information, see [Retry Behavior](#).

For [asynchronous invocation](#), Lambda adds events to a queue before sending them to your function. If your function does not have enough capacity to keep up with the queue, events may be lost. Occasionally, your function may receive the same event multiple times, even if no error occurs. To retain events that were not processed, configure your function with a [dead-letter queue](#).

The status code in the API response doesn't 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 might 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. 565)

Up to 3583 bytes of base64-encoded data about the invoking client to pass to the function in the `context` object.

[FunctionName](#) (p. 565)

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 \(p. 565\)](#)

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 it's 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 \(p. 565\)](#)

Set to Tail to include the execution log in the response.

Valid Values: None | Tail

[Qualifier \(p. 565\)](#)

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 \(p. 565\)](#)

The 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 \(p. 566\)](#)

The HTTP status code is in the 200 range for a successful request. For the RequestResponse invocation type, this status code is 200. For the Event invocation type, this status code is 202. For the DryRun invocation type, the status code is 204.

The response returns the following HTTP headers.

[ExecutedVersion \(p. 566\)](#)

The version of the function that executed. When you invoke a function with an alias, this indicates which version the alias resolved to.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[FunctionError \(p. 566\)](#)

If present, indicates that an error occurred during function execution. Details about the error are included in the response payload.

[LogResult \(p. 566\)](#)

The last 4 KB of the execution log, which is base64 encoded.

The response returns the following as the HTTP body.

[Payload \(p. 566\)](#)

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 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.

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

KMSInternalServerError

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

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ResourceNotReadyException

The function is inactive and its VPC connection is no longer available. Wait for the VPC connection to reestablish and try again.

HTTP Status Code: 502
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

The request throughput limit was 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 V3](#)

InvokeAsync

This action has been deprecated.

Important

For asynchronous function invocation, use [Invoke \(p. 565\)](#).

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. 570)

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. 570)

The 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 \(p. 570\)](#)

The status code.

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

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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 V3](#)

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. 572)

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. 572)

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. 572)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 572)

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 \(p. 572\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 659\)](#) objects

[NextMarker \(p. 572\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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 \(p. 575\)](#)

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 \(p. 575\)](#)

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 \(p. 575\)](#)

A pagination token returned by a previous call.

[MaxItems \(p. 575\)](#)

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,
            "BisectBatchOnFunctionError": boolean,
            "DestinationConfig": {
                "OnFailure": {
                    "Destination": "string"
                },
                "OnSuccess": {
                    "Destination": "string"
                }
            },
            "EventSourceArn": "string",
            "FunctionArn": "string",
            "LastModified": number,
            "LastProcessingResult": "string",
            "MaximumBatchingWindowInSeconds": number,
            "MaximumRecordAgeInSeconds": number,
            "MaximumRetryAttempts": number,
            "ParallelizationFactor": number,
            "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 \(p. 576\)](#)

A list of event source mappings.

Type: Array of [EventSourceMappingConfiguration \(p. 668\)](#) objects

[NextMarker \(p. 576\)](#)

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.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource 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

The request throughput limit was 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 V3](#)

ListFunctionEventInvokeConfigs

Retrieves a list of configurations for asynchronous invocation for a function.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 611\)](#).

