

---

# AWS Lambda

## 개발자 가이드



## AWS Lambda: 개발자 가이드

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

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

## Table of Contents

AWS Lambda란 무엇입니까?	1
언제 AWS Lambda를 사용해야 합니까?	1
AWS Lambda를 처음 사용하십니까?	2
시작하기	3
함수 만들기	3
디자이너 사용	3
Lambda 함수 호출	4
개념	5
명령행 도구	6
AWS CLI 설정	6
AWS Serverless Application Model CLI	6
제한	7
권한	9
실행 역할	9
리소스 기반 정책	10
함수에 AWS 서비스에 대한 액세스 권한 부여	11
함수에 다른 계정에 대한 액세스 권한 부여	12
계층에 다른 계정에 대한 액세스 권한 부여	12
리소스 기반 정책 정리	13
사용자 정책	13
함수 개발	14
계층 개발 및 사용	17
교차 계정 역할	18
리소스와 조건	18
함수	19
이벤트 소스 맵핑	20
계층	21
Lambda 함수	22
Lambda 함수 빌드	22
Lambda 함수에 대한 코드 작성	22
코드 배포 및 Lambda 함수 생성	23
모니터링 및 문제 해결	24
코드 편집기	24
파일 및 폴더 작업	25
코드 작업	27
메뉴 모음 사용	29
전체 화면 모드에서 작업	30
기본 설정 작업	30
명령 작업	30
프로그래밍 모델	31
배포 패키지	32
Lambda 배포 패키지에 대한 권한 정책	32
AWS 리소스에 액세스	33
AWS 제품 액세스	33
비 AWS 서비스 액세스	34
프라이빗 서비스 또는 리소스 액세스	34
함수 구성	35
함수 구성	35
동시성 관리	37
계정 수준 동시 실행 한도	37
함수 수준 동시 실행 한도	37
조절 동작	39
동시 사용 모니터링	39
환경 변수	39

설정	40
환경 변수의 이름을 지정하기 위한 규칙	41
환경 변수 및 함수 버전 관리	42
환경 변수 암호화	42
환경 변수를 사용하여 Lambda 함수 생성	43
민감한 정보를 저장하도록 환경 변수를 사용하여 Lambda 함수 생성	44
버전 관리	45
버전 관리	47
별칭	50
리소스 정책	57
버전 관리	58
별칭을 사용한 트래픽 이동	60
계층	62
계층을 사용하기 위한 함수 구성	62
계층 관리	63
계층 내 라이브러리 종속 항목들을 포함	65
계층 권한	66
VPC 설정	66
Amazon VPC 액세스를 위한 Lambda 함수 구성	67
Lambda 함수에 대한 인터넷 액세스	67
VPC 활성화 Lambda 함수 설정에 대한 지침	68
자습서: Amazon ElastiCache	68
자습서: Amazon RDS	71
태그 지정	74
결제용 Lambda 함수 태그 지정	74
콘솔을 사용하여 Lambda 함수에 태그 적용	75
CLI를 사용하여 Lambda 함수에 태그 적용	75
태그 지정된 Lambda 함수 필터링	75
태그 제한	77
함수 호출	78
예제 1: Amazon S3가 이벤트를 푸시하고 Lambda 함수를 호출	78
예제 2: AWS Lambda가 Kinesis 스트림에서 이벤트를 가져와서 Lambda 함수 호출	79
호출 유형	80
이벤트 소스 맵핑	80
AWS 서비스의 이벤트 소스 맵핑	81
AWS 폴 기반 서비스의 이벤트 소스 맵핑	81
사용자 지정 애플리케이션의 이벤트 소스 맵핑	82
재시도 동작	83
확장	84
요청 속도	85
자동 조정	85
배달 못한 편지 대기열	86
AWS CLI	87
사전 조건	87
실행 역할 만들기	88
함수 만들기	88
계정의 Lambda 함수 목록 표시	89
정리	90
Android용 Mobile SDK	90
자습서	91
샘플 코드	96
Lambda 런타임	99
환경 변수	100
실행 콘텍스트	101
런타임 지원 정책	102
사용자 지정 런타임	103
사용자 지정 런타임 사용	103

사용자 지정 런타임 빌드 .....	103
런타임 인터페이스 .....	105
다음 호출 .....	105
호출 응답 .....	106
호출 오류 .....	106
초기화 오류 .....	107
자습서 – 사용자 지정 런타임 .....	107
사전 조건 .....	107
함수 만들기 .....	108
계층 생성 .....	109
함수 업데이트 .....	110
런타임 업데이트 .....	111
계층 공유 .....	111
정리 .....	112
Lambda 애플리케이션 .....	113
애플리케이션 관리 .....	113
애플리케이션 모니터링 .....	114
사용자 지정 대시보드 .....	114
AWS SAM .....	116
샘플 – 오류 처리자 .....	116
아키텍처 및 이벤트 구조 .....	117
AWS X-Ray를 사용하여 계층 .....	117
AWS CloudFormation 템플릿과 추가 리소스 .....	118
지속적 전송(CD) .....	119
사전 조건 .....	120
AWS CloudFormation 역할 생성 .....	120
리포지토리 설정 .....	121
파이프라인 만들기 .....	122
빌드 단계 역할 업데이트 .....	123
배포 단계 완료 .....	123
모범 사례 .....	124
함수 코드 .....	124
함수 구성 .....	125
경보 및 지표 .....	125
스트림 이벤트 호출 .....	126
비동기 호출 .....	126
Lambda VPC .....	126
다른 서비스와 함께 사용 .....	128
Application Load Balancer .....	129
Alexa .....	131
Amazon API Gateway .....	131
자습서 .....	133
샘플 코드 .....	141
マイ크로서비스 블루프린트 .....	144
샘플 템플릿 .....	145
AWS CloudTrail .....	145
자습서 .....	147
샘플 코드 .....	152
CloudWatch 이벤트 .....	153
자습서 .....	154
샘플 템플릿 .....	157
예약 표현식 .....	158
Amazon CloudWatch Logs .....	158
AWS CloudFormation .....	159
CloudFront .....	161
AWS CodeCommit .....	163
Amazon Cognito .....	163

AWS Config .....	164
Amazon DynamoDB .....	165
이벤트 소스 매핑 생성 .....	166
실행 역할 권한 .....	167
자습서 .....	167
샘플 코드 .....	171
샘플 템플릿 .....	174
Kinesis .....	175
데이터 스트림과 함수 구성 .....	176
이벤트 소스 매핑 생성 .....	177
이벤트 소스 매핑 API .....	178
실행 역할 권한 .....	178
Amazon CloudWatch 측정치 .....	179
자습서 .....	179
샘플 코드 .....	183
샘플 템플릿 .....	185
Kinesis Data Firehose .....	187
Amazon Lex .....	188
Amazon S3 .....	188
자습서 .....	190
샘플 코드 .....	197
샘플 템플릿 .....	203
Amazon SES .....	203
Amazon SNS .....	205
자습서 .....	206
샘플 코드 .....	209
Amazon SQS .....	211
Lambda에서 사용할 수 있도록 대기열 구성 .....	212
대기열을 이벤트 소스로 구성 .....	212
실행 역할 권한 .....	213
자습서 .....	213
샘플 코드 .....	216
샘플 템플릿 .....	219
모니터링 .....	220
사용 Amazon CloudWatch .....	220
AWS Lambda 문제 해결 시나리오 .....	220
모니터링 그래프 .....	222
CloudWatch 로그에 액세스 .....	222
Lambda 지표 .....	223
AWS X-Ray 사용 .....	226
AWS X-Ray를 통한 Lambda 기반 애플리케이션 추적 .....	226
Lambda를 통한 AWS X-Ray 설정 .....	227
Lambda 함수에서 트레이스 세그먼트 방출 .....	229
Lambda 환경의 AWS X-Ray 데몬 .....	229
환경 변수를 사용하여 AWS X-Ray와 통신 .....	229
AWS X-Ray 콘솔의 Lambda 트레이스: 예제 .....	230
CloudTrail 사용 .....	231
CloudTrail의 AWS Lambda 정보 .....	232
AWS Lambda 로그 파일 항목 이해 .....	233
CloudTrail을 사용하여 함수 호출 추적 .....	234
Node.js 작업 .....	235
배포 패키지 .....	235
핸들러 .....	236
콜백 파라미터 사용 .....	237
예 .....	237
콘텍스트 .....	238
로깅 .....	239

오류 .....	240
함수 오류 처리 .....	242
추적 .....	243
Python 작업 .....	245
배포 패키지 .....	245
추가 종속 프로그램 제외 .....	246
추가 종속 프로그램 포함 .....	246
가상 환경 포함 .....	247
핸들러 .....	249
콘텍스트 .....	250
로깅 .....	251
오류 .....	252
함수 오류 처리 .....	254
추적 .....	254
Java 작업 .....	257
배포 패키지 .....	258
Maven .....	258
Maven 및 Eclipse .....	260
Gradle 및 ZIP .....	262
Eclipse IDE 및 AWS SDK 플러그인 .....	265
핸들러 .....	265
핸들러 오버로드 해결 .....	266
추가 정보 .....	266
핸들러 입력/출력 유형(Java) .....	267
핸들러를 생성하기 위해 사전 정의된 인터페이스 활용(Java) .....	271
컨텍스트 .....	274
로깅 .....	275
LambdaLogger .....	276
Log4j 2용 사용자 지정 Appender .....	277
오류 .....	278
함수 오류 처리 .....	279
추적 .....	280
예제 함수 .....	281
Go로 작업 .....	283
배포 패키지 .....	283
Windows에서 배포 패키지 만들기 .....	284
핸들러 .....	284
핸들러 .....	285
글로벌 상태 사용 .....	287
콘텍스트 .....	288
호출 콘텍스트 정보 액세스 .....	288
로깅 .....	289
오류 .....	291
함수 오류 처리 .....	294
예기치 않은 오류 처리 .....	292
추적 .....	293
Go용 X-Ray SDK 설치 .....	293
Go용 X-Ray SDK 구성 .....	293
하위 세그먼트 생성 .....	293
Capture .....	293
HTTP 요청 추적 .....	294
환경 변수 .....	294
C#을 사용한 작업 .....	295
배포 패키지 .....	295
.NET Core CLI .....	296
AWS Toolkit for Visual Studio .....	303
핸들러 .....	304

스트림 처리 .....	304
표준 데이터 유형 처리 .....	305
핸들러 서명 .....	306
Lambda 함수 핸들러 제한 사항 .....	306
AWS Lambda에서 C# 함수로 작성된 비동기식 핸들러 사용 .....	307
콘텍스트 .....	308
로깅 .....	308
오류 .....	310
함수 오류 처리 .....	312
PowerShell을 사용하여 작업 .....	314
배포 패키지 .....	314
PowerShell 개발 환경 설정 .....	315
AWSLambdaPSCore 모듈 사용 .....	315
핸들러 .....	317
데이터 반환 .....	318
콘텍스트 .....	318
로깅 .....	319
오류 .....	320
함수 오류 처리 .....	320
Ruby로 작업 .....	322
핸들러 .....	323
배포 패키지 .....	324
종속 프로그램이 없는 함수를 업데이트 .....	324
추가 종속 프로그램이 있는 함수를 업데이트 .....	325
콘텍스트 .....	326
로깅 .....	326
오류 .....	327
API 참조 .....	329
Actions .....	329
AddLayerVersionPermission .....	331
AddPermission .....	335
CreateAlias .....	339
CreateEventSourceMapping .....	343
CreateFunction .....	347
DeleteAlias .....	355
DeleteEventSourceMapping .....	357
DeleteFunction .....	360
DeleteFunctionConcurrency .....	362
DeleteLayerVersion .....	364
GetAccountSettings .....	366
GetAlias .....	368
GetEventSourceMapping .....	371
GetFunction .....	374
GetFunctionConfiguration .....	377
GetLayerVersion .....	382
GetLayerVersionByArn .....	385
GetLayerVersionPolicy .....	388
GetPolicy .....	390
Invoke .....	392
InvokeAsync .....	397
ListAliases .....	399
ListEventSourceMappings .....	402
ListFunctions .....	405
ListLayers .....	408
ListLayerVersions .....	410
ListTags .....	413
ListVersionsByFunction .....	415

PublishLayerVersion .....	418
PublishVersion .....	422
PutFunctionConcurrency .....	428
RemoveLayerVersionPermission .....	431
RemovePermission .....	433
TagResource .....	436
UntagResource .....	438
UpdateAlias .....	440
UpdateEventSourceMapping .....	444
UpdateFunctionCode .....	448
UpdateFunctionConfiguration .....	455
<b>Data Types .....</b>	<b>462</b>
AccountLimit .....	463
AccountUsage .....	464
AliasConfiguration .....	465
AliasRoutingConfiguration .....	467
Concurrency .....	468
DeadLetterConfig .....	469
Environment .....	470
EnvironmentError .....	471
EnvironmentResponse .....	472
EventSourceMappingConfiguration .....	473
FunctionCode .....	475
FunctionCodeLocation .....	476
FunctionConfiguration .....	477
Layer .....	481
LayersListItem .....	482
LayerVersionContentInput .....	483
LayerVersionContentOutput .....	484
LayerVersionsListItem .....	485
TracingConfig .....	487
TracingConfigResponse .....	488
VpcConfig .....	489
VpcConfigResponse .....	490
SDK를 사용할 때의 인증서 오류 .....	490
릴리스 .....	492
이전 업데이트 .....	494
AWS Glossary .....	499

# AWS Lambda란 무엇입니까?

AWS Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 해주는 컴퓨팅 서비스입니다. AWS Lambda는 필요 시에만 코드를 실행하며, 하루에 몇 개의 요청에서 조당 수천 개의 요청까지 자동으로 확장이 가능합니다. 사용한 컴퓨팅 시간에 대해서만 요금을 지불하면 되고 코드가 실행되지 않을 때는 요금이 부과되지 않습니다. AWS Lambda에서는 사실상 모든 유형의 애플리케이션이나 백엔드 서비스에 대한 코드를 별도의 관리 없이 실행할 수 있습니다. AWS Lambda는 고가용성 컴퓨팅 인프라에서 코드를 실행하고 서버 및 운영 체제 유지 관리, 용량 프로비저닝 및 자동 조정, 코드 및 보안 패치 배포, 코드 모니터링 및 로깅 등 모든 컴퓨팅 리소스 관리를 수행합니다. [AWS Lambda가 지원하는 언어 \(p. 99\)](#) 중 하나로 코드를 공급하기만 하면 됩니다.

AWS Lambda를 사용하여 Amazon S3 버킷 또는 Amazon DynamoDB 테이블의 데이터 변경과 같은 이벤트에 대한 응답으로 코드를 실행할 수 있습니다. Amazon API Gateway를 사용하여 HTTP 요청에 대한 응답으로 코드를 실행할 수도 있으며, 또는 AWS SDK를 사용하여 만든 API 호출을 통해 코드를 호출할 수 있습니다. 이러한 기능을 제공하므로 Lambda를 사용하여 Amazon S3 및 Amazon DynamoDB와 같은 AWS 서비스에 대한 데이터 처리 트리거를 손쉽게 빌드하거나, Kinesis에 저장된 스트리밍 데이터를 처리하거나 AWS 규모, 성능, 보안에 따라 작동하는 자체 백엔드를 생성할 수 있습니다.

또한 이벤트에 의해 트리거되고 CodePipeline 및 AWS CodeBuild를 사용하여 자동으로 배포하는 함수로 구성된 [서버리스](#) 애플리케이션을 빌드할 수 있습니다. 자세한 내용은 [AWS Lambda 애플리케이션 \(p. 113\)](#) 단원을 참조하십시오.

AWS Lambda 실행 환경에 대한 자세한 내용은 [AWS Lambda 런타임 \(p. 99\)](#) 단원을 참조하십시오. AWS Lambda가 코드를 실행하는 데 필요한 컴퓨팅 리소스를 확인하는 방법에 대한 내용은 [AWS Lambda 함수 구성 \(p. 35\)](#) 단원을 참조하십시오.

## 언제 AWS Lambda를 사용해야 합니까?

AWS Lambda는 AWS Lambda에서 지원되는 언어로 애플리케이션 코드를 작성하고, AWS Lambda 표준 런타임 환경 및 Lambda에서 제공된 리소스 내에서 실행할 수 있는 경우에 많은 애플리케이션 시나리오를 위한 이상적인 컴퓨팅 플랫폼입니다.

AWS Lambda를 사용하면 사용자는 자신의 코드에 대해서만 책임을 갖습니다. AWS Lambda는 메모리, CPU, 네트워크 및 기타 리소스의 균형을 제공하는 컴퓨팅 플랫폼을 관리합니다. 이는 유연성을 대신하는 것으로, 로그인하여 인스턴스를 컴퓨팅하거나 운영 체제 또는 언어 런타임을 사용자 지정할 수 없습니다. 이러한 제약 조건을 통해 AWS Lambda는 용량 프로비저닝, 플랫폼 상태 모니터링, 보안 패치 적용, 코드 배포, Lambda 함수 모니터링 및 로깅과 같은 운영 및 관리 작업을 수행할 수 있습니다.

자체 컴퓨팅 리소스를 관리해야 하는 경우 Amazon Web Services는 사용자의 요구를 충족하는 다른 컴퓨팅 서비스도 제공합니다.

- Amazon Elastic Compute Cloud(Amazon EC2) 서비스는 유연성 및 선택의 폭이 넓은 다양한 EC2 인스턴스 유형을 제공합니다. 이는 운영 체제, 네트워크 및 보안 설정 및 전체 소프트웨어 스택을 사용자 지정할 수 있는 옵션을 제공하지만, 사용자는 용량 프로비저닝, 플랫폼 상태 및 성능 모니터링, 내결합성을 위한 가용 영역 사용에 대한 책임을 갖습니다.
- Elastic Beanstalk은 사용자가 기본 EC2 인스턴스에 대한 소유권과 완전한 제어권을 보유하고 있는 Amazon EC2에 애플리케이션을 배포 및 확장하기 위한 편리한 서비스를 제공합니다.

Lambda는 고가용 서비스입니다. 자세한 정보는 [AWS Lambda 서비스 수준 계약\(SLA\)](#)을 참조하십시오.

# AWS Lambda를 처음 사용하십니까?

AWS Lambda를 처음 사용한다면, 다음 단원을 순서대로 읽어보기를 권장합니다.

- 제품 개요를 읽고 소개 동영상을 시청하여 샘플 사용 사례를 이해하십시오. 이 리소스는 [AWS Lambda 웹 페이지](#)에서 사용 가능합니다.
- 이 안내서의 [Lambda 함수 \(p. 22\)](#) 단원을 읽어 보십시오. Lambda 함수의 프로그래밍 모델 및 배포 옵션을 이해하려면 몇 가지 핵심 개념을 숙지해야 합니다. 이 단원에서는 이러한 개념을 설명하고 함수 코드를 작성하는 데 사용할 수 있는 다양한 언어로 작업하는 방법에 대해 자세히 설명합니다.
- 콘솔 기반 시작하기 연습을 수행해 보십시오. 이 연습에서는 콘솔을 사용하여 첫 번째 Lambda 함수를 만들고 테스트할 수 있는 지침을 제공합니다. 또한 콘솔에서 제공되는 블루프린트로 함수를 빠르게 생성하는 방법을 알아봅니다. 자세한 내용은 [AWS Lambda 시작하기 \(p. 3\)](#) 단원을 참조하십시오.
- 이 안내서의 [AWS Lambda를 사용한 애플리케이션 배포 \(p. 113\)](#) 단원을 읽어 보십시오. 이 단원에서는 종합적 경험 생성에 사용하는 다양한 AWS Lambda 구성 요소를 소개합니다.

시작하기 연습 이외에도 다양한 사용 사례를 살펴볼 수 있으며 각 시나리오에는 예제 시나리오를 안내하는 자습서가 제공됩니다. 애플리케이션 요구 사항(예: 이벤트 중심 Lambda 함수 호출 또는 온디マン드 호출을 원하는지 여부)에 따라 특정 요구 사항을 충족하는 특정 자습서를 수행할 수 있습니다. 자세한 내용은 [다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#) 단원을 참조하십시오.

다음 주제에서는 AWS Lambda에 대한 추가 정보를 제공합니다.

- [AWS Lambda 함수 버전 관리 및 별칭 \(p. 45\)](#)
- [사용 Amazon CloudWatch \(p. 220\)](#)
- [AWS Lambda 함수 작업의 모범 사례 \(p. 124\)](#)
- [AWS Lambda 한도 \(p. 7\)](#)

# AWS Lambda 시작하기

AWS Lambda 사용을 시작하려면 Lambda 콘솔에서 함수를 생성해야 합니다. 몇 분 안에 함수를 만들고, 호출하고, 로그와 지표를 확인하고, 데이터를 추적할 수 있습니다.

Lambda 및 기타 AWS 서비스를 사용하려면 AWS 계정이 필요합니다. 계정이 없는 경우 [aws.amazon.com](https://aws.amazon.com)으로 이동하여 AWS 계정 생성을 선택합니다. 자세한 지침은 [AWS 계정 생성 및 활성화](#)를 참조하십시오.

모범 사례에 따르면, 관리자 권한이 있는 AWS Identity and Access Management(IAM) 사용자도 만들고 루트 자격 증명이 필요 없는 모든 작업은 그 사용자로 처리해야 합니다. 콘솔 액세스를 위한 암호 및 명령줄 도구를 사용하기 위한 액세스 키를 만듭니다. 지침은 IAM 사용 설명서의 [첫 번째 IAM 관리 사용자 및 그룹 생성 단원](#)을 참조하십시오.

## 섹션

- [콘솔로 Lambda 함수 만들기 \(p. 3\)](#)
- [AWS Lambda 개념 \(p. 5\)](#)
- [명령행 도구 \(p. 6\)](#)
- [AWS Lambda 한도 \(p. 7\)](#)

## 콘솔로 Lambda 함수 만들기

이 시작하기 실습에서는 AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성합니다. 다음으로, 샘플 이벤트 데이터를 사용하여 Lambda 함수를 수동으로 호출합니다. AWS Lambda에서는 Lambda 함수를 실행해 결과를 반환합니다. 그런 다음 Lambda 함수가 생성한 로그와 다양한 CloudWatch 측정치를 포함하여 실행 결과를 확인합니다.

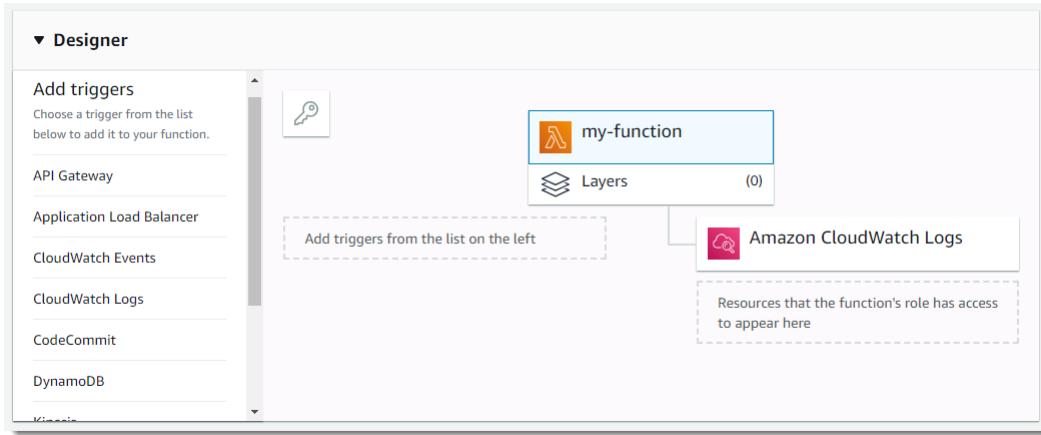
Lambda 함수를 만들려면

1. [AWS Lambda 콘솔](#)을 엽니다.
2. [Create a function]을 선택합니다.
3. 함수 이름에 **my-function**을 입력합니다.
4. 함수 생성을 선택합니다.

Lambda는 Node.js 함수와, 함수에 로그를 업로드할 수 있는 권한을 부여하는 실행 역할을 생성합니다. 함수를 실행하면 Lambda가 실행 역할을 수임하고 이를 사용하여 AWS SDK에 대한 자격 증명을 생성하고 이벤트 소스로부터 데이터를 읽습니다.