Request Syntax

```
GET /2019-09-25/functions/FunctionName/event-invoke-config/list?  
Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 578\)](#)

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-_]+))?

[Marker \(p. 578\)](#)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems \(p. 578\)](#)

The maximum number of configurations 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  
  
{  
    "FunctionEventInvokeConfigs": [  
        {  
            "DestinationConfig": {  
                "OnFailure": {  
                    "Destination": "string"  
                }  
            }  
        }  
    ]  
}
```

```
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "FunctionArn": "string",
    "LastModified": number,
    "MaximumEventAgeInSeconds": number,
    "MaximumRetryAttempts": 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.

[FunctionEventInvokeConfigs \(p. 578\)](#)

A list of configurations.

Type: Array of [FunctionEventInvokeConfig \(p. 678\)](#) objects

[NextMarker \(p. 578\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource 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](#)

The request throughput limit was 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 V3](#)

ListFunctions

Returns a list of Lambda functions, with the version-specific configuration of each. Lambda returns up to 50 functions per call.

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 \(p. 538\)](#).

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 \(p. 581\)](#)

Set to `ALL` to include entries for all published versions of each function.

Valid Values: `ALL`

[Marker \(p. 581\)](#)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MasterRegion \(p. 581\)](#)

For Lambda@Edge functions, the AWS Region of the master function. For example, `us-east-1` filters the list of functions to only include Lambda@Edge functions replicated from a master function in US East (N. Virginia). If specified, you must set `FunctionVersion` to `ALL`.

Pattern: `ALL | [a-z]{2}(-gov)?-[a-z]+-\d{1}`

[MaxItems \(p. 581\)](#)

The maximum number of functions 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  
  
{  
    "Functions": [  
        {  
            "CodeSha256": "string",  
            "CodeSize": number,  
            "DeadLetterConfig": {  
                "TargetArn": "string"  
            },  
            "Description": "string",  
            "FunctionArn": "string",  
            "FunctionName": "string",  
            "Handler": "string",  
            "LastModified": "string",  
            "MemorySize": number,  
            "Runtime": "string",  
            "Timeout": number  
        }  
    ]  
}
```

```
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "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 \(p. 581\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 673\)](#) objects

[NextMarker \(p. 581\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was 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 V3](#)

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. 584](#))

A runtime identifier. For example, go1.x.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Marker ([p. 584](#))

A pagination token returned by a previous call.

MaxItems ([p. 584](#))

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 \(p. 584\)](#)

A list of function layers.

Type: Array of [LayersListItem \(p. 681\)](#) objects

[NextMarker \(p. 584\)](#)

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.

HTTP Status Code: 400

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was 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 V3](#)

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. 586)

A runtime identifier. For example, go1.x.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[LayerName](#) (p. 586)

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. 586)

A pagination token returned by a previous call.

[MaxItems](#) (p. 586)

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",  
            "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 \(p. 586\)](#)

A list of versions.

Type: Array of [LayerVersionsListItem \(p. 684\)](#) objects

[NextMarker \(p. 586\)](#)

A pagination token returned when the response doesn't contain all versions.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource 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](#)

The request throughput limit was 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 V3](#)

ListProvisionedConcurrencyConfigs

Retrieves a list of provisioned concurrency configurations for a function.

Request Syntax

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?  
List=ALL&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 589)

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-_]+))?

[Marker](#) (p. 589)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 589)

Specify a number to limit the number of configurations returned.

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  
  
{  
    "NextMarker": "string",  
    "ProvisionedConcurrencyConfigs": [  
        {  
            "AllocatedProvisionedConcurrentExecutions": number,  
            "AvailableProvisionedConcurrentExecutions": number,  
            "FunctionArn": "string",  
            "ProvisionedConcurrencyConfig": {  
                "AllocatedConcurrentExecutions": number,  
                "AvailableConcurrentExecutions": number,  
                "FunctionArn": "string",  
                "LastModified": "2019-09-10T12:00:00Z",  
                "ProvisionedConcurrencyType": "PROVISIONED",  
                "ProvisionedConcurrentExecutions": number,  
                "Status": "ACTIVE"  
            }  
        }  
    ]  
}
```

```
        "LastModified": "string",
        "RequestedProvisionedConcurrentExecutions": number,
        "Status": "string",
        "StatusReason": "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 \(p. 589\)](#)

The pagination token that's included if more results are available.

Type: String

[ProvisionedConcurrencyConfigs \(p. 589\)](#)

A list of provisioned concurrency configurations.

Type: Array of [ProvisionedConcurrencyConfigListItem \(p. 688\)](#) objects

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

ListTags

Returns a function's [tags](#). You can also view tags with [GetFunction \(p. 538\)](#).

Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[ARN \(p. 592\)](#)

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 \(p. 592\)](#)

The function's tags.

Type: String to string map

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

ListVersionsByFunction

Returns a list of [versions](#), with the version-specific configuration of each. Lambda returns up to 50 versions per call.

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. 594)

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. 594)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 594)

The maximum number of versions 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

{
  "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",
        "LastUpdateStatus": "string",
        "LastUpdateStatusReason": "string",
        "LastUpdateStatusReasonCode": "string",
        "Layers": [
            {
                "Arn": "string",
                "CodeSize": number
            }
        ],
        "MasterArn": "string",
        "MemorySize": number,
        "RevisionId": "string",
        "Role": "string",
        "Runtime": "string",
        "State": "string",
        "StateReason": "string",