## 디자이너 사용

디자이너를 사용하여 트리거를 구성하고 권한을 볼 수 있습니다.



실행 역할이 함수에 부여하는 로그 관련 권한을 보려면 Amazon CloudWatch Logs를 선택합니다. 추가 권한이 필요한 기능을 트리거하거나 구성하면 Lambda가 함수의 실행 역할이나 리소스 기반 정책을 수정하여 최소 필수 액세스 권한을 부여합니다. 이러한 정책을 보려면 키 아이콘을 선택합니다.

디자이너에서 my-function을 선택하여 함수의 코드 및 구성으로 돌아갑니다. 스크립트 언어의 경우 Lambda는 성공 응답을 반환하는 샘플 코드를 포함합니다. 소스 코드가 3 MB 한도를 초과하지 않는 경우, 내장된 AWS Cloud9 편집기를 사용하여 함수 코드를 편집할 수 있습니다.

## Lambda 함수 호출

콘솔에서 제공된 샘플 이벤트 데이터를 사용하여 Lambda 함수를 호출합니다.

함수를 호출하려면

1. 오른쪽 상단 모서리에서 테스트를 선택합니다.
2. [Configure test event] 페이지에서 [Create new test event]를 선택하고 [Event template]에서 기본값 [Hello World] 옵션을 그대로 둡니다. [Event name]을 입력하고 다음 샘플 이벤트 템플릿에 유의하십시오.

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

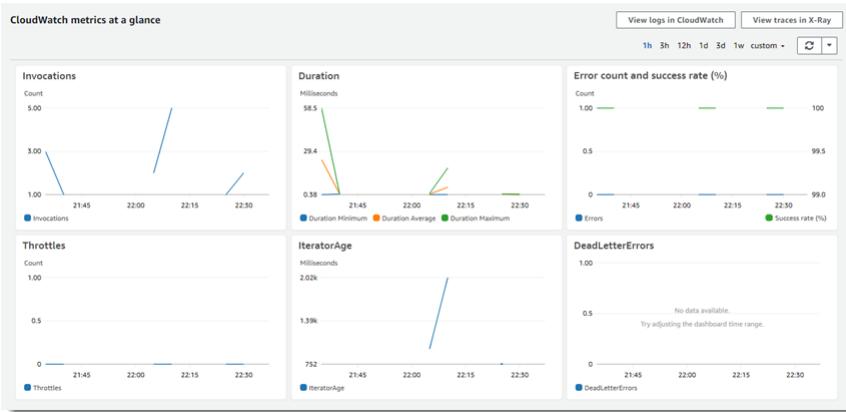
샘플 JSON에서 키와 값을 변경할 수 있지만, 이벤트 구조는 변경할 수 없습니다. 키와 값을 변경한 경우에는 이에 따라 반드시 샘플 코드를 업데이트해야 합니다.

3. 생성을 선택한 다음 테스트를 선택합니다. 각 사용자는 함수당 최대 10개의 테스트 이벤트를 생성할 수 있습니다. 이러한 테스트 이벤트는 다른 사용자가 사용할 수 없습니다.
4. AWS Lambda는 사용자를 대신하여 함수를 실행합니다. Lambda 함수의 `handler`는 샘플 이벤트를 수신하여 처리합니다.
5. 실행 성공 시, 콘솔에서 결과를 확인합니다.

- [Execution result] 섹션은 실행 상태를 [succeeded]로 표시하고, `return` 문이 반환한 함수 실행 결과를 표시합니다.
- [Summary] 섹션에 [Log output] 섹션(실행 로그의 REPORT 줄)에 보고된 주요 정보가 표시됩니다.
- 로그 출력 섹션에 AWS Lambda가 각 실행마다 생성하는 로그가 표시됩니다. 이들은 Lambda 함수가 CloudWatch에 기록한 로그입니다. 콘솔은 사용자의 편의를 위해 이러한 로그를 표시합니다.

여기지를 클릭하십시오 링크를 누르면 CloudWatch 콘솔에 로그가 표시됩니다. 해당 함수는 Amazon CloudWatch에서 Lambda 함수에 해당하는 로그 그룹에 로그를 추가합니다.

6. 몇 차례 Lambda 함수를 실행하여 다음 단계에서 볼 수 있는 몇 가지 측정치를 수집합니다.
7. [Monitoring]을 선택합니다. 이 페이지에는 Lambda가 CloudWatch로 보내는 지표의 그래프가 표시됩니다.



이러한 그래프에 대한 자세한 내용은 [AWS Lambda에 대한 Amazon CloudWatch 지표 액세스 \(p. 222\)](#) 단원을 참조하십시오.

## AWS Lambda 개념

AWS Lambda를 사용하면 서비스 환경에서 함수를 실행하여 원하는 언어로 이벤트를 처리할 수 있습니다. 함수의 각 인스턴스는 별도의 실행 컨텍스트에서 실행되며 이벤트를 한 번에 하나씩만 처리합니다. 이벤트 처리가 끝나면 응답을 반환하고, 그러면 Lambda는 다른 이벤트를 보내 줍니다. Lambda는 대량의 이벤트를 처리하기 위해 함수 인스턴스 수를 자동으로 늘립니다.

- **함수** – AWS Lambda에서 실행되는 스크립트 또는 프로그램. Lambda는 함수에 호출 이벤트를 전달합니다. 이 함수는 이벤트를 처리하고 응답을 반환합니다. 자세한 내용은 [Lambda 함수 작업 \(p. 22\)](#) 단원을 참조하십시오.
- **런타임** – Lambda 런타임을 사용하면 서로 다른 언어의 함수들을 동일한 기본 실행 환경에서 실행할 수 있습니다. 프로그래밍 언어와 일치하는 런타임을 사용하도록 함수를 구성합니다. 런타임은 Lambda 서비스와 함수 코드 사이에 위치하며 호출 이벤트, 컨텍스트 정보 및 이들 사이의 응답을 각각 중계합니다. Lambda에서 제공하는 런타임을 사용하거나 나만의 런타임을 빌드할 수 있습니다. 자세한 내용은 [AWS Lambda 런타임 \(p. 99\)](#) 단원을 참조하십시오.
- **계층** – Lambda 계층은 라이브러리, 사용자 지정 런타임 및 그 외 함수 종속성에 대한 배포 메커니즘입니다. 계층을 사용하면 개발 중인 함수 코드를 변경되지 않는 코드 및 리소스와는 별도로 관리할 수 있습니다. 사용자가 만든 계층, AWS에서 제공하는 계층 또는 다른 AWS 고객들의 계층을 사용하도록 함수를 구성할 수 있습니다. 자세한 내용은 [AWS Lambda 계층 \(p. 62\)](#) 단원을 참조하십시오.
- **이벤트 소스** – 함수를 트리거하고 함수의 로직을 실행하는 AWS 서비스(예: Amazon SNS)이거나 사용자 지정 서비스입니다. 자세한 내용은 [AWS Lambda 이벤트 소스 매핑 \(p. 80\)](#) 단원을 참조하십시오.
- **다운스트림 리소스** – 트리거 시 Lambda 함수가 호출하는 DynamoDB 테이블이나 Amazon S3 버킷 같은 AWS 서비스입니다.
- **로그 스트림** – Lambda는 자동으로 함수 호출을 모니터링하고 CloudWatch에 측정치를 보고하지만, 함수 코드에 사용자 지정 로깅 문으로 주석을 달면 Lambda 함수의 실행 흐름 및 성능을 분석해 함수가 제대로 작동하는지 확인할 수 있습니다.
- **AWS SAM** – [서비스 애플리케이션](#)을 정의하기 위한 모델입니다. AWS SAM은 기본적으로 AWS CloudFormation에서 지원되며 서비스 리소스를 표현하기 위해 단순화된 구문을 정의합니다. 자세한 내

용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM이란 무엇입니까?](#) 단원을 참조하십시오.

## 명령행 도구

명령줄에서 Lambda 함수를 관리하고 사용하려면 AWS Command Line Interface를 설치합니다. 이 가이드의 자습서에서는 모든 Lambda API 작업의 명령이 있는 AWS CLI를 사용합니다. 일부 기능은 Lambda 콘솔에서만 사용할 수 있고 AWS CLI 또는 AWS SDK를 통해서만 액세스할 수 있습니다.

AWS SAM CLI는 AWS SAM 애플리케이션을 관리하고 테스트할 때 사용할 수 있는 별도의 명령줄 도구입니다. 아티팩트 업로드 명령과 AWS CloudFormation 스택 시작 명령 등 AWS CLI에서 사용 가능한 명령 외에, SAM CLI는 템플릿을 확인하고 도커 컨테이너에서 애플리케이션을 로컬로 실행하는 등 추가 명령을 제공합니다.

## AWS CLI 설정

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 지침은 AWS Command Line Interface 사용 설명서에서 다음 항목을 참조하십시오.
  - [AWS Command Line Interface를 이용한 설정](#)
  - [AWS Command Line Interface 구성](#)
2. [AWS CLI 구성 파일](#)에서 관리자 사용자의 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 이 프로파일의 생성에 대한 자세한 내용은 [명명된 프로파일](#) 단원을 참조하십시오.

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

사용 가능한 AWS 리전 목록은 Amazon Web Services 일반 참조의 [리전 및 엔드포인트](#)를 참조하십시오.

3. 명령 프롬프트에 다음 명령을 입력하여 설정을 확인합니다.
  - help 명령을 실행하여 AWS CLI가 컴퓨터에 설치되어 있는지 확인합니다.

```
$ aws help
```

- Lambda 명령을 실행해 사용자가 AWS Lambda에 접속할 수 있는지 확인합니다. 이 명령은 계정의 함수를 나열합니다(있는 경우). AWS CLI는 adminuser 자격 증명을 사용하여 요청을 인증합니다.

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

## AWS Serverless Application Model CLI

AWS SAM CLI는 AWS SAM 템플릿 및 애플리케이션에서 작동하는 명령줄 도구입니다. AWS SAM CLI를 사용하여, Lambda 함수를 로컬에서 호출하고, 서비스 애플리케이션을 위한 배포 패키지를 생성하며, 서비스 애플리케이션을 AWS 클라우드에 배포할 수 있습니다.

AWS SAM CLI 설치에 대한 자세한 내용은 AWS Serverless Application Model 개발자 안내서에서 [AWS SAM CLI 설치](#)를 참조하십시오.

## AWS Lambda 한도

AWS Lambda는 함수를 실행하고 저장하는 데 사용할 수 있는 컴퓨팅 및 스토리지 리소스의 양을 제한합니다. 리전당 다음과 같은 한도가 적용되며 이 한도를 늘릴 수 있습니다. 한도 증가를 요청하려면 [AWS 지원 센터 콘솔](#)을 이용하십시오.

리소스	기본 제한
<a href="#">동시 실행 (p. 37)</a>	1,000
<a href="#">함수 및 계층 스토리지</a>	75 GB

Lambda가 트래픽에 대한 응답으로 함수 동시성을 확장하는 방법에 대한 자세한 내용은 [조정 동작 이해 \(p. 84\)](#) 단원을 참조하십시오.

함수 구성, 배포, 실행에는 다음 제한이 적용됩니다. 변경할 수 없습니다.

리소스	한도
<a href="#">함수 메모리 할당 (p. 35)</a>	128MB ~ 3,008 MB, 64MB씩 증분됨
<a href="#">함수 제한 시간 (p. 35)</a>	900초 (15 minutes)
<a href="#">함수 환경 변수 (p. 39)</a>	4 KB
<a href="#">함수 리소스 기반 정책 (p. 10)</a>	20 KB
<a href="#">함수 계층 (p. 62)</a>	5개 계층
<a href="#">호출 빈도(초당 요청 수)</a>	10배 동시 실행 한도( <a href="#">동기식 (p. 80)</a> – 모든 소스) 10배 동시 실행 한도(비동기식 – AWS 외의 소스) 무제한(비동기식 – <a href="#">AWS 서비스 소스 (p. 128)</a> )
<a href="#">호출 페이로드 (p. 78)(요청 및 응답)</a>	6 MB(동기식) 256 KB(비동기식)
<a href="#">배포 패키지 (p. 32) 크기</a>	50 MB(직접 업로드용 압축 파일) 250 MB(계층을 포함해 압축 해제됨) 3 MB(콘솔 편집기)
<a href="#">테스트 이벤트(콘솔 편집기)</a>	10
<a href="#">/tmp 디렉터리 스토리지</a>	512 MB
<a href="#">파일 설명자</a>	1,024
<a href="#">실행 프로세스/스레드</a>	1,024

AWS Identity and Access Management, Amazon CloudFront(Lambda@Edge), Amazon Virtual Private Cloud 등과 같은 다른 서비스에 대한 제한이 Lambda 함수에 영향을 줄 수 있습니다. 자세한 내용은 [AWS 서비스 제한 및 다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#)를 참조하십시오.

# AWS Lambda 권한

AWS Identity and Access Management(IAM)을 사용하여 Lambda API 및 리소스(함수 및 계층 등)에 대한 액세스를 관리할 수 있습니다. 계정에서 Lambda를 사용하는 사용자와 애플리케이션에 대해 IAM 사용자, 그룹, 역할 등에 적용할 수 있는 권한 정책의 권한을 관리할 수 있습니다. 내 Lambda 리소스를 사용하는 다른 계정이나 AWS 서비스에 권한을 부여하려면 리소스 자체에 적용되는 정책을 사용합니다.

Lambda 함수에는 [실행 역할 \(p. 9\)](#)이라는 정책도 있으며, 이것은 AWS 서비스 및 리소스에 대한 액세스 권한을 부여합니다. 함수는 최소한 로그 스트리밍을 위해 Amazon CloudWatch Logs에 액세스할 수 있어야 합니다. [AWS X-Ray를 사용하여 함수를 추적 \(p. 226\)](#)하거나, AWS SDK를 통해 함수가 서비스에 액세스할 경우 실행 역할에서 호출 권한을 부여할 수 있습니다. [이벤트 소스 매핑 \(p. 80\)](#)을 사용하여 함수를 트리거할 경우 Lambda는 실행 역할을 사용하여 이벤트 소스에서 읽을 수 있는 권한을 받습니다.

## Note

함수가 AWS API 또는 인터넷을 통해 액세스할 수 없는 관계형 데이터베이스와 같은 리소스에 네트워크를 통해 액세스해야 할 경우, [VPC에 연결하도록 구성 \(p. 66\)](#)하십시오.

[리소스 기반 정책 \(p. 10\)](#)을 사용하면 다른 계정과 AWS 서비스에 Lambda 리소스를 사용할 수 있는 권한을 부여할 수 있습니다. Lambda 리소스에는 함수, 버전, 별칭, 계층 버전이 포함됩니다. 각 리소스는 해당 리소스를 액세스할 때 적용되는 권한 정책 및 사용자에게 적용되는 기타 정책을 갖습니다. Amazon S3와 같은 AWS 서비스가 Lambda 함수를 호출할 때 리소스 기반 정책을 통해 액세스가 부여됩니다.

계정의 사용자 및 애플리케이션에 대한 권한을 관리하려면 [Lambda가 제공하는 관리형 정책을 사용 \(p. 13\)](#)하거나 직접 작성하십시오. Lambda 콘솔은 함수의 구성 및 트리거에 대한 정보를 받기 위해 여러 서비스를 사용합니다. 관리형 정책을 그대로 사용하거나, 이를 바탕으로 보다 제한적인 정책을 만들 수 있습니다.

작업이 영향을 주는 리소스별로 또는 경우에 따라 추가 조건을 적용하여 사용자 권한을 제한할 수 있습니다. 예를 들어 함수의 Amazon 리소스 이름(ARN) 패턴을 지정하여 생성하는 함수 이름에 사용자가 사용자 이름을 포함시키도록 할 수 있습니다. 또한 예를 들어 로깅 소프트웨어를 가져올 때 특정 계층을 사용하도록 사용자가 함수를 구성해야 하는 조건을 추가할 수 있습니다. 각 작업이 지원하는 리소스 및 조건을 보려면 [리소스 및 조건 \(p. 18\)](#)을 참조하십시오.

IAM에 대한 자세한 내용은 IAM 사용 설명서의 [IAM이란 무엇입니까?](#)를 참조하십시오.

## 주제

- [AWS Lambda 실행 역할 \(p. 9\)](#)
- [AWS Lambda에서 리소스 기반 정책 사용 \(p. 10\)](#)
- [AWS Lambda에 대한 자격 증명 기반 IAM 정책 \(p. 13\)](#)
- [Lambda 작업의 리소스와 조건 \(p. 18\)](#)

# AWS Lambda 실행 역할

AWS Lambda 함수의 실행 역할은 AWS 서비스 및 리소스에 대한 액세스 권한을 부여합니다. 함수를 만들 때 이 역할을 제공하면 함수를 호출할 때 Lambda가 이 역할을 수임합니다. 개발 작업을 위해 Amazon CloudWatch에 로그를 전송하고, AWS X-Ray에 추적 데이터를 업로드하는 권한을 가진 실행 역할을 만들 수 있습니다.

### 실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda
  - 권한 – AWSLambdaBasicExecutionRole, AWSXrayWriteOnlyAccess
  - 역할 이름 – **lambda-role**

언제든지 함수의 실행 역할에서 원한을 추가하거나 제거할 수 있으며, 다른 역할에 사용하도록 함수를 구성할 수 있습니다. 함수가 AWS SDK를 사용하여 호출하는 서비스 및 Lambda가 선택적 기능을 활성화하기 위해 사용하는 서비스에 대해 권한을 추가합니다.

다음 관리형 정책은 Lambda 기능을 사용하는 데 필요한 권한을 제공합니다.

- AWSLambdaBasicExecutionRole – CloudWatch에 로그를 업로드할 수 있는 권한.
- AWSLambdaKinesisExecutionRole – Amazon Kinesis 데이터 스트림 또는 소비자의 이벤트를 읽을 수 있는 권한.
- AWSLambdaDynamoDBExecutionRole – Amazon DynamoDB 스트림에서 레코드를 읽을 수 있는 권한.
- AWSLambdaSQSQueueExecutionRole – Amazon Simple Queue Service(Amazon SQS) 대기열에서 메시지를 읽을 수 있는 권한.
- AWSLambdaVPCAccessExecutionRole – 함수를 VPC에 연결하기 위해 탄력적 네트워크인터페이스를 관리하는 권한.
- AWSXrayWriteOnlyAccess – X-Ray에 추적 데이터를 업로드하는 권한.

[이벤트 소스 매핑](#) (p. 80)을 사용하여 함수를 호출할 경우 Lambda는 이벤트 데이터를 읽는 실행 역할을 사용합니다. 예를 들어 Amazon Kinesis에 대한 이벤트 소스 매핑은 데이터 스트림의 이벤트를 읽고 이를 함수에 배치(batch)로 전송합니다. 다음과 같은 서비스에 이벤트 소스 매핑을 사용할 수 있습니다.

Lambda가 이벤트를 읽는 서비스

- [Amazon Kinesis](#) (p. 175)
- [Amazon DynamoDB](#) (p. 165)
- [Amazon Simple Queue Service](#) (p. 211)

관리형 정책 외에도, Lambda 콘솔은 추가 사용 사례와 관련된 권한을 가진 사용자 지정 정책을 생성하는 템플릿을 제공합니다. 함수를 생성할 때 하나 이상의 템플릿의 권한을 사용하여 새로운 실행 역할을 만들 수 있습니다. 블루프린트에서 함수를 생성하거나, 다른 서비스에 액세스해야 하는 옵션을 구성할 때 이러한 템플릿도 자동으로 적용됩니다. 예제 템플릿은 이 설명서의 [GitHub 리포지토리](#)에서 구할 수 있습니다.

## AWS Lambda에서 리소스 기반 정책 사용

AWS Lambda는 Lambda 함수 및 계층에 대해 리소스 기반 권한 정책을 지원합니다. 리소스 기반 정책을 사용하여 리소스별로 다른 계정에 사용 권한을 부여할 수 있습니다. 리소스 기반 정책을 사용하여 AWS 서비스가 해당 함수를 호출할 수 있도록 허용할 수도 있습니다.

Lambda 함수의 경우 함수를 호출하거나 관리하기 위한 [권한을 계정에 부여](#) (p. 12) 할 수 있습니다. 다중 구문을 추가하여 여러 계정에 액세스를 부여하거나, 함수를 호출하는 계정에 액세스를 부여할 수 있습니다. 계정에서의 작업에 대한 응답으로 다른 AWS 서비스에서 호출하는 함수의 경우 [서비스에 호출 권한을 부여](#) (p. 11)하는 정책을 사용하십시오.

Lambda 계층의 경우 계층의 한 버전에 대해 리소스 기반 정책을 사용하여 다른 계정에서 해당 버전을 사용하게 할 수 있습니다. 단일 계정 또는 모든 계정에 권한을 부여하는 정책 외에도 계층에 대해 조직의 모든 계정에 권한을 부여할 수도 있습니다.

Note

[AddPermission \(p. 335\)](#) 및 [AddLayerVersionPermission \(p. 331\)](#) API 작업 범위 내의 Lambda 리소스에 대해서만 리소스 기반 정책을 업데이트할 수 있습니다. Lambda 리소스에 대한 정책을 JSON으로 작성할 수 없으며, 해당 작업에 대한 파라미터로 매핑되지 않는 조건은 사용할 수 없습니다.

리소스 기반 정책은 한 버전의 함수, 버전, 별칭 또는 계층에 적용됩니다. 이 정책은 하나 이상의 서비스 및 계정에 권한을 부여합니다. 여러 리소스에 대한 액세스 권한을 갖게 하거나 리소스 기반 정책이 지원하지 않는 API 작업을 사용하게 하려는 신뢰할 수 있는 계정의 경우, [교차 계정 역할 \(p. 13\)](#)을 사용할 수 있습니다.

주제

- [함수에 AWS 서비스에 대한 액세스 권한 부여 \(p. 11\)](#)
- [함수에 다른 계정에 대한 액세스 권한 부여 \(p. 12\)](#)
- [계층에 다른 계정에 대한 액세스 권한 부여 \(p. 12\)](#)
- [리소스 기반 정책 정리 \(p. 13\)](#)

## 함수에 AWS 서비스에 대한 액세스 권한 부여

함수를 호출하는 AWS 서비스를 사용 ([p. 81](#))하는 경우, 리소스 기반 정책의 구문에 권한을 부여하십시오. 함수에 구문을 적용하거나, 한 버전 또는 별칭으로 제한할 수 있습니다.

Note

Lambda 콘솔을 사용하여 함수에 트리거를 추가하는 경우, 콘솔은 함수의 리소스 기반 정책을 업데이트하여 서비스가 함수를 호출할 수 있도록 합니다. Lambda 콘솔에서 사용할 수 없는 다른 계정이나 서비스에 권한을 부여하려면 AWS CLI를 사용합니다.

`add-permission` 명령을 사용하여 구문을 추가합니다. 가장 간단한 리소스 기반 정책 구문은 서비스가 함수를 호출할 수 있도록 허용합니다. 다음 명령은 `my-function`라는 함수를 호출할 수 있는 권한을 Amazon SNS에 부여합니다.

```
$ 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"}
```

이렇게 하면 Amazon SNS가 함수를 호출하지만, 호출을 트리거하는 Amazon SNS 주제를 제한하지 않습니다. 함수를 특정 리소스에서만 호출하도록 하려면, `source-arn` 옵션을 사용하여 리소스의 Amazon 리소스 이름(ARN)을 지정합니다. 다음 명령은 Amazon SNS가 `my-topic`이라는 주제 구독에 대해서만 함수를 호출할 수 있도록 허용합니다.

```
$ 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
```

일부 서비스는 다른 계정의 함수를 호출할 수 있습니다. 해당 계정 ID를 소유하는 소스 ARN을 지정하면 문제가 되지 않습니다. 하지만 Amazon S3의 경우 소스는 ARN에 버킷의 계정 ID가 포함되지 않은 버킷입니다.

버킷을 삭제하고, 다른 계정에서 동일한 이름으로 버킷을 생성할 수 있습니다. 계정의 리소스만 함수를 호출할 수 있게 하려면 account-id 옵션을 사용합니다.

```
$ 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
```

## 함수에 다른 계정에 대한 액세스 권한 부여

다른 AWS 계정에 권한을 부여하려면 계정 ID를 principal로 지정합니다. 다음 예제는 prod 별칭을 가진 my-function을 호출할 수 있는 권한을 210987654321 계정에 부여합니다.

```
$ 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"}
```

별칭 (p. 50)은 다른 계정에서 호출할 수 있는 버전을 제한합니다. 다른 계정에서 함수 ARN에 별칭을 포함시켜야 합니다.