        "StateReasonCode": "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 \(p. 594\)](#)

The pagination token that's included if more results are available.

Type: String

[Versions \(p. 594\)](#)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration \(p. 673\)](#) objects

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

PublishLayerVersion

Creates an [AWS Lambda layer](#) from a ZIP archive. Each time you call `PublishLayerVersion` with the same layer name, a new version is created.

Add layers to your function with [CreateFunction \(p. 504\)](#) or [UpdateFunctionConfiguration \(p. 643\)](#).

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 \(p. 597\)](#)

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 \(p. 597\)](#)

A list of compatible [function runtimes](#). Used for filtering with [ListLayers \(p. 584\)](#) and [ListLayerVersions \(p. 586\)](#).

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: `nodejs10.x` | `nodejs12.x` | `java8` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `dotnetcore2.1` | `dotnetcore3.1` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided`

Required: No

[Content \(p. 597\)](#)

The function layer archive.

Type: [LayerVersionContentInput \(p. 682\)](#) object

Required: Yes

[Description \(p. 597\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[LicenseInfo \(p. 597\)](#)

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 \(p. 598\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 598\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 683\)](#) object

[CreatedDate \(p. 598\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 598\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 598\)](#)

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 \(p. 598\)](#)

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 \(p. 598\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 598\)](#)

The version number.

Type: Long

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400
`InvalidParameterValueException`

One of the parameters in the request is invalid.

HTTP Status Code: 400
`ResourceNotFoundException`

The resource 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`

The request throughput limit was 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 V3](#)

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 doesn't publish a version if the function's configuration and code haven't changed since the last version. Use [UpdateFunctionCode \(p. 636\)](#) or [UpdateFunctionConfiguration \(p. 643\)](#) to update the function before publishing a version.

Clients can invoke versions directly or with an alias. To create an alias, use [CreateAlias \(p. 494\)](#).

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 \(p. 601\)](#)

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 \(p. 601\)](#)

Only publish a version if the hash value matches the value that's 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 that you uploaded from the output of [UpdateFunctionCode \(p. 636\)](#).

Type: String

Required: No

Description (p. 601)

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. 601)

Only update the function if the revision ID matches the ID that's 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",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "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 \(p. 602\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 602\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 602\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 663\)](#) object

[Description \(p. 602\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 602\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 667\)](#) object

[FunctionArn \(p. 602\)](#)

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 \(p. 602\)](#)

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 \(p. 602\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 602\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:\.*|()`

[LastModified \(p. 602\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 602\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful | Failed | InProgress`

[LastUpdateStatusReason \(p. 602\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 602\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfRangeAddresses | InvalidSubnet | InvalidSecurityGroup`

[Layers \(p. 602\)](#)

The function's `layers`.

Type: Array of [Layer \(p. 680\)](#) objects

[MasterArn \(p. 602\)](#)

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 \(p. 602\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 602\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 602\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\\-/]+`

[Runtime \(p. 602\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs10.x` | `nodejs12.x` | `java8` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `dotnetcore2.1` | `dotnetcore3.1` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided`

[State \(p. 602\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 602\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 602\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Timeout \(p. 602\)](#)

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 602\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) object

[Version \(p. 602\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 602\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 693\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid.

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, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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 ensures that your function has capacity to process the specified number of events simultaneously, and prevents it from scaling beyond that level. Use [GetFunction \(p. 538\)](#) to see the current setting for a function.

Use [GetAccountSettings \(p. 529\)](#) 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 \(p. 608\)](#)

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 \(p. 608\)](#)

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 \(p. 609\)](#)

The number of concurrent executions that are 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.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource 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](#)

The request throughput limit was 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 V3](#)

PutFunctionEventInvokeConfig

Configures options for [asynchronous invocation](#) on a function, version, or alias. If a configuration already exists for a function, version, or alias, this operation overwrites it. If you exclude any settings, they are removed. To set one option without affecting existing settings for other options, use [UpdateFunctionEventInvokeConfig \(p. 652\)](#).

By default, Lambda retries an asynchronous invocation twice if the function returns an error. It retains events in a queue for up to six hours. When an event fails all processing attempts or stays in the asynchronous invocation queue for too long, Lambda discards it. To retain discarded events, configure a dead-letter queue with [UpdateFunctionConfiguration \(p. 643\)](#).

To send an invocation record to a queue, topic, function, or event bus, specify a [destination](#). You can configure separate destinations for successful invocations (on-success) and events that fail all processing attempts (on-failure). You can configure destinations in addition to or instead of a dead-letter queue.

Request Syntax