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

그런 다음 별칭을 업데이트하여 필요에 따라 새 버전을 지정할 수 있습니다. 별칭을 업데이트할 때 다른 계정은 새 버전을 사용하기 위해 코드를 변경할 필요가 없으며, 사용자가 선택한 버전을 호출할 수 있는 권한만 갖습니다.

기존 함수에서 작업을 수행 (p. 19)하는 API 작업에 대해 교차 계정 액세스 권한을 부여할 수 있습니다. 예를 들어 계정에서 별칭 목록을 확인할 수 있게 하려면 lambda>ListAliases에 대한 액세스를 부여하고, 함수 코드를 다운로드할 수 있게 하려면 lambda:GetFunction에 대한 액세스를 부여할 수 있습니다. 각 권한을 별도로 추가하거나, lambda:\*를 사용하여 특정 함수에 대한 모든 작업에 대한 액세스를 부여합니다.

여러 함수에 대한 권한 또는 함수에서 작업을 수행하지 않는 작업에 대한 권한을 다른 계정에 부여하려면 역할 (p. 13)을 사용합니다.

## 계층에 다른 계정에 대한 액세스 권한 부여

다른 계정에 계층 사용 권한을 부여하려면 add-layer-version-permission 명령과 함께 해당 계층 버전의 권한 정책에 명령문을 추가하십시오. 각 명령문에서 단일 계정, 모든 계정 또는 조직을 대상으로 권한을 부여할 수 있습니다.

```
$ 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"}
```

권한은 단일 버전의 계층에만 적용됩니다. 새 계층 버전을 만들 때마다 해당 절차를 반복합니다.

조직 내 모든 계정에 권한을 부여하려면 `organization-id` 옵션을 사용합니다. 다음 예제에서는 버전 3의 계층을 사용할 권한을 조직의 모든 계정에 부여합니다.

```
$ 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"}}}"
```

모든 AWS 계정에 권한을 부여하려면 보안 주체에 대해 \*를 사용하고 조직 ID는 생략하십시오. 계정 또는 조직이 다수인 경우, 여러 가지 명령문을 추가하십시오.

## 리소스 기반 정책 정리

함수의 리소스 기반 정책을 보려면 `get-policy` 명령을 사용합니다.

```
$ 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
```

버전 및 별칭의 경우 버전 번호나 별칭을 함수 이름에 추가합니다.

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

함수에서 권한을 제거하려면 `remove-permission`을 사용합니다.

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

하나의 계층에 대한 권한을 보려면 `get-layer-version-policy` 명령을 사용하고, 해당 정책에서 명령문을 제거하려면 `remove-layer-version-permission` 명령을 사용하십시오.

```
$ 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
```

## AWS Lambda에 대한 자격 증명 기반 IAM 정책

AWS Identity and Access Management(IAM)에서 자격 증명 기반 정책을 사용하여 계정의 사용자에게 Lambda에 대한 액세스 권한을 부여할 수 있습니다. 자격 증명 기반 정책은 사용자에게 직접 적용하거나, 사

용자와 연결된 그룹 및 역할에 적용할 수 있습니다. 다른 계정의 사용자에게 내 계정의 역할을 수행할 수 있는 권한 및 Lambda 리소스에 대한 액세스 권한을 부여할 수도 있습니다.

Lambda는 Lambda API 작업에 대한 액세스 권한 및 경우에 따라 Lambda 리소스를 개발하고 관리하는 데 사용되는 다른 서비스에 대한 액세스 권한을 부여하는 관리형 정책을 제공합니다. Lambda는 필요에 따라 관리형 정책을 업데이트하여 정책 제공 시 사용자가 새 기능에 액세스할 수 있도록 보장합니다.

- AWSLambdaFullAccess – AWS Lambda 작업에 대한 모든 액세스 권한 및 Lambda 리소스를 개발 및 유지하는 데 사용되는 다른 서비스에 대한 모든 액세스 권한을 부여합니다.
- AWSLambdaReadOnlyAccess – AWS Lambda 리소스에 대한 읽기 전용 액세스 권한을 부여합니다.
- AWSLambdaRole – Lambda 함수를 호출할 수 있는 권한을 부여합니다.

관리형 정책은 사용자가 수정할 수 있는 계층이나 함수를 제한하지 않고 API 작업에 대한 권한을 부여합니다. 보다 세부적으로 제어하기 위해 사용자의 권한을 제한하는 정책을 직접 만들 수 있습니다.

#### 섹션

- [함수 개발 \(p. 14\)](#)
- [계층 개발 및 사용 \(p. 17\)](#)
- [교차 계정 역할 \(p. 18\)](#)

## 함수 개발

다음은 범위를 제한한 권한 정책의 예입니다. 사용자가 지정된 접두사(intern-)로 이름이 지정되고 지정된 실행 역할로 구성된 Lambda 함수를 생성하고 관리할 수 있도록 허용합니다.

#### Example 함수 개발 정책

```
{  
    "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:PutEventSourceMapping",  
                "lambda:UpdateEventSourceMapping"  
            ]  
        }  
    ]  
}
```

```
        "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:PassRole"
    ],
    "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-*"
}
]
```

정책의 권한은 구문이 지원하는 [리소스 및 조건 \(p. 18\)](#)에 따라 구문으로 구성됩니다.

- `ReadOnlyPermissions` – 찾아보기 및 보기 기능을 사용할 때 Lambda 콘솔은 이러한 권한을 사용합니다. 이들 권한은 리소스 패턴 또는 조건을 지원하지 않습니다.

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

- `DevelopFunctions` – `AddPermission` 및 `PutFunctionConcurrency`를 제외하고 `intern-`로 접두사가 지정된 함수에서 작동하는 Lambda 작업을 사용합니다. `AddPermission`은 함수에 대한 [리소스 기반 정책 \(p. 10\)](#)을 수정하여 보안에 영향을 줄 수 있습니다. `PutFunctionConcurrency`는 함수에 대한 조정 용량을 예약하여 다른 함수로부터 용량을 가져올 수 있습니다.

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

- `DevelopEventSourceMappings` – intern-으로 접두사가 지정된 함수에 대한 소스 매핑을 관리합니다. 이러한 작업은 이벤트 소스 매핑에 대해 작용하지만 조건과 함께 함수를 제한할 수 있습니다.

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

- `PassExecutionRole` – IAM 권한을 가진 사용자가 생성하고 관리해야 하는 `intern-lambda-execution-role`이라는 역할만 조회하고 전달합니다. 함수에 실행 역할을 할당할 경우 `PassRole`이 사용됩니다.

```
"Action": [
    "iam>ListRolePolicies",
    "iam>ListAttachedRolePolicies",
    "iam:GetRole",
    "iam:PassRole"
],
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- `ViewExecutionRolePolicies` – 실행 역할에 연결된 AWS 제공 관리형 정책을 조회합니다. 이렇게 하면 콘솔에서 함수의 사용 권한을 볼 수 있지만 계정에서 다른 사용자가 생성한 정책을 볼 수 있는 권한은 포함되지 않습니다.

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

- `ViewLogs` – CloudWatch Logs를 사용하여, intern-으로 접두사가 지정된 함수에 대한 로그를 조회합니다.

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

이 정책은 사용자가 다른 사용자의 리소스를 위험하게 만들지 않고 Lambda를 시작할 수 있도록 허용합니다. 사용자가 더 광범위한 IAM 권한이 필요한 다른 AWS 서비스에 의해 트리거되는 기능을 구성하거나 호출하는 것은 허용하지 않습니다. 또한 CloudWatch 및 X-Ray와 같이 범위 제한 정책을 지원하지 않는 서비스에 대한 권한은 포함하지 않습니다. 사용자에게 지표 및 추적 데이터에 대한 액세스 권한을 부여하려면 이러한 서비스에 대해 읽기 전용 정책을 사용합니다.

함수에 대한 트리거를 구성할 경우 해당 함수를 호출하는 AWS 서비스를 사용하기 위한 액세스 권한이 필요합니다. 예를 들어 Amazon S3 트리거를 구성하려면 버킷 알림을 관리하기 위해 Amazon S3 작업에 대한 권한이 필요합니다. 이러한 권한의 대부분은 AWSLambdaFullAccess 관리형 정책에 포함됩니다. 예제 정책은 이 설명서의 [GitHub 리포지토리](#)에서 사용할 수 있습니다.

## 계층 개발 및 사용

다음 정책은 계층을 만들고 이를 함수에 사용할 수 있는 권한을 사용자에게 부여합니다. 이 리소스 패턴에 따르면 계층 이름이 test-로 시작되는 한 사용자는 원하는 AWS 리전에서 원하는 계층 버전으로 작업할 수 있습니다.

### Example 계층 개발 정책

```
{  
    "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-*:  
        }  
    ]  
}
```

함수 생성 및 구성 중 lambda:Layer 조건을 사용하여 계층 사용을 적용할 수도 있습니다. 예를 들어 다른 계정에서 계시한 계층을 사용자가 사용하지 못하게 할 수 있습니다. 다음 정책은 CreateFunction 및 UpdateFunctionConfiguration 작업에 조건을 추가하여 지정된 계층이 123456789012 계정에 속한 계층이어야 하도록 요구합니다.

```
{  
    "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:*:  
                    ]  
                }  
            }  
        }  
    ]  
}
```

}

조건이 적용되도록 하려면, 다른 구문이 이러한 작업에 대한 권한을 사용자에게 부여하지 않는지 확인하십시오.

## 교차 계정 역할

이전의 정책과 구문을 역할에 적용할 수 있으며, 그런 다음 다른 계정과 공유하여 Lambda 리소스에 대한 액세스 권한을 부여할 수 있습니다. IAM 사용자와 달리, 역할에는 인증을 위한 자격 증명이 없습니다. 대신, 누가 역할을 맡을 수 있고 해당 권한을 사용할 수 있는지를 지정하는 신뢰 정책이 있습니다.

신뢰하는 계정에 교차 계정 역할을 사용하여 Lambda 작업 및 리소스에 대한 액세스 권한을 부여할 수 있습니다. 함수를 호출하거나 계층을 사용하는 권한만 부여하려면 [리소스 기반 정책 \(p. 10\)](#)을 사용하십시오.

자세한 내용은 IAM 사용 설명서에서 [IAM 역할](#)을 참조하십시오.

## Lambda 작업의 리소스와 조건

IAM 정책에 리소스와 조건을 지정하여 사용자 권한의 범위를 제한할 수 있습니다. 각각의 API 작업은 해당 작업의 동작에 따라 다양한 조합의 리소스와 조건을 지원합니다.

각 IAM 정책 구문은 리소스에 대해 수행되는 작업에 대한 권한을 부여합니다. 지정된 리소스에서 이루어지는 작업이 아니거나 모든 리소스에 대해 그 작업을 수행할 수 있도록 권한을 부여하는 경우, 정책에서 해당 리소스의 값은 와일드카드(\*)가 됩니다. 대부분의 API 작업에서는 리소스의 Amazon 리소스 이름(ARN) 또는 복수의 리소스에 맞는 ARN 패턴을 지정함으로써 사용자 수정이 가능한 리소스를 제한할 수 있습니다.

리소스별 권한을 제한하려면 ARN으로 리소스를 지정하십시오.

### Lambda 리소스 ARN 형식

- 함수 – arn:aws:lambda:**us-west-2:123456789012:function:my-function**
- 함수 버전 – arn:aws:lambda:**us-west-2:123456789012:function:my-function:1**
- 함수 별칭 – arn:aws:lambda:**us-west-2:123456789012:function:my-function:TEST**
- 이벤트 소스 매핑 – arn:aws:lambda:**us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47**
- 계층 – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer**
- 계층 버전 – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer:1**

예를 들어 다음 정책은 123456789012 계정의 사용자가 미국 서부(오레곤) 리전에서 my-function 함수를 호출할 수 있도록 허용합니다.

### Example Lambda 정책 호출

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

}

이것은 작업 ID(`lambda:InvokeFunction`)가 API 작업([호출 \(p. 392\)](#))과 다른 특수한 경우입니다. 다른 작업의 경우, 작업 이름 앞에 `lambda:`를 붙인 것이 작업 ID입니다.

조건이란 허용되는 작업인지 여부를 판단하기 위해 로직을 추가로 적용하는 선택적 정책 요소를 말합니다. 모든 작업에서 지원되는 [일반 조건](#) 외에, Lambda에는 일부 작업에서 추가 파라미터 값을 제한하는 데 사용할 수 있는 조건 유형도 정의되어 있습니다.

예를 들어, `lambda:Principal` 조건으로는 사용자가 함수의 리소스 기반 정책에 대한 호출 액세스 권한을 부여할 수 있는 서비스 또는 계정을 제한합니다. 다음 정책에 따르면 사용자는 `test` 함수 호출 권한을 SNS 주제에 부여할 수 있습니다.

### Example 함수 정책 권한 관리