```
PUT /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 611)

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. 611)

A version number or alias name.

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.

[DestinationConfig \(p. 611\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

Required: No

[MaximumEventAgeInSeconds \(p. 611\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

[MaximumRetryAttempts \(p. 611\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "FunctionArn": "string",
    "LastModified": number,
```

```
"MaximumEventAgeInSeconds": number,  
"MaximumRetryAttempts": 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.

[DestinationConfig \(p. 612\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

[FunctionArn \(p. 612\)](#)

The Amazon Resource Name (ARN) of the 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 \(p. 612\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 612\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 612\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

PutProvisionedConcurrencyConfig

Adds a provisioned concurrency configuration to a function's alias or version.

Request Syntax

```
PUT /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
    "ProvisionedConcurrentExecutions": number
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 615\)](#)

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-_]+))?

[Qualifier \(p. 615\)](#)

The version number or alias name.

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.

[ProvisionedConcurrentExecutions \(p. 615\)](#)

The amount of provisioned concurrency to allocate for the version or alias.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "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.

[AllocatedProvisionedConcurrentExecutions \(p. 616\)](#)

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

[AvailableProvisionedConcurrentExecutions \(p. 616\)](#)

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

[LastModified \(p. 616\)](#)

The date and time that a user last updated the configuration, in [ISO 8601](#) format.

Type: String

[RequestedProvisionedConcurrentExecutions \(p. 616\)](#)

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

[Status \(p. 616\)](#)

The status of the allocation process.

Type: String

Valid Values: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 616\)](#)

For failed allocations, the reason that provisioned concurrency could not be allocated.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

RemoveLayerVersionPermission

Removes a statement from the permissions policy for a version of an [AWS Lambda layer](#). For more information, see [AddLayerVersionPermission \(p. 487\)](#).

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 \(p. 618\)](#)

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 \(p. 618\)](#)

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 \(p. 618\)](#)

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 \(p. 618\)](#)

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.

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 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

The request throughput limit was 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 V3](#)

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 \(p. 560\)](#).

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 \(p. 620\)](#)

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 \(p. 620\)](#)

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 \(p. 620\)](#)

Only update the policy if the revision ID matches the ID that's specified. Use this option to avoid modifying a policy that has changed since you last read it.

[StatementId \(p. 620\)](#)

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.

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 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

The request throughput limit was 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 V3](#)

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.

[ARN](#) (p. 622)

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. 622)

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.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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.

[ARN](#) (p. 624)

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. 624)

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.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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. 626)

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. 626)

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. 626)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 626\)](#)

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 \(p. 626\)](#)

Only update the alias if the revision ID matches the ID that's specified. Use this option to avoid modifying an alias that has changed since you last read it.

Type: String

Required: No

[RoutingConfig \(p. 626\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 661\)](#) 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 \(p. 627\)](#)

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 \(p. 627\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 627\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 627\)](#)

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 \(p. 627\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 627\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 661\)](#) object

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

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, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

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.

The following error handling options are only available for stream sources (DynamoDB and Kinesis):

- `BisectBatchOnFunctionError` - If the function returns an error, split the batch in two and retry.
- `DestinationConfig` - Send discarded records to an Amazon SQS queue or Amazon SNS topic.
- `MaximumRecordAgeInSeconds` - Discard records older than the specified age.
- `MaximumRetryAttempts` - Discard records after the specified number of retries.
- `ParallelizationFactor` - Process multiple batches from each shard concurrently.

Request Syntax

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "Enabled": boolean,
    "FunctionName": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number
}
```

URI Request Parameters

The request requires the following URI parameters.

UUID (p. 630)

The identifier of the event source mapping.

Request Body

The request accepts the following data in JSON format.

BatchSize (p. 630)

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

[BisectBatchOnFunctionError \(p. 630\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

[DestinationConfig \(p. 630\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

Required: No

[Enabled \(p. 630\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

[FunctionName \(p. 630\)](#)

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

[MaximumBatchingWindowInSeconds \(p. 630\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

[MaximumRecordAgeInSeconds \(p. 630\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

[MaximumRetryAttempts \(p. 630\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

[ParallelizationFactor \(p. 630\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "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 \(p. 632\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 632\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 632\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

[EventSourceArn \(p. 632\)](#)

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 \(p. 632\)](#)

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 \(p. 632\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 632\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 632\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 632\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 632\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 632\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 632\)](#)

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 \(p. 632\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 632\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

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 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

The request throughput limit was 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 V3](#)

UpdateFunctionCode

Updates a Lambda function's code.

The function's code is locked when you publish a version. You can't 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. 636)

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. 636)

Set to true to validate the request parameters and access permissions without modifying the function code.

Type: Boolean

Required: No

[Publish \(p. 636\)](#)

Set to true to publish a new version of the function after updating the code. This has the same effect as calling [PublishVersion \(p. 601\)](#) separately.

Type: Boolean

Required: No

[RevisionId \(p. 636\)](#)

Only update the function if the revision ID matches the ID that's specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[S3Bucket \(p. 636\)](#)

An Amazon S3 bucket in the same AWS 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 \(p. 636\)](#)

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[S3ObjectVersion \(p. 636\)](#)

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 \(p. 636\)](#)

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",  
    "LastUpdateStatus": "string",  
    "LastUpdateStatusReason": "string",  
    "LastUpdateStatusReasonCode": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "State": "string",  
    "StateReason": "string",  
    "StateReasonCode": "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 \(p. 637\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 637\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 637\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 663\)](#) object

[Description \(p. 637\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 637\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 667\)](#) object

[FunctionArn \(p. 637\)](#)

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 \(p. 637\)](#)

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 \(p. 637\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 637\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+\.*\.)|()`

[LastModified \(p. 637\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 637\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 637\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 637\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` |
`InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` |
`InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 637\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 680\)](#) objects

[MasterArn \(p. 637\)](#)

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 \(p. 637\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 637\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 637\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.]+`

[Runtime \(p. 637\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 637\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

[StateReason \(p. 637\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 637\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup

[Timeout \(p. 637\)](#)

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 637\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) object

[Version \(p. 637\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 637\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 693\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid.

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, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

UpdateFunctionConfiguration

Modify the version-specific settings of a Lambda function.

When you update a function, Lambda provisions an instance of the function and its supporting resources. If your function connects to a VPC, this process can take a minute. During this time, you can't modify the function, but you can still invoke it. The `LastUpdateStatus`, `LastUpdateStatusReason`, and `LastUpdateStatusReasonCode` fields in the response from [GetFunctionConfiguration \(p. 543\)](#) indicate when the update is complete and the function is processing events with the new configuration. For more information, see [Function States](#).

These settings can vary between versions of a function and are locked when you publish a version. You can't modify the configuration of a published version, only the unpublished version.

To configure function concurrency, use [PutFunctionConcurrency \(p. 608\)](#). To grant invoke permissions to an account or AWS service, use [AddPermission \(p. 490\)](#).

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. 643)

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 \(p. 643\)](#)

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 \(p. 663\)](#) object

Required: No

[Description \(p. 643\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 643\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 665\)](#) object

Required: No

[Handler \(p. 643\)](#)

The name of the method within your code that Lambda calls to execute your function. The format includes the file name. It 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: No

[KMSKeyArn \(p. 643\)](#)

The ARN of the AWS Key Management Service (AWS KMS) key that's used to encrypt your function's environment variables. If it's not provided, AWS Lambda uses a default service key.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-z0-9-.]+:[.]*|()|)`

Required: No

[Layers \(p. 643\)](#)

A list of [function layers](#) to add to the function's execution environment. Specify each layer by its 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 \(p. 643\)](#)

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

[RevisionId \(p. 643\)](#)

Only update the function if the revision ID matches the ID that's specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[Role \(p. 643\)](#)

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: No

[Runtime \(p. 643\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: `nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided`

Required: No

[Timeout \(p. 643\)](#)

The amount of time that Lambda allows a function to run before stopping it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 643\)](#)

Set Mode to Active to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 690\)](#) object

Required: No

[VpcConfig \(p. 643\)](#)

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 \(p. 692\)](#) 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",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "Timeout": "string"
}
```

```
"StateReasonCode": "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 \(p. 646\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 646\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 646\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 663\)](#) object

[Description \(p. 646\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 646\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 667\)](#) object

[FunctionArn \(p. 646\)](#)

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 \(p. 646\)](#)

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 \(p. 646\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 646\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:\.*|()`

[LastModified \(p. 646\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 646\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 646\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 646\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 646\)](#)

The function's `layers`.