```
{  
    "Version": "2012-10-17",  
    "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"  
                }  
            }  
        }  
    ]  
}
```

이 조건은 보안 주체가 다른 서비스 또는 계정이 아닌 Amazon SNS여야 합니다. 리소스 패턴에 따르면 함수 이름은 `test`이고 버전 번호나 별칭이 포함되어야 합니다. 예: `test:v1`.

Lambda 및 기타 AWS 서비스의 리소스 및 조건에 대한 자세한 내용은 IAM 사용 설명서의 [작업](#), [리소스](#) 및 [조건 키](#)를 참조하십시오.

#### 섹션

- [함수 \(p. 19\)](#)
- [이벤트 소스 매핑 \(p. 20\)](#)
- [계층 \(p. 21\)](#)

## 함수

아래 표에 설명된 것과 같이, 함수에 대한 작업도 특정 함수에만 실행되도록 함수, 버전, 별칭 ARN 등으로 제한할 수 있습니다. 리소스 제한이 지원되지 않는 작업은 모든 리소스(\*)를 대상으로만 부여할 수 있습니다.

### 함수

작업	리소스	Condition
<a href="#">AddPermission (p. 335)</a>	함수	<code>lambda:Principal</code>
<a href="#">RemovePermission (p. 433)</a>	함수 버전	

작업	리소스	Condition
	함수 별칭	
<a href="#">Invoke (p. 392)</a>  권한: <code>lambda:InvokeFunction</code>	함수  함수 버전  함수 별칭	없음
<a href="#">CreateFunction (p. 347)</a> <a href="#">UpdateFunctionConfiguration (p. 455)</a>	함수	<code>lambda:Layer</code>
<a href="#">CreateAlias (p. 339)</a>  <a href="#">DeleteAlias (p. 355)</a>  <a href="#">DeleteFunction (p. 360)</a>  <a href="#">DeleteFunctionConcurrency (p. 362)</a>  <a href="#">GetAlias (p. 368)</a>  <a href="#">GetFunction (p. 374)</a>  <a href="#">GetFunctionConfiguration (p. 377)</a>  <a href="#">GetPolicy (p. 390)</a>  <a href="#">ListAliases (p. 399)</a>  <a href="#">ListVersionsByFunction (p. 415)</a>  <a href="#">PublishVersion (p. 422)</a>  <a href="#">PutFunctionConcurrency (p. 428)</a>  <a href="#">UpdateAlias (p. 440)</a>  <a href="#">UpdateFunctionCode (p. 448)</a>	함수	없음
<a href="#">GetAccountSettings (p. 366)</a>  <a href="#">ListFunctions (p. 405)</a>  <a href="#">ListTags (p. 413)</a>  <a href="#">TagResource (p. 436)</a>  <a href="#">UntagResource (p. 438)</a>	*	없음

## 이벤트 소스 매핑

이벤트 소스 매핑에서는 삭제 및 업데이트 권한을 특정 이벤트 소스로 제한할 수 있습니다. 사용자가 이벤트 소스를 호출하기 위해 구성할 수 있는 함수를 `lambda:FunctionArn` 조건으로 제한할 수 있습니다.

이러한 작업에서는 리소스가 이벤트 소스 매핑이며, 따라서 해당 이벤트 소스 매핑으로 호출되는 함수에 따라 Lambda의 조건으로 권한을 제어할 수 있습니다.

## 이벤트 소스 매핑

작업	리소스	Condition
<a href="#">DeleteEventSourceMapping (p. 357)</a>	이벤트 소스 매핑	<code>lambda:FunctionArn</code>
<a href="#">UpdateEventSourceMapping (p. 444)</a>		
<a href="#">CreateEventSourceMapping (p. 343)</a>	*	<code>lambda:FunctionArn</code>
<a href="#">GetEventSourceMapping (p. 371)</a>	*	없음
<a href="#">ListEventSourceMappings (p. 402)</a>		

## 계층

계층 작업으로 사용자가 함수에 사용하거나 관리할 수 있는 계층을 제한할 수 있습니다. 계층 사용과 관련된 작업 및 권한은 계층의 버전에 적용되며, `PublishLayerVersion`은 계층 이름에 적용됩니다. 어느 쪽이든 와일드카드를 사용하여 사용자가 작업할 수 있는 계층을 이름별로 제한할 수 있습니다.

### 계층

작업	리소스	Condition
<a href="#">AddLayerVersionPermission (p. 331)</a>	계층 버전	없음
<a href="#">RemoveLayerVersionPermission (p. 431)</a>		
<a href="#">GetLayerVersion (p. 382)</a>		
<a href="#">GetLayerVersionPolicy (p. 388)</a>		
<a href="#">DeleteLayerVersion (p. 364)</a>		
<a href="#">PublishLayerVersion (p. 418)</a>	계층	없음
<a href="#">ListLayers (p. 408)</a>	*	없음
<a href="#">ListLayerVersions (p. 410)</a>		

# Lambda 함수 작업

AWS Lambda를 처음 사용하는 경우라면 AWS Lambda로 어떻게 코드를 실행할 수 있는지 궁금할 것입니다. AWS Lambda가 어떻게 내 Lambda 코드를 실행하는 데 필요한 메모리 및 CPU 용량을 알 수 있을까요? 다음 단원에서는 함수의 작동 방법에 대한 개요를 제공합니다.

다음 단원에서는 생성한 함수가 호출되는 방법 및 함수를 배포하고 모니터링하는 방법에 대해 다릅니다. [AWS Lambda 함수 작업의 모범 사례 \(p. 124\)](#)의 함수 코드 및 함수 구성 단원도 읽어보는 것이 좋습니다.

먼저 Lambda 함수 작성의 기본 원리에 대해 설명하는 [Lambda 함수 빌드 \(p. 22\)](#) 주제를 소개하겠습니다.

## 주제

- [Lambda 함수 빌드 \(p. 22\)](#)
- [AWS Lambda 콘솔 편집기를 사용하여 함수 생성 \(p. 24\)](#)
- [프로그래밍 모델 \(p. 31\)](#)
- [배포 패키지 만들기 \(p. 32\)](#)
- [Lambda 함수에서 AWS 리소스에 액세스 \(p. 33\)](#)

## Lambda 함수 빌드

하나 이상의 Lambda 형태로 된 애플리케이션 코드를 컴퓨팅 서비스인 AWS Lambda에 업로드합니다. 그러면 AWS Lambda에서 대신 코드를 실행합니다. AWS Lambda는 호출 시 코드를 실행하기 위해 서버의 프로비저닝 및 관리를 담당합니다.

일반적으로 AWS Lambda 기반 애플리케이션의 수명 주기에는 코드 작성, AWS Lambda로의 코드 배포, 모니터링 및 문제 해결이 포함됩니다. 다음은 이러한 각각의 수명 주기 단계에서 나올 수 있는 일반적인 질문들입니다.

- Lambda 함수에 대한 코드 작성 – 어떤 언어가 지원되고 있습니까? 따라야 하는 프로그래밍 모델이 있습니까? 예 업로드하려면 코드 및 종속 프로그램을 어떻게 패키징해야 합니까? 어떤 도구를 사용할 수 있습니까?
- 코드 업로드 및 Lambda 함수 생성 – AWS Lambda에 내 코드 패키지를 업로드하려면 어떻게 해야 합니까? 예 코드 실행을 시작하는 지점을 알려주려면 어떻게 해야 합니까? 메모리 및 제한 시간 같은 컴퓨팅 요구 사항은 어떻게 지정합니까?
- 모니터링 및 문제 해결 – 내 Lambda 함수가 프로덕션 단계에 있는 경우, 어떤 측정치를 사용할 수 있습니까? 오류가 있을 경우, 로그를 얻거나 문제를 해결하려면 어떻게 해야 합니까?

다음 단원은 기본 정보를 제공하며, 맨 끝의 예제 단원은 탐색이 가능하도록 실제 예제를 제공합니다.

## Lambda 함수에 대한 코드 작성

AWS Lambda에서 지원되는 언어로 Lambda 함수 코드를 작성할 수 있습니다. 지원 언어의 목록은 [AWS Lambda 런타임 \(p. 99\)](#)을 참조하십시오. 콘솔, Eclipse IDE, Visual Studio IDE 같은 코드 작성 도구가 있습니다. 그러나 사용 가능한 도구와 옵션은 다음에 따라 다릅니다.

- Lambda 함수 코드를 작성하기 위해 선택하는 언어.

- 코드에서 사용하는 라이브러리. AWS Lambda 런타임은 라이브러리의 일부를 제공하기 때문에 사용하려는 추가 라이브러리를 업로드해야 합니다.

다음 표에는 언어와 사용 가능한 도구 및 옵션이 나열되어 있습니다.

언어	코드 작성을 위한 도구 및 옵션
Node.js	<ul style="list-style-type: none"><li>AWS Lambda 콘솔</li><li>IDE 플러그인이 포함된 Visual Studio(<a href="#">Visual Studio에서 AWS Lambda 지원 참조</a>)</li><li>자체 작성 환경</li><li>자세한 내용은 <a href="#">코드 배포 및 Lambda 함수 생성 (p. 23)</a> 단원을 참조하십시오.</li></ul>
Java	<ul style="list-style-type: none"><li>AWS Toolkit for Eclipse가 포함된 Eclipse(<a href="#">AWS Toolkit for Eclipse에서 AWS Lambda 사용 참조</a>)</li><li>자체 작성 환경</li><li>자세한 내용은 <a href="#">코드 배포 및 Lambda 함수 생성 (p. 23)</a> 단원을 참조하십시오.</li></ul>
C#	<ul style="list-style-type: none"><li>IDE 플러그인이 포함된 Visual Studio(<a href="#">Visual Studio에서 AWS Lambda 지원 참조</a>)</li><li>.NET Core(<a href="#">.NET Core 설치 안내서 참조</a>)</li><li>자체 작성 환경</li><li>자세한 내용은 <a href="#">코드 배포 및 Lambda 함수 생성 (p. 23)</a> 단원을 참조하십시오.</li></ul>
Python	<ul style="list-style-type: none"><li>AWS Lambda 콘솔</li><li>자체 작성 환경</li><li>자세한 내용은 <a href="#">코드 배포 및 Lambda 함수 생성 (p. 23)</a> 단원을 참조하십시오.</li></ul>
Go	<ul style="list-style-type: none"><li>자체 작성 환경</li><li>자세한 내용은 <a href="#">코드 배포 및 Lambda 함수 생성 (p. 23)</a> 단원을 참조하십시오.</li></ul>
PowerShell	<ul style="list-style-type: none"><li>자체 작성 환경</li><li>PowerShell Core 6.0(<a href="#">PowerShell Core 설치 참조</a>)</li><li>.NET Core 2.1 SDK(<a href="#">.NET 다운로드 참조</a>)</li><li>AWSLambdaPSCore 모듈(<a href="#">PowerShell Gallery 참조</a>)</li></ul>

뿐만 아니라 선택한 언어에 관계 없이 Lambda 함수 코드 작성에 패턴이 있습니다. 예를 들어 Lambda 함수의 핸들러 메서드(즉, 코드 실행을 시작할 때 AWS Lambda가 가장 먼저 호출하는 메서드)를 작성하는 방법, 핸들러에 이벤트를 전달하는 방법, CloudWatch Logs에서 로그를 생성하기 위해 코드에서 사용할 수 있는 명령문, AWS Lambda 런타임과 상호 작용하여 제한 시간까지 남아 있는 시간 같은 정보를 획득하는 방법, 예외를 처리하는 방법 등이 여기에 해당됩니다. [프로그래밍 모델 \(p. 31\)](#) 단원은 지원되는 각각의 언어로 정보를 제공합니다.

## 코드 배포 및 Lambda 함수 생성

Lambda 함수를 생성하려면 먼저 코드와 종속 프로그램을 배포 패키지에 패키징합니다. 그런 다음, AWS Lambda에 배포 패키지를 업로드하여 Lambda 함수를 생성합니다.

#### 주제

- [배포 패키지 만들기 \(p. 24\)](#)
- [배포 패키지 업로드 \(p. 24\)](#)
- [Lambda 함수 테스트 \(p. 24\)](#)

## 배포 패키지 만들기

먼저, 특정한 방법으로 코드와 종속 프로그램을 구성하고 배포 패키지를 생성해야 합니다. 배포 패키지를 생성하기 위한 지침은 코드 작성을 위해 선택한 언어에 따라 다릅니다. 예를 들어 Jenkins(Node.js 및 Python용) 및 Maven(Java용) 같은 빌드 플러그인을 사용하여 배포 패키지를 생성할 수 있습니다. 자세한 내용은 [배포 패키지 만들기 \(p. 32\)](#) 단원을 참조하십시오.

콘솔을 사용하여 Lambda 함수를 생성하면 콘솔은 사용자를 위한 배포 패키지를 생성한 다음, 이를 업로드하여 Lambda 함수를 생성합니다.

## 배포 패키지 업로드

AWS Lambda는 Lambda 함수 생성에 사용되는 [CreateFunction \(p. 347\)](#) 작업을 제공합니다. AWS Lambda 콘솔, AWS CLI 및 AWS SDK를 사용하여 Lambda 함수를 생성할 수 있습니다. 내부적으로 이러한 모든 인터페이스는 CreateFunction 작업을 호출합니다.

배포 패키지를 제공하는 것 외에도, Lambda 함수에 대한 컴퓨팅 요구 사항, Lambda 함수의 핸들러 메서드 이름, 코드 작성을 위해 선택한 언어에 따라 다른 런타임 등을 포함하여 Lambda 함수를 생성할 때 구성 정보를 제공할 수 있습니다. 자세한 내용은 [Lambda 함수 작업 \(p. 22\)](#) 단원을 참조하십시오.

## Lambda 함수 테스트

Lambda 함수가 특정 유형의 이벤트를 처리하도록 설계된 경우, 샘플 이벤트 데이터를 사용하여 다음 방법 중 하나로 Lambda 함수를 테스트할 수 있습니다.

- 콘솔에서 Lambda 함수를 테스트합니다.
- AWS CLI를 사용하여 Lambda 함수를 테스트합니다. Invoke 메서드를 사용하여 Lambda 함수를 호출하여 샘플 이벤트 데이터에 전달할 수 있습니다.
- [AWS SAM CLI](#)를 사용하여 Lambda 함수를 로컬로 테스트합니다.

## 모니터링 및 문제 해결

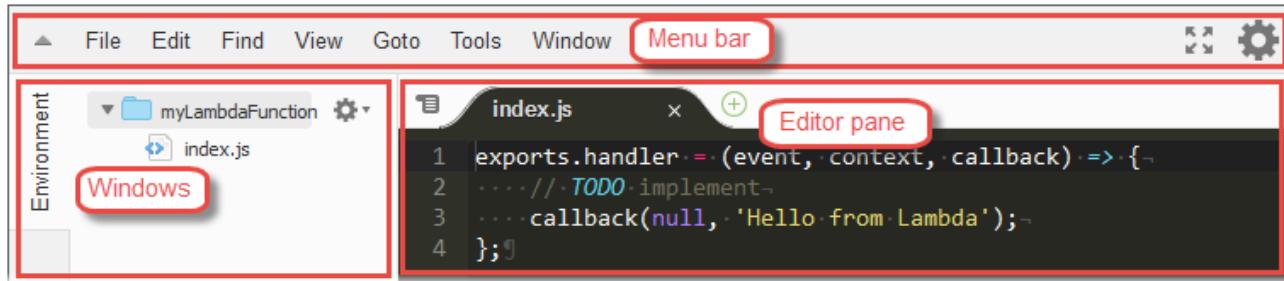
Lambda 함수가 프로덕션 단계를 거친 후에는 AWS Lambda는 사용자를 대신하여 함수를 자동으로 모니터링하고 Amazon CloudWatch를 통해 측정치를 보고합니다. 자세한 내용은 [AWS Lambda에 대한 Amazon CloudWatch 지표 액세스 \(p. 222\)](#) 단원을 참조하십시오.

함수의 실패 문제를 해결하는 데 도움을 제공하기 위해 Lambda는 함수에 의해 처리되는 모든 요청을 로깅하고 Amazon CloudWatch Logs에서 코드가 생성하는 로그를 자동으로 저장합니다. 자세한 내용은 [AWS Lambda에 대한 Amazon CloudWatch 로그 액세스 \(p. 222\)](#) 단원을 참조하십시오.

## AWS Lambda 콘솔 편집기를 사용하여 함수 생성

AWS Lambda 콘솔의 코드 편집기를 사용하여 Lambda 함수 코드를 작성하고 테스트하고 실행 결과를 볼 수 있습니다.

코드 편집기에는 메뉴 모음, 창 및 편집기 창이 포함되어 있습니다.



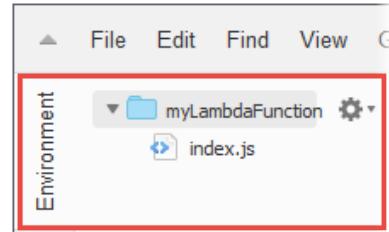
메뉴 모음은 일반적인 명령을 실행할 때 사용합니다. 자세한 내용은 [메뉴 모음 사용 \(p. 29\)](#) 단원을 참조하십시오.

창은 파일, 폴더 및 기타 명령으로 작업할 때 사용합니다. 자세한 내용은 [파일 및 폴더 작업 \(p. 25\)](#) 및 [명령 작업 \(p. 30\)](#) 단원을 참조하십시오.

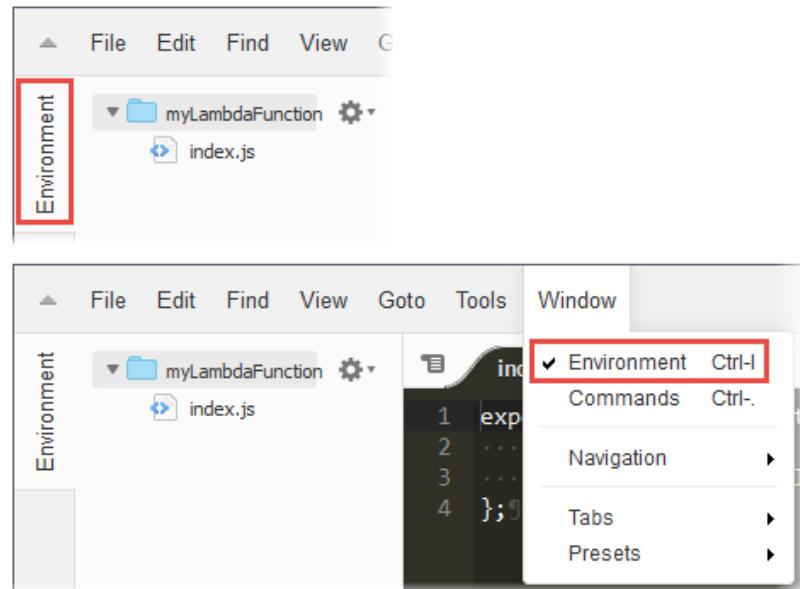
편집기 창은 코드를 작성할 때 사용합니다. 자세한 내용은 [코드 작업 \(p. 27\)](#) 단원을 참조하십시오.

## 파일 및 폴더 작업

코드 편집기에서 [Environment] 창을 사용하여 함수의 파일을 생성하고, 열고, 관리할 수 있습니다.



[Environment] 창을 표시하거나 숨기려면 [Environment] 버튼을 선택합니다. [Environment] 버튼이 표시되지 않으면 메뉴 모음에서 [Window], [Environment]를 선택합니다.

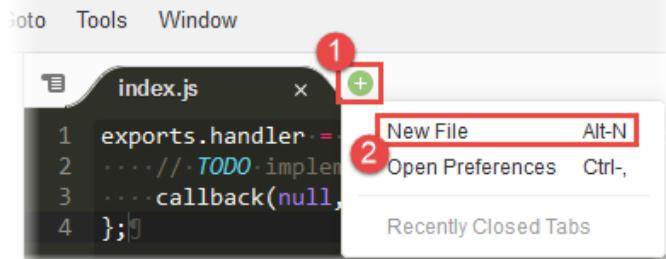


단일 파일을 열고 해당 콘텐츠를 편집기 창에 표시하려면 [Environment] 창에서 해당 파일을 두 번 클릭합니다.

여러 파일을 열고 해당 콘텐츠를 편집기 창에 표시하려면 [Environment] 창에서 해당 파일을 선택합니다. 선택한 파일을 마우스 오른쪽 버튼으로 클릭하고 [Open]을 선택합니다.

새 파일을 생성하려면 다음 중 하나를 수행합니다.

- [Environment] 창에서 새 파일을 추가할 폴더를 마우스 오른쪽 버튼으로 클릭하고 [New File]을 선택합니다. 파일 이름 및 확장명을 입력한 다음 Enter 키를 누릅니다.
- 메뉴 모음에서 [File], [New File]을 선택합니다. 파일을 저장할 준비가 되면 메뉴 모음에서 [File], [Save] 또는 [File], [Save As]를 선택합니다. 그런 다음 [Save As] 대화 상자가 표시되면 파일 이름을 지정하고 파일을 저장할 위치를 선택합니다.
- 편집기 창의 탭 버튼 모음에서 [+] 버튼을 선택한 다음 [New File]을 선택합니다. 파일을 저장할 준비가 되면 메뉴 모음에서 [File], [Save] 또는 [File], [Save As]를 선택합니다. 그런 다음 [Save As] 대화 상자가 표시되면 파일 이름을 지정하고 파일을 저장할 위치를 선택합니다.



새 폴더를 생성하려면 [Environment] 창에서 새 폴더를 저장할 폴더를 마우스 오른쪽 버튼으로 클릭하고 [New Folder]를 선택합니다. 폴더 이름을 입력한 다음 Enter 키를 누릅니다.

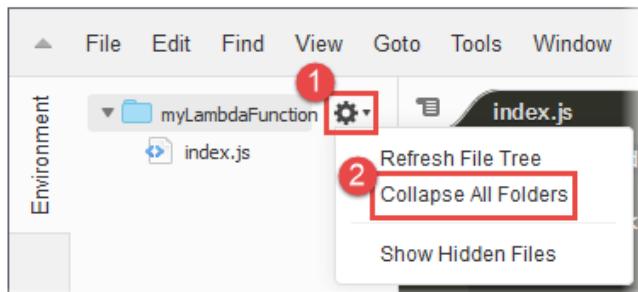
파일을 저장하려면 파일을 열고 해당 콘텐츠를 편집기 창에 표시한 다음 메뉴 모음에서 [File], [Save]를 선택합니다.

파일이나 폴더의 이름을 바꾸려면 [Environment] 창에서 파일이나 폴더를 마우스 오른쪽 버튼으로 클릭합니다. 바꿀 이름을 입력한 다음 Enter 키를 누릅니다.

파일이나 폴더를 삭제하려면 [Environment] 창에서 파일이나 폴더를 선택합니다. 선택한 파일이나 폴더를 마우스 오른쪽 버튼으로 클릭하고 [Delete]를 선택합니다. 그런 다음 [Yes](단일 개체를 선택한 경우) 또는 [Yes to All]을 선택하여 삭제를 확인합니다.

파일이나 폴더를 잘라내기, 복사, 붙여넣기 또는 복제하려면 [Environment] 창에서 파일이나 폴더를 선택합니다. 선택한 파일이나 폴더를 마우스 오른쪽 버튼으로 클릭한 다음 각각 [Cut], [Copy], [Paste] 또는 [Duplicate]를 선택합니다.

폴더를 축소하려면 [Environment] 창에서 기어 모양 아이콘을 선택한 다음 [Collapse All Folders]를 선택합니다.

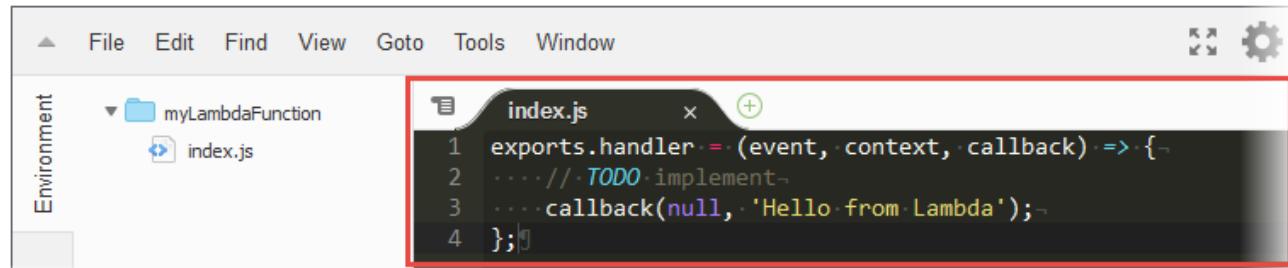


숨김 파일을 표시하거나 숨기려면 [Environment] 창에서 기어 모양 아이콘을 선택한 다음 [Show Hidden Files]를 선택합니다.

[Commands] 창을 사용하여 파일을 생성하고, 열고, 관리할 수도 있습니다. 자세한 내용은 [명령 작업 \(p. 30\)](#) 단원을 참조하십시오.

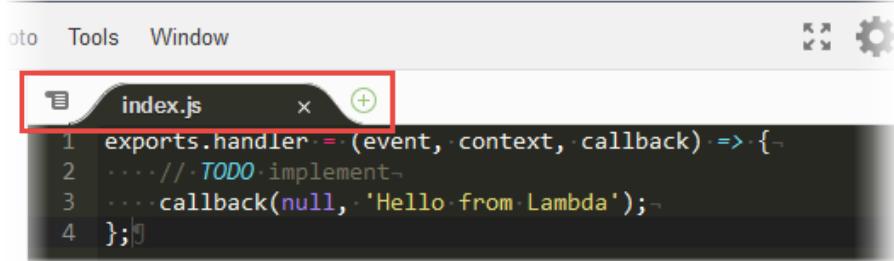
## 코드 작업

코드 편집기의 편집기 창을 사용하여 코드를 보고 작성할 수 있습니다.



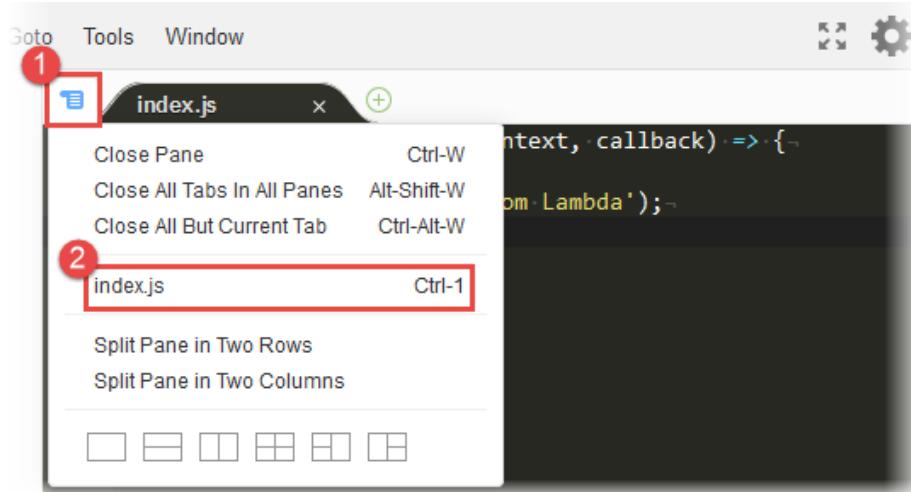
## 탭 버튼 작업

탭 버튼 모음을 사용하여 파일을 선택하고, 보고, 생성할 수 있습니다.



열려 있는 파일의 콘텐츠를 표시하려면 다음 중 하나를 수행합니다.

- 파일의 탭을 선택합니다.
- 탭 버튼 모음에서 드롭다운 메뉴 버튼을 선택한 다음 파일 이름을 선택합니다.



열려 있는 파일을 닫으려면 다음 중 하나를 수행합니다.

- 파일의 탭에서 [X] 아이콘을 선택합니다.

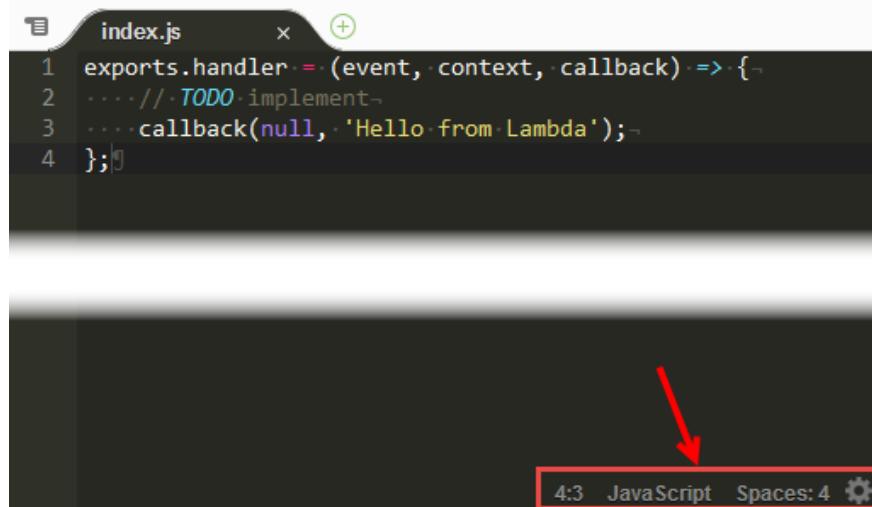
- 파일의 탭을 선택합니다. 그런 다음 탭 버튼 모음에서 드롭다운 메뉴 버튼을 선택하고 [Close Pane]을 선택합니다.

열려 있는 여러 파일을 닫으려면 탭 버튼 모음에서 드롭다운 메뉴를 선택한 다음 필요에 따라 [Close All Tabs in All Panes] 또는 [Close All But Current Tab]을 선택합니다.

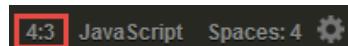
새 파일을 생성하려면 탭 버튼 모음에서 [+] 버튼을 선택한 다음 [New File]을 선택합니다. 파일을 저장할 준비가 되면 메뉴 모음에서 [File], [Save] 또는 [File], [Save As]를 선택합니다. 그런 다음 [Save As] 대화 상자가 표시되면 파일 이름을 지정하고 파일을 저장할 위치를 선택합니다.

## 상태 표시줄 작업

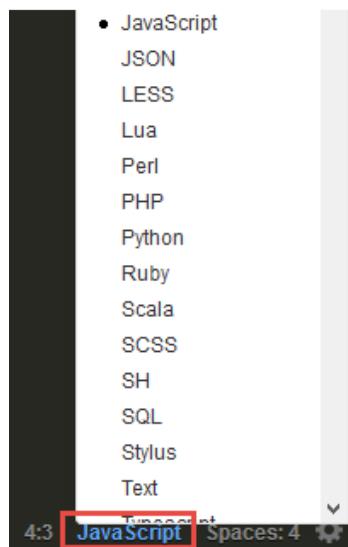
상태 표시줄을 사용하여 활성 파일의 특정 행으로 빠르게 이동하고 코드 표시 방법을 변경할 수 있습니다.



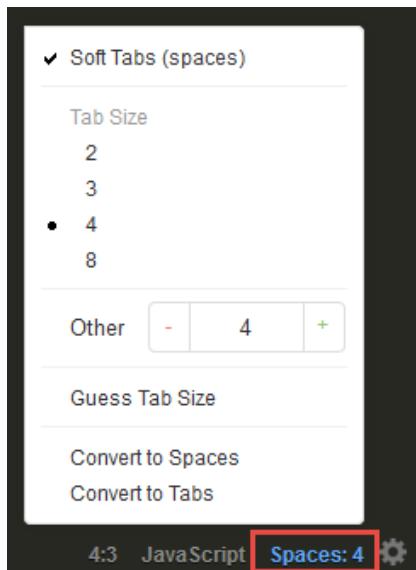
활성 파일의 특정 행으로 빠르게 이동하려면 행 선택기를 선택하고 이동할 행 번호를 입력한 다음 Enter 키를 누릅니다.



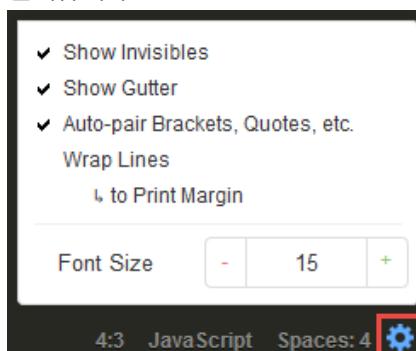
활성 파일의 코드 색상 체계를 변경하려면 코드 색상 체계 선택기를 선택한 다음 새 코드 색상 체계를 선택합니다.



활성 파일에서 소프트 탭을 사용할지 공백을 사용할지에 대한 설정, 탭 크기 설정, 공백이나 탭으로 변환할지에 대한 설정을 변경하려면 공백 및 탭 선택기를 선택한 다음 새 설정을 선택합니다.



모든 파일에 대해 표시되지 않는 문자나 여백을 표시할지 또는 숨길지에 대한 설정, 괄호 또는 따옴표 쌍 자동 완성 설정, 줄 바꿈 또는 글꼴 크기에 대한 설정을 변경하려면 기어 모양 아이콘을 선택한 다음 새 설정을 선택합니다.

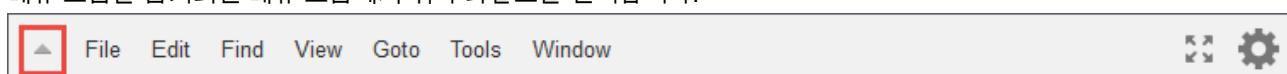


## 메뉴 모음 사용

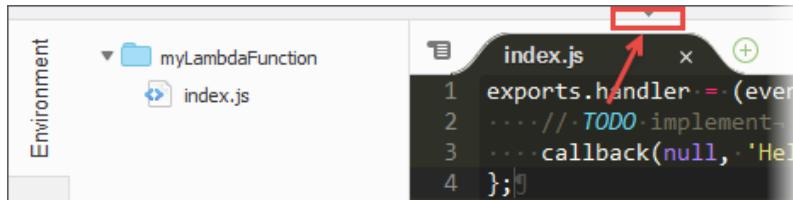
메뉴 모음을 사용하여 일반적인 명령을 실행할 수 있습니다.



메뉴 모음을 숨기려면 메뉴 모음에서 위쪽 화살표를 선택합니다.



숨겨져 있는 메뉴 모음을 표시하려면 메뉴 모음에서 아래쪽 화살표를 선택합니다.



명령이 수행하는 작업 목록은 AWS Cloud9 사용 설명서에서 [메뉴 명령 참조](#) 단원을 참조하십시오. 이 참조에 나와 있는 일부 명령은 코드 편집기에서 사용할 수 없습니다.

[Commands] 창을 사용하여 명령을 실행할 수도 있습니다. 자세한 내용은 [명령 작업 \(p. 30\)](#) 단원을 참조하십시오.

## 전체 화면 모드에서 작업

코드 편집기를 확장하여 코드 작업 공간을 추가로 확보할 수 있습니다.

코드 편집기를 웹 브라우저 창의 가장자리까지 확장하려면 메뉴 모음에서 [Toggle fullscreen] 버튼을 선택합니다.



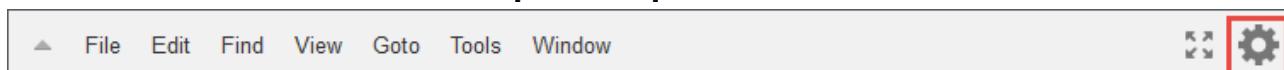
코드 편집기를 원래 크기로 축소하려면 [Toggle fullscreen] 버튼을 다시 선택합니다.

전체 화면 모드에서는 메뉴 모음에 저장과 테스트라는 두 가지 옵션이 추가로 표시됩니다. [Save]를 선택하면 함수 코드가 저장됩니다. [Test] 또는 [Configure Events]를 선택하면 함수의 테스트 이벤트를 생성하거나 편집할 수 있습니다.

## 기본 설정 작업

표시할 코딩 힌트와 경고, 코드 접기 동작, 코드 자동 완성 동작 등의 다양한 코드 편집기 설정을 변경할 수 있습니다.

코드 편집기 설정을 변경하려면 메뉴 모음에서 [Preferences] 기어 모양 아이콘을 선택합니다.



설정이 수행하는 작업의 목록은 AWS Cloud9 사용 설명서에서 다음 참조를 참조하십시오.

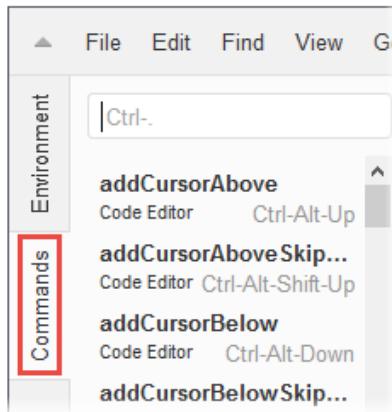
- 사용자가 변경할 수 있는 프로젝트 설정
- 사용자가 변경할 수 있는 사용자 설정

이러한 참조에 나와 있는 일부 설정은 코드 편집기에서 사용할 수 없습니다.

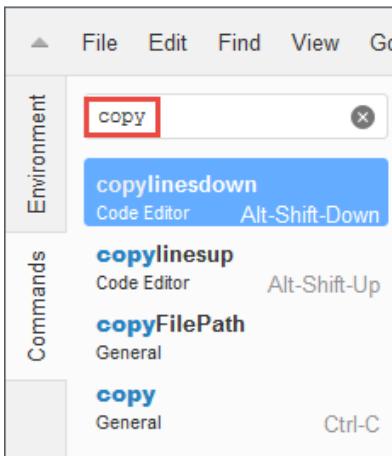
## 명령 작업

[Commands] 창을 사용하여 메뉴 모음, [Environment] 창 및 편집기 창에 있는 다양한 명령을 실행할 수 있습니다.

[Commands] 창을 표시하거나 숨기려면 [Commands] 버튼을 선택합니다. [Commands] 버튼이 표시되지 않으면 메뉴 모음에서 [Window], [Commands]를 선택합니다.



명령을 실행하려면 [Commands] 창에서 명령을 선택합니다. 명령을 찾으려면 검색 상자에 해당 명령의 이름 전체 또는 일부를 입력합니다.



명령이 수행하는 작업 목록은 AWS Cloud9 사용 설명서에서 [명령 참조](#) 단원을 참조하십시오. 이 참조에 나와 있는 일부 명령은 코드 편집기에서 사용할 수 없습니다.

## 프로그래밍 모델

AWS Lambda가 지원하는 언어 중 하나로 Lambda 함수의 코드를 작성합니다. 선택한 언어에 관계 없이 아래의 핵심 개념을 포함하여 함수의 코드 작성에 공통된 패턴이 있습니다.

- **핸들러** – 핸들러는 Lambda 함수의 실행을 시작하기 위한 함수 AWS Lambda 호출입니다. 함수를 생성할 때 핸들러를 식별합니다. Lambda 함수가 호출되면 AWS Lambda는 핸들러 함수를 호출하여 코드 실행을 시작합니다. AWS Lambda는 이 핸들러에 첫 번째 파라미터로 이벤트 데이터를 전달합니다. 핸들러는 수신 이벤트 데이터를 처리해야 하고 코드에서 다른 모든 함수/메서드를 호출할 수 있습니다.
- **Context** – AWS Lambda는 두 번째 파라미터로 컨텍스트 객체도 핸들러 함수에 전달합니다. 이 컨텍스트 객체를 통해 코드는 와 상호 작용을 할 수 있습니다. 예를 들어 AWS Lambda가 Lambda 함수를 종료하기 앞서 코드에서 실행 시간이 남을 수 있습니다.

뿐만 아니라 Node.js 같은 언어의 경우에는 콜백을 사용하는 비동기식 플랫폼이 있습니다. AWS Lambda는 이 컨텍스트 객체에 추가 메서드를 제공합니다. 이 컨텍스트 객체 메서드를 사용하여 AWS Lambda에 Lambda 함수를 종료하고 선택에 따라 호출자에게 값을 반환하라고 명령합니다.

- **로깅** – Lambda 함수는 로깅 문을 포함할 수 있기 때문에 AWS Lambda가 CloudWatch Logs에 이러한 로그를 기록합니다. 특정 언어 문은 함수 코드를 작성하는 데 사용되는 언어에 따라 로그 항목을 생성합니다.

로깅에는 CloudWatch Logs 제한이 적용됩니다. 제한 또는 실행 컨텍스트 (p. 101)의 종료 등으로 인해 로그 데이터가 사라질 수 있습니다.

- 예외 – Lambda 함수는 AWS Lambda와 함수 실행의 결과를 주고 받아야 합니다. Lambda 함수 코드를 작성하는 언어에 따라 요청을 성공적으로 종료하거나 실행 동안 발생한 오류를 AWS Lambda에 알릴 수 있는 다양한 방법이 있습니다. 함수를 동기식으로 호출하면 가 결과를 클라이언트에 다시 전송합니다.
- 동시성 - 함수 인스턴스 하나의 이벤트 처리 속도보다 빠르게 함수를 호출하는 경우, Lambda는 추가 인스턴스를 실행하여 규모를 늘립니다. 각각의 함수 인스턴스는 한 번에 요청 하나만 처리하므로 스레드 또는 프로세스의 동기화를 염려할 필요가 없습니다. 그러나 이벤트 배치를 별별 처리하기 위해 비동기 언어 기능을 사용할 수 있으며, 이후 동일한 인스턴스의 호출에 사용하기 위해 /tmp 디렉터리에 데이터를 저장해둘 수 있습니다.

Lambda 함수 코드는 상태 비저장 스타일로 작성되어야 하며, 기본 컴퓨팅 인프라에 선호도가 없습니다. 코드는 로컬 파일 시스템 액세스, 하위 프로세스 및 비슷한 아티팩트가 요청의 수명으로 제한된다는 것을 예측해야 합니다. Amazon S3, Amazon DynamoDB 또는 다른 클라우드 스토리지 서비스에 영구적인 상태가 저장되어야 합니다. 함수를 상태 비저장 스타일로 작성하도록 하면 AWS Lambda가 이벤트 및 요청의 수신 속도에 따라 필요한 수 만큼 함수의 복사본을 실행할 수 있습니다. 이들 함수는 요청 간에 동일한 컴퓨팅 인스턴스에서 항상 실행되는 것은 아니며, Lambda 함수의 해당 인스턴스는 AWS Lambda에서 한 번 이상 사용될 수 있습니다. 자세한 내용은 [AWS Lambda 함수 작업의 모범 사례 \(p. 124\)](#) 단원을 참조하십시오.

- Node.js를 사용하여 Lambda 함수 빌드 (p. 235)
- Python을 사용하여 Lambda 함수 빌드 (p. 245)
- Ruby를 사용하여 Lambda 함수 빌드 (p. 322)
- Java를 사용하여 Lambda 함수 빌드 (p. 257)
- Go를 사용하여 Lambda 함수 빌드 (p. 283)
- C#을 사용하여 Lambda 함수 빌드 (p. 295)
- PowerShell을 사용하여 Lambda 함수 빌드 (p. 314)

## 배포 패키지 만들기

Lambda 함수를 생성하려면 먼저, 코드와 종속 프로그램으로 구성된 zip 또는 jar 파일인 Lambda 함수 배포 패키지를 만듭니다. zip을 만들 때 상위 폴더가 아닌 코드 및 종속성만 포함합니다. 그런 다음 zip 패키지에 대한 적절한 보안 권한을 설정해야 합니다.

- [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#)
- [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#)
- [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#)
- [AWS Lambda 배포 패키지\(Go\) \(p. 283\)](#)
- [AWS Lambda 배포 패키지\(C#\) \(p. 295\)](#)
- [AWS Lambda 배포 패키지\(PowerShell\) \(p. 314\)](#)

## Lambda 배포 패키지에 대한 권한 정책

잘못된 권한으로 업로드된 Zip 패키지로 인해 실행에 실패할 수 있습니다. AWS Lambda는 배포 패키지를 구성하는 코드 파일과 모든 종속 라이브러리에 대해 전역 읽기 권한을 요구합니다. 권한이 특정 사용자 계정으로 제한되지 않도록 다음 샘플을 사용하여 확인하십시오.

- Linux/Unix/OSX 환경: 아래 샘플과 같이 zipinfo를 사용합니다.