Type: Array of [Layer \(p. 680\)](#) objects

[MasterArn \(p. 646\)](#)

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 \(p. 646\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 646\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 646\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\\-/]+

[Runtime \(p. 646\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 646\)](#)

The current state of the function. When the state is Inactive, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

[StateReason \(p. 646\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 646\)](#)

The reason code for the function's current state. When the code is Creating, you can't invoke or modify the function.

Type: String

Valid Values: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfRangeAddresses | InvalidSubnet | InvalidSecurityGroup

[Timeout \(p. 646\)](#)

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 646\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) object

[Version \(p. 646\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 646\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 693\)](#) object

Errors

InvalidOperationException

One of the parameters in the request is invalid.

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, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

UpdateFunctionEventInvokeConfig

Updates the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 611\)](#).

Request Syntax

```
POST /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 652\)](#)

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 \(p. 652\)](#)

A version number or alias name.

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.

[DestinationConfig \(p. 652\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

Required: No

[MaximumEventAgeInSeconds \(p. 652\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

[MaximumRetryAttempts \(p. 652\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "FunctionArn": "string",
  "LastModified": number,
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": 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.

[DestinationConfig \(p. 653\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

[FunctionArn \(p. 653\)](#)

The Amazon Resource Name (ARN) of the 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 \(p. 653\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 653\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 653\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource 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

The request throughput limit was 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 V3](#)

Data Types

The following data types are supported:

- [AccountLimit \(p. 657\)](#)
- [AccountUsage \(p. 658\)](#)
- [AliasConfiguration \(p. 659\)](#)
- [AliasRoutingConfiguration \(p. 661\)](#)
- [Concurrency \(p. 662\)](#)
- [DeadLetterConfig \(p. 663\)](#)
- [DestinationConfig \(p. 664\)](#)
- [Environment \(p. 665\)](#)
- [EnvironmentError \(p. 666\)](#)
- [EnvironmentResponse \(p. 667\)](#)
- [EventSourceMappingConfiguration \(p. 668\)](#)
- [FunctionCode \(p. 671\)](#)
- [FunctionCodeLocation \(p. 672\)](#)
- [FunctionConfiguration \(p. 673\)](#)
- [FunctionEventInvokeConfig \(p. 678\)](#)
- [Layer \(p. 680\)](#)
- [LayersListItem \(p. 681\)](#)
- [LayerVersionContentInput \(p. 682\)](#)
- [LayerVersionContentOutput \(p. 683\)](#)
- [LayerVersionsListItem \(p. 684\)](#)
- [OnFailure \(p. 686\)](#)
- [OnSuccess \(p. 687\)](#)
- [ProvisionedConcurrencyConfigListItem \(p. 688\)](#)

- [TracingConfig \(p. 690\)](#)
- [TracingConfigResponse \(p. 691\)](#)
- [VpcConfig \(p. 692\)](#)
- [VpcConfigResponse \(p. 693\)](#)

AccountLimit

Limits that are related to concurrency and storage. All file and storage sizes are in bytes.

Contents

CodeSizeUnzipped

The maximum size of a function's deployment package and layers when they're extracted.

Type: Long

Required: No

CodeSizeZipped

The maximum size of a deployment package when it's 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, minus the capacity that's reserved for individual functions with [PutFunctionConcurrency \(p. 608\)](#).

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 V3](#)

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, that's being used 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 V3](#)

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 \(p. 661\)](#) 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 V3](#)

AliasRoutingConfiguration

The [traffic-shifting](#) configuration of a Lambda function alias.

Contents

AdditionalVersionWeights

The second version, and the percentage of traffic that's 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 V3](#)

Concurrency

Contents

ReservedConcurrentExecutions

The number of concurrent executions that are 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 V3](#)

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 V3](#)

DestinationConfig

A configuration object that specifies the destination of an event after Lambda processes it.

Contents

OnFailure

The destination configuration for failed invocations.

Type: [OnFailure \(p. 686\)](#) object

Required: No

OnSuccess

The destination configuration for successful invocations.

Type: [OnSuccess \(p. 687\)](#) 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 V3](#)

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 V3](#)

EnvironmentError

Error messages for environment variables that couldn't 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 V3](#)

EnvironmentResponse

The results of an operation to update or read environment variables. If the operation is successful, the response contains the environment variables. If it failed, the response contains details about the error.

Contents

Error

Error messages for environment variables that couldn't be applied.

Type: [EnvironmentError \(p. 666\)](#) 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 V3](#)

EventSourceMappingConfiguration

A mapping between an AWS resource and an AWS Lambda function. See [CreateEventSourceMapping \(p. 498\)](#) 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

BisectBatchOnFunctionError

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