```
$ zipinfo test.zip
```

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

-r-----은 해당 파일 소유자에게 읽기 권한만 있으며, 따라서 Lambda 함수 실행에 실패할 수 있음을 나타냅니다. 다음은 전역 읽기 권한이 필요할 때 보게 되는 화면입니다.

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

이 문제를 재구적으로 수정하려면 다음 명령을 실행하십시오.

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

- 첫 번째 명령은 /tmp/package\_contents의 모든 파일이 소유자에게 읽기/쓰기 권한과 그룹 및 전체에 읽기 권한을 부여하도록 변경합니다.
- 두 번째 명령은 디렉터리에 대해 동일한 권한을 부여합니다.

## Lambda 함수에서 AWS 리소스에 액세스

Lambda는 함수 로직에 어떠한 제한도 적용하지 않습니다. 로직을 코딩하고 Lambda 함수 내에서 실행할 수 있기만 하면 됩니다. 함수의 일부로 다른 API를 호출하거나 데이터베이스와 같은 다른 AWS 서비스에 액세스해야 할 수 있습니다.

### AWS 제품 액세스

다른 AWS 서비스에 액세스하려면 AWS SDK를 사용하면 됩니다. AWS Lambda는 SDK에 필요한 자격 증명을 함수와 연결된 IAM 역할의 자격 증명으로 자동 설정합니다. 따라서 추가 단계를 수행할 필요가 없습니다. 예를 들어 다음은 S3 개체에 액세스하기 위해 Python SDK를 사용하는 샘플 코드입니다.

```
import boto3
import botocore

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

s3 = boto3.resource('s3')

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

편의를 위해 AWS Lambda에는 실행 환경의 일부로 AWS SDK의 여러 버전이 포함되어 있으므로 별도로 포함하지 않아도 됩니다. 포함된 SDK의 버전은 [AWS Lambda 런타임 \(p. 99\)](#) 단원을 참조하십시오. 종속성을 제어할 수 있도록 프로덕션 애플리케이션용 자체 AWS SDK 사본을 포함하는 것이 좋습니다.

## 비 AWS 서비스 액세스

Lambda 함수 내에서 서비스에 액세스하기 위해 모든 SDK를 포함할 수 있습니다. 예를 들어 Twilio 계정의 정보에 액세스하기 위해 [Twilio용 SDK](#)를 포함할 수 있습니다. 자격 증명을 암호화한 후 [AWS Lambda 환경 변수](#) (p. 39)를 사용하여 SDK 자격 증명 정보를 저장할 수 있습니다.

## 프라이빗 서비스 또는 리소스 액세스

기본적으로 AWS Lambda용 퍼블릭 인터넷을 통해 서비스나 API에 액세스할 수 있어야 합니다. 하지만 이런 식으로 표시되지 않은 API나 서비스가 있을 수 있습니다. 일반적으로 이러한 리소스는 퍼블릭 인터넷을 통해 액세스할 수 없도록 Amazon Virtual Private Cloud(Amazon VPC) 내에 생성합니다. 이러한 리소스는 Amazon Redshift 데이터 웨어하우스, Amazon ElastiCache 클러스터 또는 Amazon RDS 인스턴스와 같은 AWS 서비스 리소스일 수 있습니다. 또한 자체 EC2 인스턴스에서 실행되는 자체 서비스일 수도 있습니다. 기본적으로 VPC 내의 리소스는 Lambda 함수 내에서 액세스할 수 없습니다.

AWS Lambda는 기본적으로 VPC 내에서 함수 코드를 안전하게 실행합니다. 하지만 Lambda 함수를 프라이빗 VPC 내부의 리소스에 액세스할 수 있게 하려면 VPC 서브넷 ID 및 보안 그룹 ID가 포함된 추가 VPC 관련 구성 정보를 제공해야 합니다. AWS Lambda는 이 정보를 사용하여 함수가 프라이빗 VPC 내의 다른 리소스에 안전하게 연결할 수 있게 하는 탄력적 네트워크 인터페이스(ENI)를 설정합니다.

### Important

AWS Lambda는 전용 테넌시 VPC 내 리소스 연결을 지원하지 않습니다. 자세한 내용은 [전용 VPC](#)를 참조하십시오.

VPC 내에서 리소스에 액세스하기 위해 Lambda 함수를 구성하는 방법에 대해 알아보려면 [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성](#) (p. 66) 단원을 참조하십시오.

# AWS Lambda 함수 구성

AWS Lambda API 또는 콘솔을 사용하여 Lambda 함수에 대한 설정을 구성할 수 있습니다. [기본 함수 설정 \(p. 35\)](#)에는 Lambda 콘솔에서 함수를 생성할 때 지정하는 설명, 역할, 런타임 등이 포함됩니다. 함수를 생성한 후 추가 설정을 구성하거나, 생성 중에 API를 사용하여 핸들러 이름, 메모리 할당 및 보안 그룹과 같은 항목을 설정할 수 있습니다.

보안 암호를 함수 코드와 따로 유지하려면 보안 암호를 함수의 구성에 저장하고 초기화 중에 실행 환경에서 보안 암호를 읽으십시오. [환경 변수 \(p. 39\)](#)는 저장 상태에서 항상 암호화되며 전송 중에도 암호화할 수 있습니다. 환경 변수를 사용하면 외부 리소스에 대한 연결 문자열, 암호 및 엔드포인트를 제거하여 함수 코드를 이식 가능 상태로 만들 수 있습니다.

[버전 및 별칭 \(p. 45\)](#)은 함수 배포 및 호출을 관리하기 위해 생성할 수 있는 보조 리소스입니다. 함수 버전을 발행하여 코드 및 구성을 변경할 수 없는 별도의 리소스로 저장하고 특정 버전을 가리키는 별칭을 생성하십시오. 그런 다음 함수 별칭을 호출하도록 클라이언트를 구성하고 클라이언트를 새 버전으로 가리키면 클라이언트를 업데이트하는 대신 별칭을 업데이트할 수 있습니다.

함수에 라이브러리 및 기타 종속성을 추가할 때 배포 패키지를 생성하고 업로드하면 개발이 지연될 수 있습니다. [계층 \(p. 62\)](#)을 사용하여 함수의 종속성을 독립적으로 관리하고 배포 패키지를 작게 유지합니다. 또한 계층을 사용하면 자신의 라이브러리를 다른 고객과 공유할 수 있으며, 함수와 함께 공개적으로 사용 가능한 계층들을 사용할 수도 있습니다.

Amazon VPC의 AWS 리소스에 Lambda 함수를 사용하려면, 보안 그룹과 서브넷을 사용하여 [VPC 연결을 생성 \(p. 66\)](#)하도록 구성합니다. Lambda는 [탄력적 네트워크 인터페이스 \(ENI\)](#)를 사용하여 연결을 생성하므로, 함수가 작업 로드에 따라 확장할 때 이루어지는 연결 수를 처리하기에 충분한 ENI 용량이 해당 계정에 있는지 확인해야 합니다.

## 주제

- [AWS Lambda 함수 구성 \(p. 35\)](#)
- [동시성 관리 \(p. 37\)](#)
- [AWS Lambda 환경 변수 \(p. 39\)](#)
- [AWS Lambda 함수 버전 관리 및 별칭 \(p. 45\)](#)
- [AWS Lambda 계층 \(p. 62\)](#)
- [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 \(p. 66\)](#)
- [Lambda 함수 태그 지정 \(p. 74\)](#)

## AWS Lambda 함수 구성

Lambda 함수는 코드 및 모든 연결 종속 프로그램으로 이루어져 있습니다. 뿐만 아니라, 함수는 이와 연결된 구성 정보도 가지고 있습니다. 먼저, 함수를 생성할 때 구성 정보를 지정합니다.

### 함수 설정을 구성하려면

1. [Lambda 콘솔](#)을 엽니다.
2. 함수를 선택합니다.
3. 사용 가능한 옵션을 구성한 다음 저장을 선택합니다.

## 함수 설정

- 코드 – 함수의 코드와 종속 항목입니다. 스크립트 언어의 경우, 내장된 편집기 (p. 24)에서 함수 코드를 편집할 수 있습니다. 라이브러리를 추가하려면 또는 편집기가 지원하지 않는 언어의 경우 배포 패키지 (p. 32)를 업로드합니다.
  - 런타임 – 함수를 실행하는 Lambda 런타임 (p. 99)입니다.
  - 핸들러 – 함수가 호출될 때 런타임이 실행하는 메서드입니다. 이 값의 형식은 언어마다 다릅니다. 자세한 내용은 프로그래밍 모델 (p. 31) 단원을 참조하십시오.
  - 환경 변수 – 실행 환경에서 Lambda가 설정하는 키-값 페어입니다. 환경 변수를 사용 (p. 39)하여 코드 외부에서 함수의 구성을 확장합니다.
  - 태그 – Lambda가 함수 리소스에 연결하는 키-값 페어입니다. 태그를 사용 (p. 74)하여 Lambda 콘솔에서 비용 보고 및 필터링을 위해 Lambda 함수를 그룹별로 분류합니다.
- 태그는 전체 함수(모든 버전과 별칭을 포함)에 적용됩니다.
- 실행 역할 – IAM 역할 (p. 9)이 함수를 실행할 때 AWS Lambda가 수임하는 IAM 역할입니다.
  - 설명 – 함수의 설명입니다.
  - 메모리 – 실행 과정에서 함수가 사용할 수 있는 메모리의 양입니다. 128MB-3,008 MB (p. 7) 범위에서 선택합니다(64MB 간격).

Lambda는 구성된 메모리 크기에 선형 비례하여 CPU 용량을 할당합니다. 1,792MB에서 함수는 vCPU 1개 와 동일한 값을 갖습니다(초당 1 vCPU-초 크레딧).

- 제한 시간 – Lambda가 함수를 중지하기 전에 실행을 허용하는 시간입니다. 기본값은 3초입니다. 허용되는 최댓값은 900초입니다.
- 가상 사설 클라우드(VPC) – 해당 함수가, 인터넷을 통해 사용할 수 없는 리소스에 대한 네트워크에 액세스 해야 할 경우 VPC에 연결하도록 구성하십시오 (p. 66).
- 배달 못한 편지 대기열(DLQ) – 함수가 비동기적으로 호출될 경우, 실패한 호출을 수신하는 대기열이나 주제를 선택 (p. 86)하십시오.
- 적극 추적 활성화 – 수신 요청을 샘플링하고, 샘플링된 요청을 AWS X-Ray를 사용하여 추적 (p. 226)합니다.
- 동시성 – 함수에 대한 최대 동시 실행 횟수를 설정하는 함수의 동시성을 예약 (p. 37)하고 해당 동시성 수준에 대한 용량을 예약합니다.

예약된 동시성은 전체 함수(모든 버전과 별칭을 포함)에 적용됩니다.

함수 설정은 발행되지 않은 함수 버전에서만 변경할 수 있습니다. 버전을 발생하면 사용자가 해당 버전을 일관되게 사용할 수 있도록 하기 위해 코드와 대부분 설정이 잡깁니다. 구성 변경을 제어된 방식으로 전파하려면 별칭 (p. 45)을 사용하십시오.

Lambda API를 사용하여 함수를 구성하려면 다음 작업을 사용하십시오.

- UpdateFunctionCode (p. 448) – 함수의 코드를 업데이트합니다.
- UpdateFunctionConfiguration (p. 455) – 버전별 설정을 업데이트합니다.
- TagResource (p. 436) – 함수에 태그를 지정합니다.
- AddPermission (p. 335) – 함수, 버전 또는 별칭의 리소스 기반 정책 (p. 10)을 수정합니다.
- PutFunctionConcurrency (p. 428) – 함수의 예약 동시성을 구성합니다.
- PublishVersion (p. 422) – 현재 코드와 구성을 포함하는 변경 불가능 버전을 생성합니다.
- CreateAlias (p. 339) – 함수 버전의 별칭을 생성합니다.

예를 들어 AWS CLI를 사용하여 함수의 메모리 설정을 업데이트하려면 update-function-configuration 명령을 사용합니다.

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

함수 구성 모법 사례는 [함수 구성 \(p. 125\)](#) 단원을 참조하십시오.

## 동시성 관리

AWS Lambda의 조정 단위는 동시 실행입니다(자세한 내용은 [조정 동작 이해 \(p. 84\)](#) 참조). 하지만 어떤 경우에도 무제한 조정은 바람직하지 않습니다. 예를 들어 비용상의 이유로 또는 이벤트 배치를 처리하는데 걸리는 시간을 제어해야 하거나 단순히 다운스트림 리소스에 맞게 동시성을 제어해야 할 경우가 있습니다. 이를 지원하기 위해 에서는 계정 수준 및 함수 수준 모두에서 동시 실행 한도를 제어할 수 있습니다.

### 계정 수준 동시 실행 한도

기본적으로 AWS Lambda는 특정 리전 내의 모든 함수에서 동시 실행의 총 횟수를 1000으로 제한합니다. [GetAccountSettings \(p. 366\)](#) API를 사용하여 AccountLimit 객체를 표시하고 계정 수준 설정을 볼 수 있습니다. 아래 설명된 것처럼 이 한도를 올릴 수 있습니다.

동시 실행에 대한 한도 상승을 요청하려면

1. [AWS Support Center](#) 페이지를 열고 필요 시 로그인을 한 다음, Create case(사례 생성)를 선택합니다.
2. Regarding에서 Service Limit Increase를 선택합니다.
3. 한도 유형에서 Lambda를 선택하고 양식의 필수 필드를 입력한 다음, 페이지 하단의 버튼을 선택하여 선호하는 연락 방법을 선택합니다.

### 함수 수준 동시 실행 한도

기본적으로 모든 함수의 동시 실행 합계에 대해 동시 실행 한도가 적용됩니다. 공유 동시 실행 풀을 예약되지 않은 동시성 할당이라고 합니다. 함수 수준 동시성 한도를 설정하지 않은 경우 예약되지 않은 동시성 한도가 계정 수준 동시성 한도와 동일합니다. 계정 수준 한도를 올리면 예약되지 않은 동시성 한도도 이에 상응하여 올라갑니다. [GetAccountSettings \(p. 366\)](#) API 또는 AWS Lambda 콘솔을 사용하여 함수에 대한 예약되지 않은 동시성 할당을 볼 수 있습니다. 공유 동시 실행 풀에 대해 계산되고 있는 함수는 [GetFunctionConfiguration \(p. 377\)](#) API를 사용하여 쿼리할 때 어떠한 동시성 값도 표시하지 않습니다.

함수에 대한 동시 실행 한도를 설정할 수 있습니다. 이렇게 해야 하는 경우는 다음과 같습니다.

- 기본 동작에서는 한 함수에서 동시 실행이 급증할 경우 동시 실행 한도를 적용하여 격리한 함수를 제한할 수 없습니다. 함수에 대한 동시 실행 한도를 설정하면 해당 함수에 대해 지정된 동시 실행 값을 예약하는 것과 같습니다.
- 수신되는 요청 빈도에 따라 함수가 자동으로 조정되지만 아키텍처의 일부 리소스는 그렇게 할 수 없을 수도 있습니다. 예를 들어 관계형 데이터베이스의 경우 데이터베이스가 처리할 수 있는 동시 연결의 수에 제한이 있습니다. 함수의 다운스트림 리소스가 지원하는 값에 맞게 함수에 대한 동시 실행 한도를 설정할 수 있습니다.
- 함수가 VPC 기반 리소스에 연결하는 경우 서브넷이 적절한 주소 용량이 있어 함수의 ENI 확장 요구 사항을 지원할 수 있는지 확인해야 합니다. 다음 공식을 사용하여 ENI 용량을 대략적으로 평가할 수 있습니다.