DestinationConfig

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 664\)](#) object

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, or its state changed, 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

MaximumBatchingWindowInSeconds

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

MaximumRecordAgeInSeconds

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

MaximumRetryAttempts

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

ParallelizationFactor

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

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

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

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 V3](#)

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 AWS 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 V3](#)

FunctionCodeLocation

Details about a function's deployment package.

Contents

Location

A presigned URL that you can use to download the deployment package.

Type: String

Required: No

RepositoryType

The service that's 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 V3](#)

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 \(p. 663\)](#) 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 \(p. 667\)](#) 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 that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: No

KMSKeyArn

The KMS key that's used to encrypt the function's environment variables. This key is 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

LastUpdateStatus

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

Required: No

LastUpdateStatusReason

The reason for the last update that was performed on the function.

Type: String

Required: No

LastUpdateStatusReasonCode

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalServerError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

Required: No

Layers

The function's [layers](#).

Type: Array of [Layer \(p. 680\)](#) objects

Required: No

MasterArn

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-_]+))?`

Required: No

MemorySize

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

RevisionId

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: `nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided`

Required: No

State

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending | Active | Inactive | Failed`

Required: No

StateReason

The reason for the function's current state.

Type: String

Required: No

StateReasonCode

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

Required: No

Timeout

The amount of time in seconds that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

TracingConfig

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 691\)](#) 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 \(p. 693\)](#) 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 V3

FunctionEventInvokeConfig

Contents

DestinationConfig

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 664\)](#) object

Required: No

FunctionArn

The Amazon Resource Name (ARN) of the 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 and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

Required: No

MaximumEventAgeInSeconds

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

MaximumRetryAttempts

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

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 V3](#)

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 V3](#)

LayersListItem

Details about an [AWS Lambda layer](#).

Contents

LatestMatchingVersion

The newest version of the layer.

Type: [LayerVersionsListItem \(p. 684\)](#) 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 V3](#)

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 V3](#)

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 V3](#)

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: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | 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 V3](#)

OnFailure

A destination for events that failed processing.

Contents

Destination

The Amazon Resource Name (ARN) of the destination resource.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

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 V3](#)

OnSuccess

A destination for events that were processed successfully.

Contents

Destination

The Amazon Resource Name (ARN) of the destination resource.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

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 V3](#)

ProvisionedConcurrencyConfigListItem

Details about the provisioned concurrency configuration for a function alias or version.

Contents

AllocatedProvisionedConcurrentExecutions

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

AvailableProvisionedConcurrentExecutions

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

FunctionArn

The Amazon Resource Name (ARN) of the alias or version.

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 and time that a user last updated the configuration, in [ISO 8601 format](#).

Type: String

Required: No

RequestedProvisionedConcurrentExecutions

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

Status

The status of the allocation process.

Type: String

Valid Values: `IN_PROGRESS` | `READY` | `FAILED`

Required: No

StatusReason

For failed allocations, the reason that provisioned concurrency could not be allocated.

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 V3](#)

TracingConfig

The function's AWS X-Ray tracing configuration. To sample and record incoming requests, set `Mode` to `Active`.

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 V3](#)

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 V3](#)

VpcConfig

The VPC security groups and subnets that are attached to a Lambda function. For more information, see [VPC Settings](#).

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 V3](#)

VpcConfigResponse

The VPC security groups and subnets that are 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 V3](#)

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](#).

Glossaire AWS

Pour la terminologie AWS la plus récente, consultez le [Glossaire AWS](#) dans le document AWS General Reference.