Concurrent executions * (Memory in GB / 3 GB)
---

여기서 각 항목은 다음과 같습니다.

- 동시 실행 - 자체 워크로드의 프로젝션된 동시성입니다. 이 값을 결정하려면 [조정 동작 이해 \(p. 84\)](#)의 정보를 사용하십시오.
- GB 메모리 - Lambda; 함수에 대해 구성된 메모리의 양입니다.

서브넷 크기 제한에 맞게 함수에 대한 동시 실행 한도를 설정할 수 있습니다.

#### Note

함수가 호출 처리를 중지하도록 해야 할 경우 동시성을 0으로 설정하여 수신되는 모든 실행을 조절할 수 있습니다.

함수에 대한 동시성 한도를 설정하면 Lambda는 나머지 함수를 처리 중인 트래픽의 양에 관계없이 할당이 해당 함수에 적용되도록 합니다. 해당 한도를 초과하면 함수가 조절됩니다. 조절될 때 함수가 동작하는 방식은 이벤트 소스에 따라 달라집니다. 자세한 내용은 [조절 동작 \(p. 39\)](#) 단원을 참조하십시오.

#### Note

동시성 한도는 개별 버전에 대해서가 아니라 함수 수준에서만 설정할 수 있습니다. 지정된 함수의 모든 버전 및 별칭에 대한 모든 호출이 누적되어 함수 한도에 반영됩니다.

## 예약된 동시성 한도와 예약되지 않은 동시성 한도 비교

함수에 대한 동시 실행 한도를 설정한 경우 예약되지 않은 동시성 풀에서 해당 값이 공제됩니다. 예를 들어 계정의 동시 실행 한도가 1000이고 함수가 10개인 경우 한 함수는 200으로 한도를 지정하고 다른 함수는 100으로 한도를 지정할 수 있습니다. 남은 700은 다른 8개의 함수에서 공유됩니다.

#### Note

AWS Lambda는 예약되지 않은 동시성 풀의 동시 실행 수를 최소 100개로 유지하여 특정 한도가 설정되지 않은 함수가 계속해서 요청을 처리할 수 있도록 합니다. 따라서 총 계정 한도가 1000일 경우 개별 함수에 900만 할당할 수 있습니다.

## 함수별로 동시성 한도 설정(콘솔)

Lambda 콘솔을 사용하여 Lambda 함수에 대한 동시성 한도를 설정하려면 다음 작업을 수행합니다.

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. 함수를 선택합니다.
3. [Configuration] 탭에서 [Concurrency]를 선택합니다. [Reserve concurrency]에서 함수에 대해 예약할 최대 동시 실행 수로 값을 설정합니다. 이 값을 설정하면 예약되지 않은 계정 동시성 값이 자동으로 업데이트되어 계정의 다른 모든 함수에 사용할 수 있는 나머지 동시 실행 수가 표시됩니다. 또한 이 함수의 호출을 차단하려는 경우 값을 0으로 설정합니다. 이 계정에 대한 전용 할당량 값을 제거하려면 [Use unreserved account concurrency]를 선택합니다.

## 함수별로 동시성 한도 설정(CLI)

AWS CLI를 사용하여 Lambda 함수에 대한 동시성 한도를 설정하려면 다음 작업을 수행합니다.

- [PutFunctionConcurrency \(p. 428\)](#) 작업을 사용하여 함수 이름 및 이 함수에 할당하려는 동시성 한도를 전달합니다.

```
$ aws lambda put-function-concurrency --function-name my-function --reserved-concurrent-executions 100
```

AWS CLI를 사용하여 Lambda 함수에 대한 동시성 한도를 제거하려면 다음 작업을 수행합니다.

- [DeleteFunctionConcurrency \(p. 362\)](#) 작업을 사용하여 함수 이름을 전달합니다.

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

AWS CLI를 사용하여 Lambda 함수에 대한 동시성 한도를 보려면 다음 작업을 수행합니다.

- [GetFunction \(p. 374\)](#) 작업을 사용하여 함수 이름을 전달합니다.

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

함수별 동시성 설정은 다른 함수가 사용할 수 있는 동시성 풀에 영향을 줄 수 있습니다.

[PutFunctionConcurrency \(p. 428\)](#) API 및 [DeleteFunctionConcurrency \(p. 362\)](#) API에 대한 권한을 관리자로 제한하여 이러한 변경을 수행할 수 있는 사용자 수를 제한하는 것이 좋습니다.

## 조절 동작

함수와 연결된 동시성 한도에 도달하면 해당 함수에 대한 추가 호출 요청이 조절됩니다(호출이 함수를 실행하지 않음). 호출을 조절할 때마다 해당 함수에 대한 Amazon CloudWatch Throttles 측정치가 증가합니다. AWS Lambda는 조절된 호출 요청의 소스에 따라 요청을 다르게 처리합니다.

- 스트림 기반이 아닌 이벤트 소스: 이러한 이벤트 소스 중 일부는 Lambda 함수를 동기식으로 호출하고, 일부는 비동기식으로 함수를 호출합니다. 처리 방법은 각기 다릅니다.
  - 동기식 호출: 함수가 동기식으로 호출되고 조절되는 경우, Lambda는 429 오류를 반환하고 호출 중인 서비스는 재시도를 책임집니다. ThrottledReason 오류 코드는 함수 수준 조절(지정된 경우)에 도달했는지 또는 계정 수준 조절(아래 내용 참조)에 도달했는지를 설명합니다. 각 서비스마다 고유의 재시도 정책이 있습니다. 이벤트 소스 및 해당 호출 유형 목록은 [다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#) 단원을 참조하십시오.
  - 비동기식 호출: Lambda 함수가 비동기식으로 호출되고 제한되는 경우에는 AWS Lambda가 재시도 간에 지연 시간을 두고 최대 6시간 동안 제한된 이벤트를 자동으로 재시도합니다.
- 스트림 기반인 풀 기반 이벤트 소스: [Amazon Kinesis](#) 또는 [Amazon DynamoDB](#)처럼 AWS Lambda는 스트림을 폴링하고 Lambda 함수를 호출합니다. Lambda 함수가 조절될 때 Lambda는 데이터가 만료될 때까지 조절된 레코드 배치(batch)를 처리하려고 시도합니다. 이 기간은 Amazon Kinesis에서 최대 7일입니다. 조절된 요청은 샤드당 차단으로 처리되며, 조절된 레코드 배치(batch)가 만료되거나 성공적으로 처리될 때까지는 샤드에서 새로운 레코드를 읽어 들이지 않습니다. 스트림에 하나 이상의 샤드가 있을 경우,는 한도에 도달할 때까지 제한되지 않은 샤드에서 호출을 계속합니다.
- 스트림 기반 아닌 풀 기반 이벤트 소스: [Amazon Simple Queue Service](#)처럼, AWS Lambda는 대기열을 폴링하고 Lambda 함수를 호출합니다. Lambda 함수에 병목 현상이 발생하면 성공적으로 호출될 때까지 (이 경우 대기열에서 자동으로 메시지가 삭제됨) 혹은 대기열에 설정한 MessageRetentionPeriod가 만료될 때까지 Lambda가 병목 현상이 발생한 레코드 배치의 처리를 시도합니다.

## 동시 사용 모니터링

동시 실행 사용을 파악할 수 있도록 AWS Lambda는 다음 측정치를 제공합니다.

- ConcurrentExecutions: 계정 수준의 동시 실행 및 사용자 지정 동시성 한도가 설정된 함수에 대한 동시 실행을 보여줍니다.
- UnreservedConcurrentExecutions: 예약되지 않은 기본 동시성 풀에 할당된 함수에 대한 총 동시 실행 횟수를 표시합니다.

이러한 측정치 및 측정치에 액세스하는 방법은 [사용 Amazon CloudWatch \(p. 220\)](#) 단원을 참조하십시오.

## AWS Lambda 환경 변수

Lambda 함수용 환경 변수는 코드를 변경하지 않고 함수 코드와 라이브러리에 설정을 동적으로 전달할 수 있게 해줍니다. 환경 변수는 AWS Lambda 콘솔, AWS Lambda CLI 또는 AWS Lambda SDK를 사용하여 함수 구성의 일부로 생성 및 수정할 수 있는 키-값 페어입니다. AWS Lambda는 Node.js 함수용 `process.env` 같

이 해당 언어에서 지원되는 표준 API를 사용하여 Lambda 함수 코드에서 이러한 키-값 페어를 사용할 수 있게 해줍니다.

환경 변수를 사용하여 라이브러리가 파일을 설치할 디렉터리와 출력, 연결 및 로깅 설정을 저장할 장소를 알 수 있도록 도와줄 수 있습니다. 애플리케이션 로직에서 이러한 설정을 분리하면 다른 설정에 따라 함수 동작을 변경해야 할 때 함수 코드를 업데이트할 필요가 없습니다.

## 설정

개발부터 배포에 이르기까지 수명 주기 단계에 따라 Lambda 함수가 다르게 작동하도록 하고 싶다고 가정해 봅시다. 예를 들어 개발, 테스트 및 프로덕션 단계에는 함수와 연결해야 하는 데이터베이스가 포함될 수 있습니다. 이들 데이터베이스는 서로 다른 연결 정보가 필요하고 다른 테이블 이름을 사용합니다. 데이터베이스 이름, 연결 정보 또는 테이블 이름을 참조하기 위한 환경 변수를 생성하고, 함수 코드를 그대로 유지하면서 실행 중인 단계(예: 개발, 테스트, 프로덕션)에 따라 해당 함수의 값을 설정할 수 있습니다.

다음 스크린샷은 AWS 콘솔을 사용하여 함수의 구성 설정을 보여줍니다. 첫 번째 스크린샷은 테스트 단계에 해당되는 함수의 설정을 구성합니다. 두 번째 스크린샷은 프로덕션 단계에 대한 설정을 구성합니다.

The screenshot shows the 'Environment variables' section of the AWS Lambda function configuration. It displays two environment variables: 'Test\_DB' with value 'TEST' and 'DB\_Connection' with value 'Key'. Below this, the 'Encryption configuration' section is shown, featuring an enable checkbox for encryption in transit and a dropdown for selecting a KMS key to encrypt at rest, currently set to '(default) aws/lambda'. A red box highlights the 'Value' field for 'Test\_DB'.

▼ Environment variables

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

Database	Test_DB	Remove
DB_Connection	TEST	Remove
Key	Value	Remove

▼ Encryption configuration

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

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

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

(default) aws/lambda

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

Database	Prod_DB	Remove
DB_Connection	PROD	Remove
Key	Value	Remove

▼ Environment variables

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

KMS key to encrypt at rest [Info](#)  
Select a KMS key to encrypt the environment variables at rest, or simply let Lambda manage the encryption.  
(default) aws/lambda ▾ Enter value

▼ Encryption configuration

암호화 구성 단원을 참조하십시오. [민감한 정보를 저장하도록 환경 변수를 사용하여 Lambda 함수 생성](#) (p. 44) 자습서에서 이를 사용하는 방법에 대해 자세히 알아보겠습니다.

또한 AWS CLI를 사용하여 환경 변수를 포함하는 Lambda 함수를 생성할 수 있습니다. 자세한 내용은 [CreateFunction](#) (p. 347) 및 [UpdateFunctionConfiguration](#) (p. 455) API를 참조하십시오. 을 사용하여 함수를 생성 및 업데이트 할 때도 환경 변수가 지원됩니다. 환경 변수는 함수에 포함된 언어 런타임이나 라이브러리에 고유한 설정을 구성하는 데도 사용할 수 있습니다. 예를 들어 PATH를 수정하여 실행 파일이 저장되는 디렉터리를 지정할 수 있습니다. 또한 Python용 PYTHONPATH 또는 Node.js용 NODE\_PATH와 같은 런타임 고유의 환경 변수를 설정할 수도 있습니다.

다음 예제에서는 LD\_LIBRARY\_PATH 환경 변수를 설정하는 Lambda 함수를 새로 생성하는데, 이 함수는 런타임 시 공유 라이브러리가 동적으로 로딩되는 디렉터리를 지정하는 데 사용됩니다. 이 예제에서 Lambda 함수 코드는 /usr/bin/test/lib64 디렉터리에서 공유 라이브러리를 사용합니다. Runtime 파라미터는 nodejs6.10을 사용하지만, nodejs8.10을 지정할 수도 있습니다.

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

## 환경 변수의 이름을 지정하기 위한 규칙

집합의 총 크기가 4KB를 초과하지 않는 한, 생성할 수 있는 환경 변수의 수에는 제한이 없습니다.

기타 요구 사항은 다음과 같습니다.

- 문자 [a-zA-Z]로 시작해야 합니다.
- 영숫자 문자와 밑줄 ([a-zA-Z0-9\_])만 포함할 수 있습니다.

뿐만 아니라, AWS Lambda가 예약한 고유한 키 집합이 있습니다. 이렇게 예약된 키에 대해 값을 설정하려고 하면 해당 작업이 허용되지 않는다는 오류 메시지가 표시됩니다. 이들 키에 대한 자세한 내용은 [Lambda 함수에서 사용할 수 있는 환경 변수 \(p. 100\)](#)을 참조하십시오.

## 환경 변수 및 함수 버전 관리

버전 관리는 개발 단계에서 테스트 단계 및 프로덕션 단계로 진행함에 따라 Lambda 함수 버전을 하나 이상 게시하여 Lambda 함수 코드를 관리할 수 있도록 해줍니다. 게시한 Lambda 함수의 각 버전에 대해 환경 변수를 비롯하여 `MemorySize` 및 `Timeout` 한도 같은 기타 함수 고유의 구성이 해당 버전의 스냅샷으로 저장되며, 이러한 설정은 변경이 불가능합니다(바꿀 수 없음).

애플리케이션 및 구성 요구 사항이 변화함에 따라 새로운 버전의 Lambda 함수를 생성하고, 게시 중인 최신 버전 이전의 요구 사항을 충족하도록 환경 변수를 업데이트할 수 있습니다. 최신 버전의 함수는 `$LATEST`입니다.

뿐만 아니라, 특정 버전의 함수를 가리키는 별칭을 생성할 수 있습니다. 별칭은 이전 버전의 함수로 룰백해야 하는 경우에 해당 버전에 필요한 환경 변수가 포함된 이전 버전을 가리키도록 별칭을 지정할 수 있다는 점에서 장점이 있습니다. 자세한 내용은 [AWS Lambda 함수 버전 관리 및 별칭 \(p. 45\)](#) 단원을 참조하십시오.

## 환경 변수 암호화

환경 변수를 사용하는 Lambda 함수를 생성 또는 업데이트할 때 AWS Lambda는 [AWS Key Management Service](#)를 사용하여 이를 암호화합니다. Lambda 함수가 호출되면 이러한 값들이 암호 해독되어 Lambda 코드에 사용할 수 있는 상태가 됩니다.

리전에서 환경 변수를 사용하는 Lambda 함수를 처음으로 생성 또는 업데이트할 때 AWS KMS 내에서 기본 서비스 키가 자동으로 생성됩니다. 이 키는 환경 변수를 암호화하는 데 사용됩니다. 한편, 암호화 도우미와 KMS를 사용하여 Lambda 함수가 생성된 후에 환경 변수를 암호화하고 싶은 경우에는 자체 AWS KMS 키를 생성하고 기본 키 대신에 이를 선택해야 합니다. 기본 키를 선택하면 오류가 발생합니다. 자체 키를 생성하면 액세스 제어를 생성, 교체, 비활성화 및 정의하고 데이터를 보호하는 데 사용된 암호화 키를 감사하는 등 폭넓은 작업이 가능합니다. 자세한 내용은 [AWS Key Management Service 개발자 안내서](#)를 참조하십시오.

자체 키를 사용하는 경우에는 [AWS Key Management Service 요금](#) 지침에 따라 비용이 부과됩니다. 에서 제공된 기본 서비스 키를 사용하는 경우에는 비용이 부과되지 않습니다.

Lambda에서 기본 KMS 서비스 키를 사용하고 있는 경우에는 함수 실행 역할에서 추가적인 IAM 권한이 필요하지 않습니다. 변경 없이 역할이 자동으로 수행됩니다. 자체(사용자 지정) KMS 키를 제공하는 경우에는 실행 역할에 `kms:Decrypt`를 추가해야 합니다. 뿐만 아니라 함수를 생성 및 업데이트하고 싶은 사용자는 KMS 키를 사용할 수 있는 권한을 가지고 있어야 합니다. KMS 키에 대한 자세한 내용은 [AWS KMS에서 키 정책 사용](#)을 참조하십시오.

### Note

AWS Lambda는 함수에 역할을 할당할 때 추가하는 사용자 권한 부여를 통해 기본 KMS 키를 사용할 수 있는 권한을 해당 함수에 부여합니다. 그 역할을 삭제하고 동일한 이름으로 새 역할을 만들었으면 역할에 부여된 권한을 새로 고쳐야 합니다. 함수에 역할을 다시 할당하여 부여된 권한을 새로 고치십시오.

## 민감한 정보 저장

이전 단원에서 언급한 바와 같이, Lambda 함수를 배포할 때 배포가 진행되는 동안이 아니라 그 이후에 사용자가 지정한 모든 환경 변수가 기본적으로 암호화됩니다. 그리고 함수가 호출될 때 에서 자동으로 암호 해독이 됩니다. 환경 변수에 민감한 정보를 저장해야 하는 경우에는 함수를 배포하기 앞서 해당 정보를 암호화하는 것이 좋습니다.

다행히 Lambda 콘솔은 [AWS Key Management Service](#)를 활용하여 `Ciphertext` 같은 민감한 정보를 저장하는 암호화 도우미를 제공하여 암호화를 손쉽게 수행할 수 있게 해줍니다. 또한 Lambda 콘솔은 Lambda 함수 코드에서 사용되는 정보의 암호를 해독할 수 있도록 암호 해독 도우미 코드를 제공합니다. 자세한 내용은 [민감한 정보를 저장하도록 환경 변수를 사용하여 Lambda 함수 생성 \(p. 44\)](#) 단원을 참조하십시오.

## 오류 시나리오

함수 구성이 4KB를 초과하거나 AWS Lambda가 예약한 환경 변수 키를 사용하는 경우에는 업데이트 또는 생성 작업이 실패하면서 구성 오류 메시지가 나타납니다. 실행 시간 동안 환경 변수의 암호화/암호 해독이 실패할 수 있습니다. AWS Lambda가 AWS KMS 서비스 예외로 인해 환경 변수를 암호 해독할 수 없는 경우, AWS KMS는 오류 조건이 무엇이고 문제 해결을 위해 적용할 수 있는 구제책은 무엇인지 설명하는 예외 메시지를 반환합니다. 이러한 오류는 Amazon CloudWatch Logs의 함수 로그 스트림에 로깅됩니다. 예를 들어 환경 변수를 액세스하는 데 사용 중인 KMS 키가 비활성화된 경우에는 다음과 같은 오류 메시지가 나타납니다.

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

## 환경 변수를 사용하여 Lambda 함수 생성

이 단원에서는 Lambda 함수 코드를 변경하지 않고도 구성 변경을 통해 Lambda 함수의 동작을 수정할 수 있는 방법을 보여줍니다.

이 자습서에서는 다음을 수행합니다.

- Amazon S3 버킷의 이름을 지정하는 환경 변수의 값을 반환하는 샘플 코드를 통해 배포 패키지를 만듭니다.
- Lambda 함수를 호출하고, 반환된 해당 Amazon S3 버킷 이름이 환경 변수에서 설정된 값과 일치하는지 확인합니다.
- 환경 변수에서 지정된 Amazon S3 버킷 이름을 변경하여 Lambda 함수를 업데이트합니다.
- Lambda 함수를 다시 호출하고 반환된 Amazon S3 버킷 이름이 업데이트된 값과 일치하는지 확인합니다.

## Lambda 환경 설정

아래의 코드 샘플은 Amazon S3 버킷의 이름을 반환하는 Lambda 함수의 환경 변수를 읽습니다.

1. 텍스트 에디터를 열고 다음 코드를 복사합니다.

```
exports.handler = function(event, context, callback) {  
    var bucketName = process.env.S3_BUCKET;  
    callback(null, bucketName);  
};
```

2. 파일을 index.js로 저장합니다.

3. index.js 파일을 Test\_Environment\_Variables.zip으로 압축합니다.

## 실행 역할 생성

Lambda 함수를 생성할 때 지정할 수 있는 IAM 역할(실행 역할)을 생성합니다.

- AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- IAM 사용 설명서의 [IAM 역할](#)에 나와 있는 단계에 따라 IAM 역할(실행 역할)을 생성합니다. 해당 단계에 따라 역할을 생성할 때 다음 사항에 유의하십시오.
  - Select Role Type(역할 유형 선택)에서 AWS Service Roles(AWS 서비스 역할)을 선택한 후 AWS Lambda를 선택합니다.
  - [Attach Policy]에서 [AWSLambdaBasicExecutionRole]이라는 정책을 선택합니다.

- IAM 역할의 Amazon 리소스 이름(ARN)을 적어둡니다. 다음 단계에서 Lambda 함수를 생성할 때 이 값이 필요합니다.

## Lambda 함수 생성 및 테스트

이 단원에서 Test라는 이름의 Amazon S3 버킷을 지정하는 환경 변수를 포함하는 Lambda 함수를 생성합니다. 호출 시 함수는 단순히 버킷의 이름을 반환합니다. 그리고 Amazon S3 버킷 이름을 Prod로 변경하여 구성은 업데이트하고 나면, 다시 호출이 될 때 함수가 Amazon S3 버킷의 업데이트된 이름을 반환합니다.

Lambda 함수를 생성하려면 명령 프롬프트를 열어서 다음과 같은 Lambda AWS CLI `create-function` 명령을 실행합니다. .zip 파일 경로와 실행 역할 ARN을 제공해야 합니다.

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

그런 다음, 다음과 같은 Lambda CLI `invoke` 명령을 실행하여 함수를 호출합니다.

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

Lambda 함수는 Amazon S3 버킷의 이름을 "Test"로 반환합니다.

그런 다음, 다음과 같은 Lambda CLI `update-function-configuration` 명령을 사용하여 Prod 버킷을 가리키도록 하여 Amazon S3 환경 변수를 업데이트합니다.

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

동일한 파라미터를 사용하여 `aws lambda invoke` 명령을 다시 실행합니다. 이때, Lambda 함수는 Amazon S3 버킷 이름을 Prod로 반환합니다.

## 민감한 정보를 저장하도록 환경 변수를 사용하여 Lambda 함수 생성

Lambda 함수의 구성 설정을 지정하는 것 외에도 환경 변수를 사용하면 [AWS Key Management Service](#) 및 Lambda 콘솔의 암호화 도우미를 통해 데이터베이스 암호 같은 민감한 데이터를 저장할 수 있습니다. 자세한 내용은 [환경 변수 암호화 \(p. 42\)](#) 단원을 참조하십시오. 다음 예제는 민감한 데이터를 저장하는 방법을 비롯하여 KMS를 사용하여 해당 정보를 암호 해독하는 방법을 보여줍니다.

이 자습서에서는 Lambda 콘솔을 사용하여 민감한 정보가 포함된 환경 변수를 암호화하는 방법을 보여 줍니다. 기존 함수를 업데이트하는 경우 [2단계: Lambda 함수 구성 \(p. 45\)](#)의 `Environment Variables` 섹션을 확장하는 방법을 설명하는 단계를 건너뛸 수 있습니다.

### 1단계: Lambda 함수 생성

- AWS Management 콘솔에 로그인하고 <https://console.aws.amazon.com/lambda/>에서 AWS Lambda 콘솔을 엽니다.
- Create a Lambda function(Lambda 함수 생성)을 선택합니다.
- [Select blueprint]에서 [Author from scratch] 버튼을 선택합니다.
- [Basic information]에서 다음을 수행하십시오.

- 이름에 Lambda 함수 이름을 지정합니다.
- 역할에서 기존 역할 선택을 선택합니다.
- 기존 역할에서 lambda\_basic\_execution을 선택합니다.

Note

실행 역할의 정책이 decrypt 권한을 가지고 있지 않으면 이를 추가해야 합니다.

- 함수 생성을 선택합니다.

## 2단계: Lambda 함수 구성

1. [Configuration]에서 [Runtime]을 원하는 대로 지정합니다.
2. Lambda 함수 코드 섹션에서 코드 인라인 편집 옵션을 활용하여 Lambda 함수 핸들러 코드를 사용자 지정 코드로 바꿉니다.
3. [Triggers] 탭을 기록합니다. 트리거 페이지에서 트리거 추가 버튼을 선택한 다음, 사용 가능한 서비스 목록을 표시하는 줄임표(...)가 포함된 그레이 박스를 선택하여 Lambda 함수를 자동으로 트리거하는 서비스를 선택할 수 있습니다. 이 예제에서는 트리거를 구성하지 말고 [Configuration]을 선택합니다.
4. [Monitoring] 탭을 기록합니다. 이 페이지는 Lambda 함수 호출을 위한 즉각적인 CloudWatch 측정치를 비롯하여 [AWS X-Ray 사용 \(p. 226\)](#)를 포함한 기타 유용한 가이드에 대한 링크를 제공합니다.
5. [Environment variables] 섹션을 확장합니다.
6. 키-값 페어를 입력합니다. [Encryption configuration] 섹션을 확장합니다. Lambda는 업로드 이후에 정보를 암호화할 수 있도록 유휴 시 암호화할 KMS 키 아래에 기본 서비스 키를 제공합니다. 제공된 정보가 민감한 정보인 경우, [Enable helpers for encryption in transit] 확인란을 추가적으로 체크하고 사용자 지정 키를 제공할 수 있습니다. 이렇게 하면 입력한 값을 마스킹하고 AWS KMS를 호출하여 값을 암호화하고 이를 Ciphertext로 반환합니다. 계정에 대해 KMS 키를 생성하지 않았다면 키를 생성할 수 있도록 AWS IAM 콘솔에 대한 링크가 제공됩니다. 계정에는 해당 키에 대한 encrypt 및 decrypt 권한이 있어야 합니다. 선택 이후에 [Encrypt] 단추가 [Decrypt]로 전환됩니다. 이렇게 하면 선택에 따라 정보를 업데이트할 수 있습니다. 정보가 업데이트되면 [Encrypt] 버튼을 선택합니다.

코드 버튼은 애플리케이션에서 사용할 수 있는 Lambda 함수의 런타임에 고유한 샘플 암호 해독 코드를 제공합니다.

Note

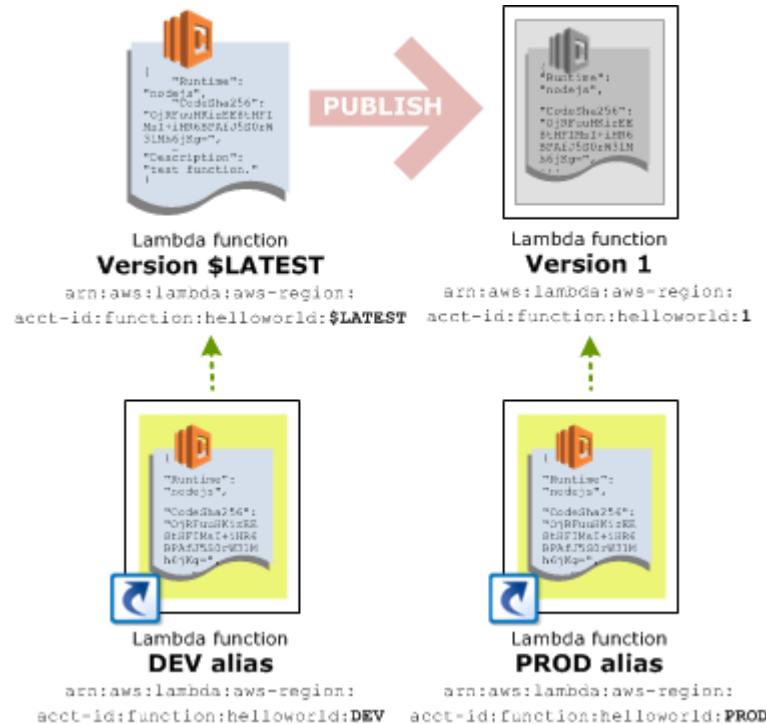
클라이언트 측에서 민감한 정보를 암호화하기 위해 기본 Lambda 서비스 키를 사용할 수 없습니다. 자세한 내용은 [환경 변수 암호화 \(p. 42\)](#) 단원을 참조하십시오.

## AWS Lambda 함수 버전 관리 및 별칭

버전 관리를 사용하여 AWS Lambda에서 프로덕션 함수를 효율적으로 관리할 수 있습니다. AWS Lambda에서 버전 관리를 사용할 때 Lambda 함수에 대한 하나 이상의 버전을 게시할 수 있습니다. 그 결과, 개발 워크플로우에서 개발, 베타, 프로덕션 등 다양하게 변형된 함수로 작업할 수 있습니다.

각 Lambda 함수 버전은 고유의 Amazon 리소스 이름(ARN)을 가집니다. 버전을 게시한 후 변경할 수 있습니다.

또한 AWS Lambda는 각 Lambda 함수 버전의 별칭을 생성할 수 있도록 지원합니다. 개념적으로, AWS Lambda 별칭은 특정 Lambda 함수 버전을 가리키는 포인터입니다. 또한 리소스는 함수와 유사하며 각 별칭에는 고유 ARN이 있습니다. 각 별칭은 별칭이 가리키는 함수 버전에 대한 ARN을 유지합니다. 별칭은 다른 별칭이 아니라 함수 버전만 가리킬 수 있습니다. 버전과 달리 별칭은 수정할 수 있습니다. 상이한 버전을 가리키도록 업데이트할 수 있습니다.



별칭을 사용하면 Lambda 함수 버전 및 이벤트 리소스의 매핑에서 새로운 Lambda 함수 버전을 프로덕션 환경에 승격하는 과정을 추상화할 수 있습니다.

예를 들어 Amazon S3를 버킷에서 객체가 새로 생성될 때 Lambda 함수를 호출하는 이벤트 소스라고 가정해 보십시오. 그 이벤트 소스일 때는 버킷 알림 구성에 이벤트 소스 매핑 정보를 저장합니다. 이 구성에서는 Amazon S3가 호출할 수 있는 Lambda 함수 ARN을 식별할 수 있지만, 이 경우 Amazon S3가 올바른 버전을 호출할 수 있도록 새로운 버전의 Lambda 함수를 게시할 때마다 알림 구성을 업데이트해야 합니다.

함수 ARN을 지정하는 대신 알림 구성에서 별칭 ARN을 지정할 수 있습니다(예를 들어 PROD 별칭 ARN). 새로운 버전의 Lambda 함수를 프로덕션 환경으로 승격함에 따라 안정적인 최신 버전을 가리키도록 PROD 별칭을 업데이트하기만 하면 됩니다. Amazon S3에서 알림 구성을 업데이트할 필요가 없습니다.

Lambda 함수의 이전 버전으로 룰백을 해야 할 때도 동일한 원칙이 적용됩니다. 이 시나리오에서는 다른 함수 버전을 가리키도록 PROD 별칭을 업데이트만 하면 될 뿐, 이벤트 소스 매핑을 업데이트할 필요가 없습니다.

여러 종속 관계 및 개발자가 관여하는 애플리케이션을 구축할 때는 버전 관리 및 별칭을 사용하여 Lambda 함수를 배포하는 것이 좋습니다.

### 주제

- [AWS Lambda 버전 관리 소개 \(p. 47\)](#)
- [AWS Lambda 별칭 소개 \(p. 50\)](#)
- [버전 관리, 별칭 및 리소스 정책 \(p. 57\)](#)
- [AWS Management 콘솔, AWS CLI 또는 Lambda API 작업을 사용하여 버전 관리 \(p. 58\)](#)
- [별칭을 사용한 트래픽 이동 \(p. 60\)](#)

## AWS Lambda 버전 관리 소개

다음으로는, Lambda 함수를 생성하고 여기에서 버전을 게시하는 방법을 설명합니다. 또한 하나 이상의 버전이 게시되었을 때 함수 코드 및 구성 정보를 업데이트하는 방법을 설명합니다. 뿐만 아니라 특정 버전이나 전체 함수(모든 버전 및 연결 별칭 포함) 등 함수 버전을 삭제하는 방법에 대한 정보도 포함되어 있습니다.

### Lambda 함수 생성(\$LATEST 버전)

Lambda 함수를 생성할 때는 오직 하나의 버전인 \$LATEST 버전만 있습니다.



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

Amazon 리소스 이름(ARN)을 사용하여 이 함수를 지칭할 수 있습니다. 이 초기 버전에는 두 개의 ARN이 연결되어 있습니다.

- 정규화된 – 버전 접미사가 포함된 함수 ARN입니다.

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

- 정규화 되지 않은 ARN – 버전 접미사가 포함되지 않은 함수 ARN입니다.

모든 관련 작업에서 정규화되지 않은 이 ARN을 사용할 수 있습니다. 그러나 별칭을 생성하는 데는 사용할 수 없습니다. 자세한 내용은 [AWS Lambda 별칭 소개 \(p. 50\)](#) 단원을 참조하십시오.

정규화되지 않은 ARN은 자체 리소스 정책을 가지고 있습니다.

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

#### Note

버전 게시를 선택하지 않는 경우, \$LATEST 함수 버전만이 유일한 Lambda 함수 버전으로 존재하게 됩니다. 이벤트 소스 매핑에서 정규화된 ARN이나 정규화되지 않은 ARN을 사용하여 이 \$LATEST 버전을 호출할 수 있습니다.

다음은 CreateFunction API 호출 응답의 예입니다.

```
{  
    "CodeSize": 287,  
    "Description": "test function."  
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",  
    "FunctionName": "helloworld",  
    "Handler": "helloworld.handler",  
    "LastModified": "2015-07-16T00:34:31.322+0000",  
    "MemorySize": 128,  
    "Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",  
}
```

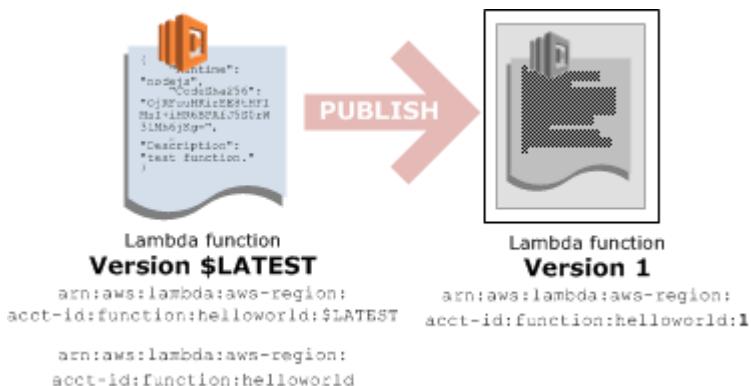
```
    "Runtime": "nodejs6.10",
    "Timeout": 3,
    "CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",
    "Version": "$LATEST"
}
```

자세한 내용은 [CreateFunction \(p. 347\)](#) 단원을 참조하십시오.

이 응답에서 AWS Lambda는 새로 생성된 함수의 정규화되지 않은 ARN 및 버전인 \$LATEST를 반환합니다. 또한 응답에서는 Version이 \$LATEST임을 보여줍니다. CodeSha256은 업로드한 배포 패키지의 체크섬입니다.

## AWS Lambda 함수 버전 게시

버전을 게시하면 AWS Lambda는 \$LATEST 버전에 Lambda 함수 코드 및 구성의 스냅샷 복사본을 만듭니다. 게시된 버전은 변경할 수 없습니다. 즉, 코드나 구성 정보를 바꿀 수 없습니다. 새 버전은 아래와 같이 버전 번호 접미사가 포함된 고유한 ARN을 가집니다.



다음과 같은 방법을 사용하여 버전을 게시할 수 있습니다.

- 버전을 명시적으로 게시 – PublishVersion API 작업을 사용하여 버전을 명시적으로 게시할 수 있습니다. 자세한 내용은 [PublishVersion \(p. 422\)](#) 단원을 참조하십시오. 이 작업은 \$LATEST 버전의 코드 및 구성을 사용하여 새 버전을 생성합니다.
- Lambda 함수를 생성 또는 업데이트할 때 버전 게시 – CreateFunction 또는 UpdateFunctionCode 요청을 사용하여 요청에 publish 파라미터 옵션을 추가하는 방법으로 버전을 게시할 수도 있습니다.
  - 새 Lambda 함수(\$LATEST 버전)를 생성하려면 CreateFunction 요청에 publish 파라미터를 지정합니다. 그런 다음 스냅샷을 생성하고 이를 버전 1로 할당하여 새 함수를 즉시 게시할 수 있습니다. For more information about CreateFunction, see [CreateFunction \(p. 347\)](#).
  - \$LATEST 버전의 코드를 업데이트하려면 UpdateFunctionCode 요청에 publish 파라미터를 지정합니다. 그런 다음 \$LATEST에서 버전을 게시할 수 있습니다. For more information about UpdateFunctionCode, see [UpdateFunctionCode \(p. 448\)](#).

Lambda 함수를 만들 때 publish 파라미터를 지정하면 AWS Lambda가 응답으로 반환하는 함수 구성 정보에 아래와 같이 새로 게시된 버전의 버전 번호가 표시됩니다. 다음 예제에서 버전은 1입니다.

```
{
    "CodeSize": 287,
    "Description": "test function."
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",
    "FunctionName": "helloworld",
    "Handler": "helloworld.handler",
    "LastModified": "2015-07-16T00:34:31.322+0000",
    "MemorySize": 128,
    "Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",
    "Runtime": "nodejs6.10",
```

```
    "Timeout": 3,
    "CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",
    "Version": "1"
}
```

### Note

코드가 아직 게시되지 않았거나 \$LATEST 버전과 비교해 코드가 바뀐 경우, Lambda는 새 버전만 게시합니다. 바뀐 것이 없는 경우에는 \$LATEST 게시 버전이 반환됩니다.

Lambda 함수를 생성하거나 Lambda 함수 코드를 업데이트할 때 버전을 게시하는 것이 좋습니다. 여러 명의 개발자가 동일한 함수 개발에 참여하고 있는 경우에는 특히 그렇습니다. 요청에서 publish 파라미터를 사용하면 이렇게 할 수 있습니다.

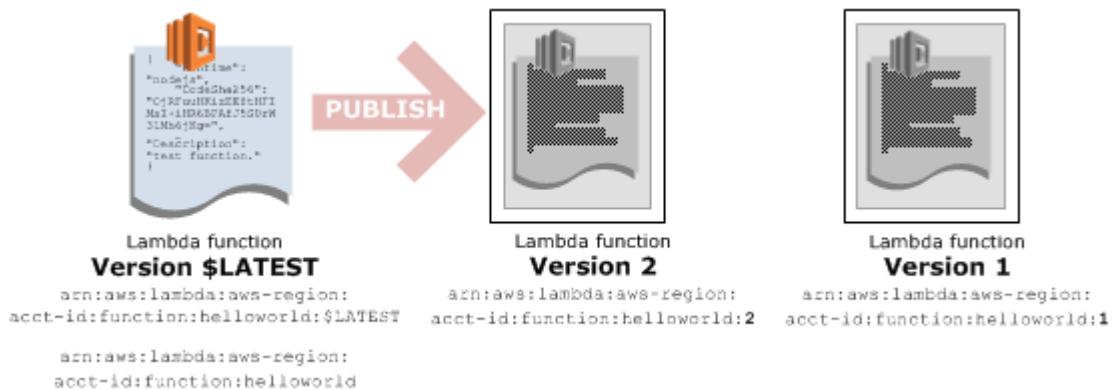
여러 명의 개발자가 한 프로젝트에 참여하고 있는 경우, 개발자 A가 Lambda 함수(\$LATEST 버전)를 생성하고, 개발자 A가 이 버전을 게시하기 전에 개발자 B가 \$LATEST 버전과 연결된 코드(배포 패키지)를 업데이트 할 수 있습니다. 이 경우 개발자 A가 업로드한 원본 코드가 손실됩니다. 두 개발자가 모두 publish 파라미터를 추가하면 앞서 설명한 교착 상태를 막을 수 있습니다.

각 Lambda 함수 버전은 Amazon 리소스 이름(ARN)이 포함된 고유의 리소스를 가집니다. 다음 예제는 helloworld Lambda 함수의 버전 번호 1에 대한 ARN을 보여줍니다.

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

Lambda 함수의 여러 버전을 게시할 수 있습니다. 버전을 게시할 때마다 AWS Lambda는 \$LATEST 버전(코드 및 구성 정보)을 복사하여 새 버전을 생성합니다. 추가 버전을 게시하는 경우, 는 함수가 삭제되거나 재생성되더라도 버전 관리를 위해 단순 증가하는 시퀀스 번호를 할당합니다. 삭제 및 재생성된 함수라 할지라도 버전 번호가 재사용되지 않기 때문에 해당 함수 버전의 사용자는 해당 버전의 실행 파일이 절대로 바뀌지 않는다는 점에 안심할 수 있습니다(삭제된 경우 제외).

한정자를 재사용하고 싶으면 버전과 함께 별칭을 사용합니다. 같은 이름으로 별칭을 삭제 및 재생성할 수 있습니다.



## Lambda 함수 코드 및 구성 업데이트

AWS Lambda는 \$LATEST 버전에 최신 함수 코드를 유지합니다. 함수 코드를 업데이트하면 AWS Lambda는 Lambda 함수의 \$LATEST 버전에서 코드를 대체합니다. 자세한 내용은 [UpdateFunctionCode \(p. 448\)](#) 단원을 참조하십시오.

다음과 같은 옵션을 통해 Lambda 함수 코드를 업데이트할 때 새 버전을 게시할 수 있습니다.

- 동일한 업데이트 코드 요청에 버전 게시 – `UpdateFunctionCode` API 작업을 사용합니다(권장 사항).
- 먼저 코드를 업데이트한 다음, 버전을 명시적으로 게시 – `PublishVersion` API 작업을 사용합니다.

Lambda 함수의 `$LATEST` 버전에 대한 코드 및 구성 정보(설명, 메모리 크기 및 실행 제한 시간)를 업데이트 할 수 있습니다.

## Lambda 함수 및 특정 버전 삭제

버전 관리를 위해 다음과 같은 옵션이 제공됩니다.

- 특정 버전 삭제 – `DeleteFunction` 요청에서 삭제하고 싶은 버전을 지정하여 Lambda 함수 버전을 삭제할 수 있습니다. 이 버전에 종속된 별칭이 있을 경우 요청이 실패합니다. 이 버전에 종속된 별칭이 없는 경우에만 AWS Lambda가 버전을 삭제합니다. 별칭에 대한 자세한 내용은 [AWS Lambda 별칭 소개 \(p. 50\)](#)을 참조하십시오.
- 전체 Lambda 함수(모든 버전 및 별칭)를 삭제 – Lambda 함수와 모든 버전을 삭제하려면 `DeleteFunction` 요청에 어떤 버전도 지정해서는 안 됩니다. 이렇게 하면 모든 버전 및 별칭을 포함하여 전체 함수가 삭제됩니다.

## AWS Lambda 별칭 소개

Lambda 함수의 별칭을 하나 이상 생성할 수 있습니다. AWS Lambda 별칭은 특정 Lambda 함수 버전을 가리키는 포인터와 같습니다. 버전 관리에 대한 자세한 내용은 [AWS Lambda 버전 관리 소개 \(p. 47\)](#) 단원을 참조하십시오.

별칭을 사용하면 호출자가 별칭이 가리키는 특정 버전을 모르더라도 별칭이 가리키는 Lambda 함수(예: 함수 호출)에 액세스할 수 있습니다.

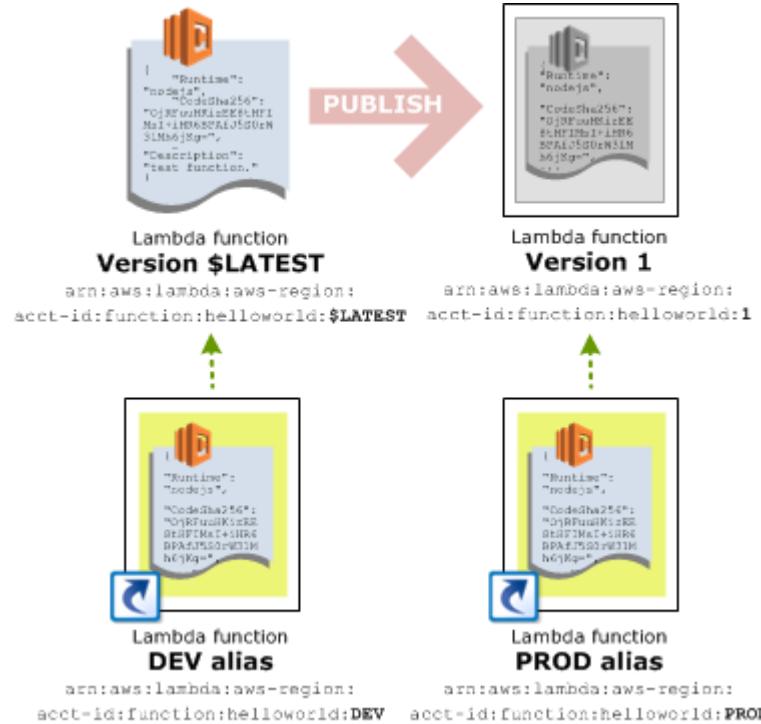
AWS Lambda 별칭의 사용 사례는 다음과 같습니다.

- 필요 시 새로운 버전의 Lambda 함수를 손쉽게 승격 및 룰백할 수 있도록 지원 – 처음에 Lambda 함수 (`$LATEST` 버전)를 생성하고 나면 버전 1을 게시할 수 있습니다. 버전 1을 가리키는 PROD라는 별칭을 생성하고 나면 이 PROD 별칭을 사용하여 함수의 버전 1을 호출할 수 있습니다.

이제 모든 개선 사항이 반영하여 코드(`$LATEST` 버전)를 업데이트한 다음, 안정적이고 향상된 또 다른 버전(버전 2)을 게시할 수 있습니다. PROD 별칭을 다시 매핑하여 버전 2를 프로덕션으로 승격하여 버전 2를 가리키도록 할 수 있습니다. 잘못된 점이 있는 경우, PROD 별칭을 다시 매핑하여 프로덕션 버전을 쉽게 버전 1로 룰백하여 버전 1을 가리키도록 할 수 있습니다.

- 이벤트 소스 매핑의 관리 간소화 – 이벤트 소스 매핑에 Lambda 함수의 ARN(Amazon 리소스 이름)을 사용하는 대신 별칭 ARN을 사용할 수 있기 때문에 새 버전을 승격하거나 이전 버전으로 룰백을 할 때 이벤트 소스 매핑을 업데이트할 필요가 없습니다.

Lambda 함수와 별칭 모두 AWS Lambda 리소스이며, 다른 모든 AWS 리소스와 마찬가지로 고유한 ARN을 가집니다. 다음 예제는 Lambda 함수(`$LATEST` 버전)와 함께 게시된 버전을 보여줍니다. 각 버전은 이를 가리키는 별칭을 가집니다.



함수 ARN이나 별칭 ARN을 사용하여 함수에 액세스할 수 있습니다.

- 정규화되지 않은 함수에 대한 함수 버전이 항상 \$LATEST에 매핑되기 때문에 정규화되거나 정규화되지 않은 함수 ARN을 사용하여 액세스가 가능합니다. 다음 예제는 \$LATEST 버전 접미사가 포함된 정규화된 함수 ARN을 보여줍니다.

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

- 별칭 ARN을 사용할 때는 정규화된 ARN을 사용하게 됩니다. 각 별칭 ARN에는 별칭 이름 접미사가 포함되어 있습니다.

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

AWS Lambda는 별칭을 생성하고 관리할 수 있도록 다음과 같은 API 작업을 제공합니다.

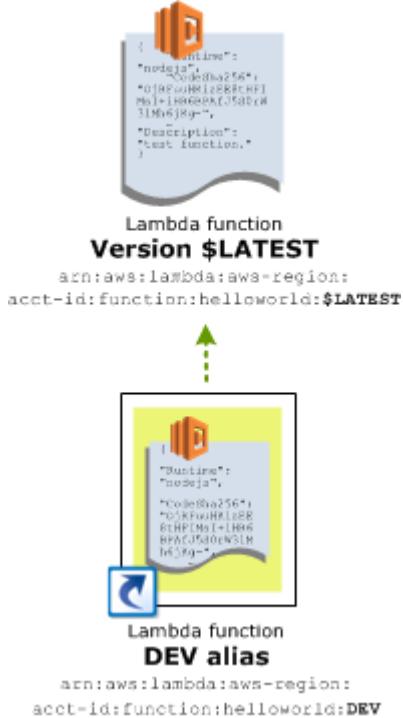
- [CreateAlias \(p. 339\)](#)
- [UpdateAlias \(p. 440\)](#)
- [GetAlias \(p. 368\)](#)
- [ListAliases \(p. 399\)](#)
- [DeleteAlias \(p. 355\)](#)

## 예제: 별칭을 사용하여 Lambda 함수 버전 관리

다음은 버전 관리 및 별칭을 사용하여 Lambda 함수의 새로운 버전을 프로덕션으로 승격하는 방법에 대한 예제입니다.

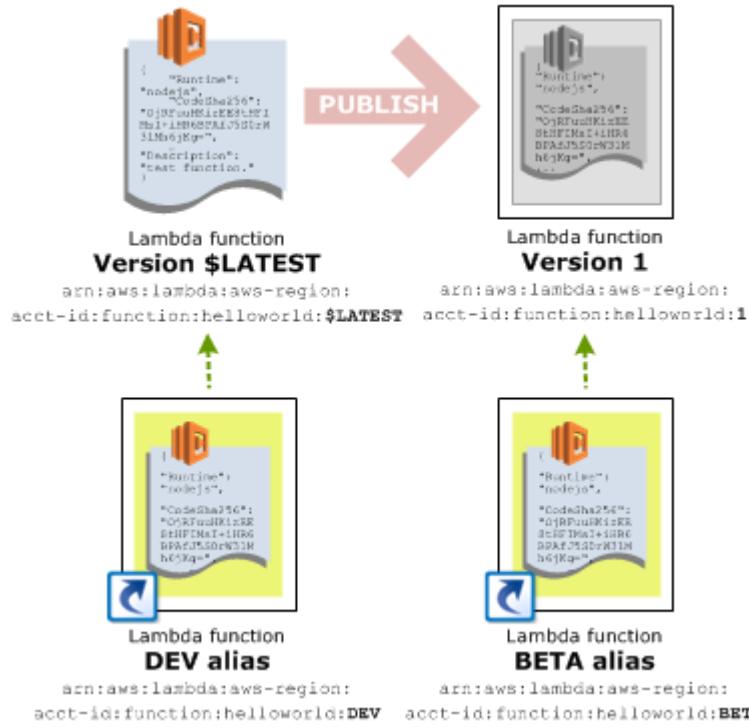
먼저, Lambda 함수를 생성합니다.

생성한 함수는 **\$LATEST** 버전입니다. 또한 새로 생성된 함수를 가리키는 별칭(개발용인 경우 **DEV**)을 생성합니다. 개발자는 이 별칭을 사용하여 개발 환경의 이벤트 리소스를 사용하여 함수를 테스트할 수 있습니다.



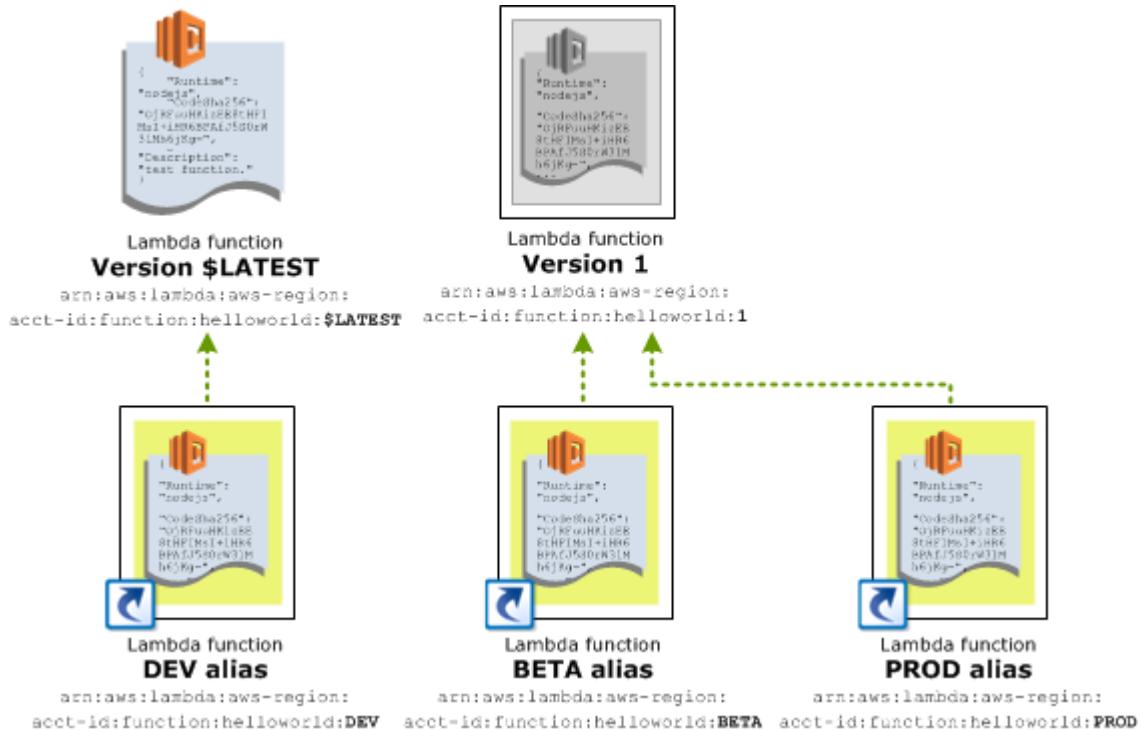
이후 최신 버전을 계속 개발하면서 안정적인 방식으로 베타 환경에서 이벤트 소스를 사용하여 함수 버전을 테스트합니다.

**\$LATEST**에서 버전을 게시하고 또 다른 별칭(BETA)이 이를 가리키도록 합니다. 이렇게 하면 특정 별칭에 베타 이벤트 소스를 연결할 수 있습니다. 이벤트 소스 매핑 시 BETA 별칭을 사용하여 함수와 이벤트 소스를 연결합니다.



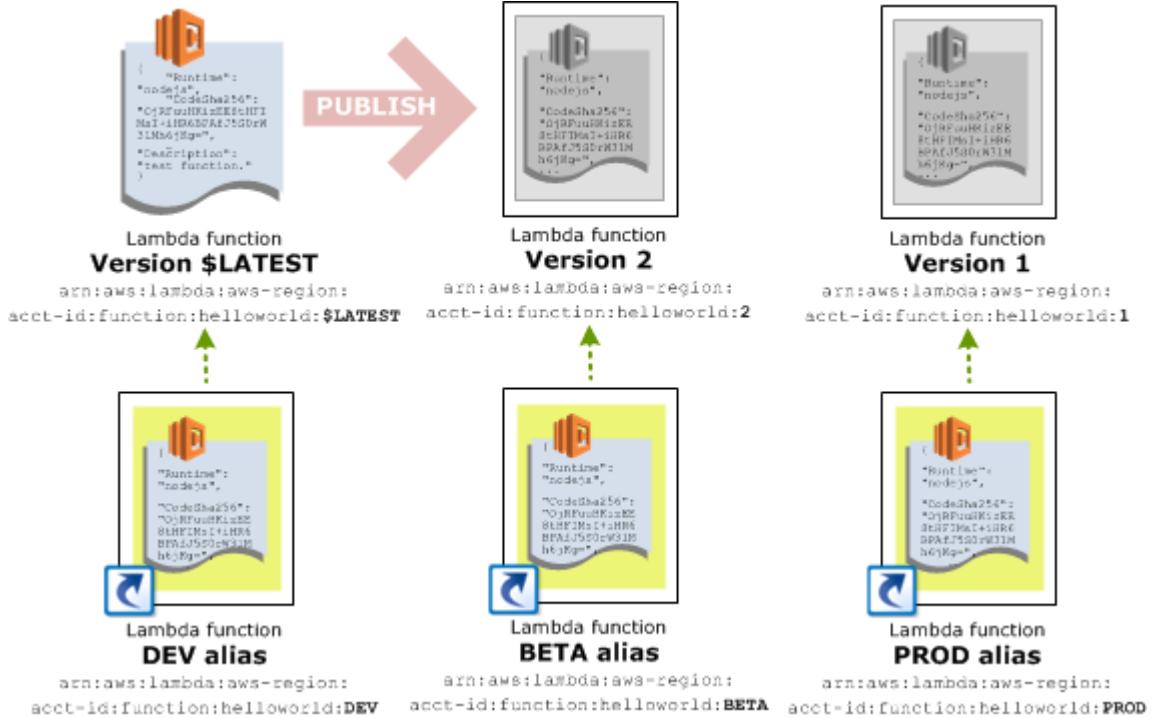
그런 다음 프로덕션 환경의 이벤트 소스로 작업할 수 있도록 프로덕션의 Lambda 함수 버전을 승격합니다.

함수의 BETA 버전을 테스트하고 나면 버전 1에 매핑되는 별칭을 생성하여 프로덕션 버전을 정의할 수 있습니다. 이렇게 하면 프로덕션 이벤트 소스가 특정 버전을 가리키도록 할 수 있습니다. PROD 별칭을 생성하여 모든 프로덕션 이벤트 소스 매핑에서 PROD 별칭 ARN을 사용하면 됩니다.



개발을 계속하고 더 많은 버전을 게시하고 테스트합니다.

코드를 개발하는 과정에서 업데이트된 코드를 업로드하여 \$LATEST 버전을 업데이트한 다음, BETA 별칭이 이를 가리키도록 하여 베타 테스트 환경에 게시할 수 있습니다. 이렇게 베타 별칭의 재매핑이 간단하기 때문에 이벤트 소스를 변경하지 않고도 함수의 버전 2를 베타 환경에 게시할 수 있습니다. 이런 방식으로 별칭은 개발 환경의 특정 이벤트 소스에서 사용되는 함수의 버전을 관리할 수 있게 해줍니다.



AWS Command Line Interface를 사용하여 이 설정의 생성을 시도하고 싶은 경우에는 [자습서: AWS Lambda 별칭 사용 \(p. 54\)](#) 단원을 참조하십시오.

## 자습서: AWS Lambda 별칭 사용

이 AWS CLI 기반 자습서는 [예제: 별칭을 사용하여 Lambda 함수 버전 관리 \(p. 51\)](#)에 설명된 대로 Lambda 함수 버전과 이를 가리키는 별칭을 생성합니다.

이 예제는 us-west-2(미국 서부 오리건) 리전을 사용하여 Lambda 함수 및 별칭을 생성합니다.

1. Lambda 함수를 생성하기 위해 업로드할 수 있는 배포 패키지를 생성합니다.

- a. 텍스트 편집기를 열고 다음 코드를 복사합니다.

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

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

- b. 파일을 helloworld.js로 저장합니다.
- c. helloworld.js 파일을 helloworld.zip로 압축합니다.

2. Lambda 함수를 생성할 때 지정할 수 있는 AWS Identity and Access Management(IAM) 역할(실행 역할)을 생성합니다.
  - a. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
  - b. IAM 사용 설명서의 [IAM 역할](#)에 나와 있는 단계에 따라 IAM 역할(실행 역할)을 생성합니다. 해당 단계에 따라 역할을 생성할 때 다음 사항에 유의하십시오.
    - 역할 유형 선택에서 AWS Service Roles(AWS 서비스 역할)를 선택한 다음 AWS Lambda를 선택합니다.
    - [Attach Policy]에서 [AWSLambdaBasicExecutionRole]이라는 정책을 선택합니다.
  - c. IAM 역할의 Amazon 리소스 이름(ARN)을 적어둡니다. 다음 단계에서 Lambda 함수를 생성할 때 이 값이 필요합니다.
3. Lambda 함수(helloworld)를 만듭니다.

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

아래 예제에서 알 수 있듯이 응답은 함수 버전으로 \$LATEST를 표시하는 구성 정보를 반환합니다.

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

4. helloworld Lambda 함수의 \$LATEST 버전을 가리키는 별칭(DEV)을 생성합니다.

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

응답은 별칭이 가리키는 함수 버전과 별칭 ARN을 포함한 별칭 정보를 반환합니다. ARN은 별칭 이름 접미사가 포함된 함수 ARN과 같습니다. 다음은 응답의 예입니다.

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

5. helloworld Lambda 함수의 버전을 게시합니다.

```
aws lambda publish-version --function-name helloworld
```

응답은 버전 번호, 버전 접미사가 포함된 함수 ARN과 같은 함수 버전의 구성 정보를 반환합니다. 다음은 응답의 예입니다.

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

6. helloworld Lambda 함수 버전 1에 대해 BETA라는 별칭을 생성합니다.

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

이제 helloworld 함수에 대한 별칭이 두 개가 되었습니다. DEV 별칭은 \$LATEST 함수 버전을 가리키고 BETA 별칭은 Lambda 함수의 버전 1을 가리킵니다.

7. helloworld 함수의 버전 1을 프로덕션으로 승격하고 싶은 경우를 생각해 봅시다. 버전 1을 가리키는 또 다른 별칭(PROD)을 생성합니다.

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

이때 모든 BETA 및 PROD 별칭은 Lambda 함수의 버전 1을 가리키게 됩니다.

8. 이제 최신 버전(예를 들어 버전 2)을 게시할 수 있지만, 먼저 코드를 업데이트하고 수정된 배포 패키지를 업로드해야 합니다. \$LATEST 버전이 변경되지 않은 경우에는 하나 이상의 버전을 게시할 수 없습니다. 배포 패키지를 업데이트하여 업로드하고 버전 2를 게시했다고 한다면 이제 Lambda 함수의 버전 2를 가리키는 BETA 별칭을 변경할 수 있습니다.

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

이제 세 개의 별칭이 Lambda 함수의 상이한 버전을 가리킵니다. DEV 별칭은 \$LATEST 버전을, BETA 별칭은 버전 2를, PROD 별칭은 Lambda 함수의 버전 1을 가리킵니다.

AWS Lambda 콘솔을 사용하여 버전을 관리하는 방법은 [AWS Management 콘솔](#), [AWS CLI](#) 또는 [Lambda API](#) 작업을 사용하여 버전 관리 (p. 58) 단원을 참조하십시오.

## 푸시 모델에서 권한 부여

푸시 모델(AWS Lambda 이벤트 소스 매핑 (p. 80) 참조)에서는 Amazon S3 같은 이벤트 소스가 Lambda 함수를 호출합니다. 이들 이벤트 소스는 이벤트 발생 시 호출되는 함수 버전이나 별칭을 식별하는 매핑을 유지합니다. 다음을 참조하십시오.

- 매핑 구성에서 기존의 Lambda 함수 별칭을 지정하는 것이 좋습니다([AWS Lambda 별칭 소개 \(p. 50\)](#) 참조). 예를 들어 이벤트 소스가 Amazon S3라면 Amazon S3가 특정 이벤트를 감지하는 별칭을 호출할 수 있도록 버킷 알림 구성에서 별칭 ARN을 지정합니다.
- 푸시 모델에서는 Lambda 함수에 연결하는 리소스 정책을 사용하여 이벤트 리소스 권한을 부여합니다. 버전 관리에서 추가한 권한은 AddPermission 요청에서 지정한 한정자에 고유합니다([버전 관리, 별칭 및 리소스 정책 \(p. 57\)](#) 참조).

예를 들어 다음의 AWS CLI 명령은 Amazon S3에게 helloworld Lambda 함수의 PROD 별칭을 호출할 수 있는 권한을 부여합니다(--qualifier 파라미터가 별칭 이름을 지정).

```
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 111111111111
```

이 경우, Amazon S3는 PROD 별칭을 호출할 수 있게 되고 AWS Lambda는 PROD 별칭이 가리키는 helloworld Lambda 함수 버전을 실행할 수 있습니다. 이를 위해서는 S3 버킷의 알림 구성에서 PROD 별칭 ARN을 사용해야 합니다.

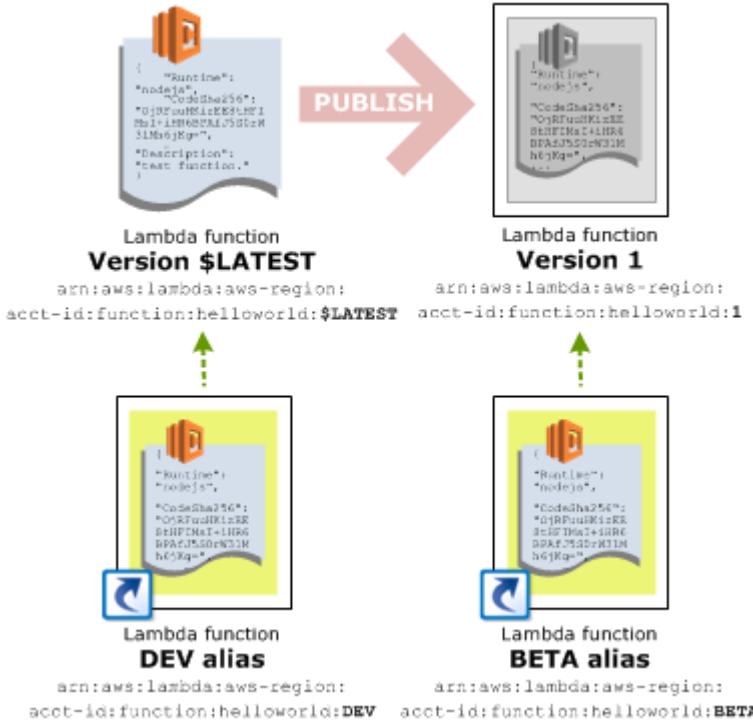
Amazon S3 이벤트를 처리하는 방법에 대한 자세한 내용은 [자습서: Amazon S3과 함께 AWS Lambda 사용 \(p. 190\)](#) 단원을 참조하십시오.

Note

AWS Lambda 콘솔을 사용하여 Lambda 함수에 이벤트 소스를 추가하면 콘솔이 사용자에게 필요 한 권한을 주어줍니다.

## 버전 관리, 별칭 및 리소스 정책

버전 관리 및 별칭에서는 다양한 ARN을 사용하여 Lambda 함수에 액세스할 수 있습니다. 예를 들어 다음 시나리오를 고려해 보십시오.



예를 들어 다음의 두 ARN 중 하나를 사용하여 helloworld 함수 버전 1을 호출할 수 있습니다.

- 다음과 같이 정규화된 함수 ARN을 사용합니다.

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

- 다음과 같이 BETA 별칭 ARN을 사용합니다.

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

푸시 모델에서는 Lambda 함수에 연결된 액세스 정책을 사용하여 이러한 이벤트 소스에 필요한 권한을 부여하기만 하면 이벤트 소스(예: Amazon S3 및 사용자 지정 애플리케이션)가 Lambda 함수 버전을 자유롭게 호출할 수 있습니다. 푸시 모델에 대한 자세한 내용은 [AWS Lambda 이벤트 소스 매핑 \(p. 80\)](#)을 참조하십시오.

권한을 부여했다고 가정한다면 이제 문제는 "이벤트 소스가 연결된 ARN 중 하나를 사용하여 함수 버전을 호출할 수 있는가?"입니다. 호출 가능 여부는 권한 추가 요청에서 함수를 식별한 방법에 따라 결정됩니다 ([AddPermission \(p. 335\)](#) 참조). 권한 추가 요청에서 사용된 ARN에만 부여한 권한이 적용된다는 것을 반드시 기억하셔야 합니다.

- 정규화된 함수 이름(예: helloworld:1)을 사용하는 경우에는 오직 정규화된 ARN을 사용하여 helloworld 함수 버전 1을 호출할 때만 권한이 유효합니다(다른 ARN을 사용하면 권한 오류가 발생).
- 별칭 이름(예: helloworld:BETA)을 사용하는 경우에는 BETA 별칭 ARN을 사용하여 helloworld 함수를 호출할 때만 권한이 유효합니다(다른 ARN을 사용하면 별칭이 가리키는 함수 버전 ARN을 포함하여 권한 오류가 발생).
- 정규화되지 않은 함수 이름(예: helloworld)을 사용하는 경우에는 정규화되지 않은 함수 ARN을 사용하여 helloworld 함수를 호출한 경우에만 권한이 유효합니다(다른 ARN을 사용하면 권한 오류가 발생).

#### Note

액세스 정책은 정규화되지 않은 ARN에만 적용되지만 호출된 Lambda 함수의 코드 및 구성은 여전히 함수 버전 \$LATEST에서 나온 것입니다. 정규화되지 않은 함수 ARN은 \$LATEST 버전에 맵핑되지만, 추가한 권한은 ARN에 고유합니다.

- \$LATEST 버전(helloworld:\$LATEST)을 사용하여 정규화된 함수 이름을 사용하는 경우에는 오직 정규화된 ARN을 사용하여 helloworld 함수 버전 \$LATEST를 호출할 때만 권한이 유효합니다(정규화되지 않은 ARN을 사용하면 권한 오류가 발생).

## AWS Management 콘솔, AWS CLI 또는 Lambda API 작업을 사용하여 버전 관리

AWS SDK를 사용(필요할 경우 직접 AWS Lambda API 호출)하거나 AWS Command Line Interface(AWS CLI) 또는 AWS Lambda 콘솔을 사용하여 프로그래밍 방식으로 Lambda 함수 버전을 관리할 수 있습니다.

AWS Lambda는 버전과 별칭을 관리하도록 다음과 같은 API를 제공합니다.

[PublishVersion \(p. 422\)](#)

[ListVersionsByFunction \(p. 415\)](#)

[CreateAlias \(p. 339\)](#)

[UpdateAlias \(p. 440\)](#)

[DeleteAlias \(p. 355\)](#)

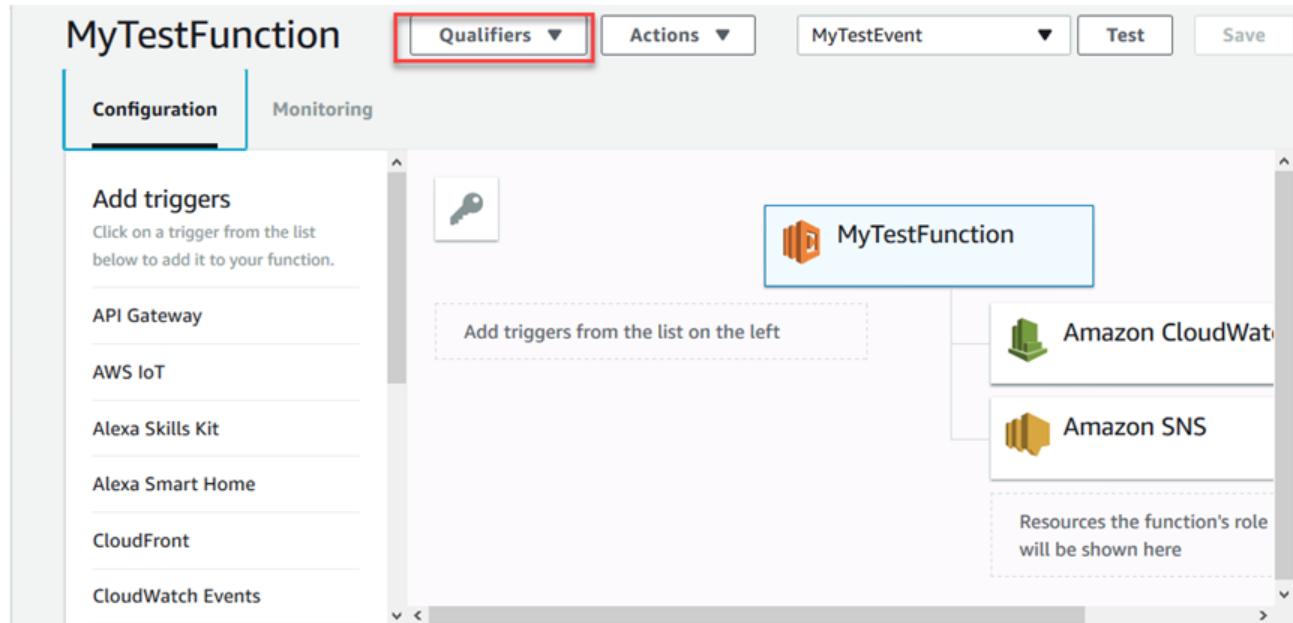
[GetAlias \(p. 368\)](#)

[ListAliases \(p. 399\)](#)

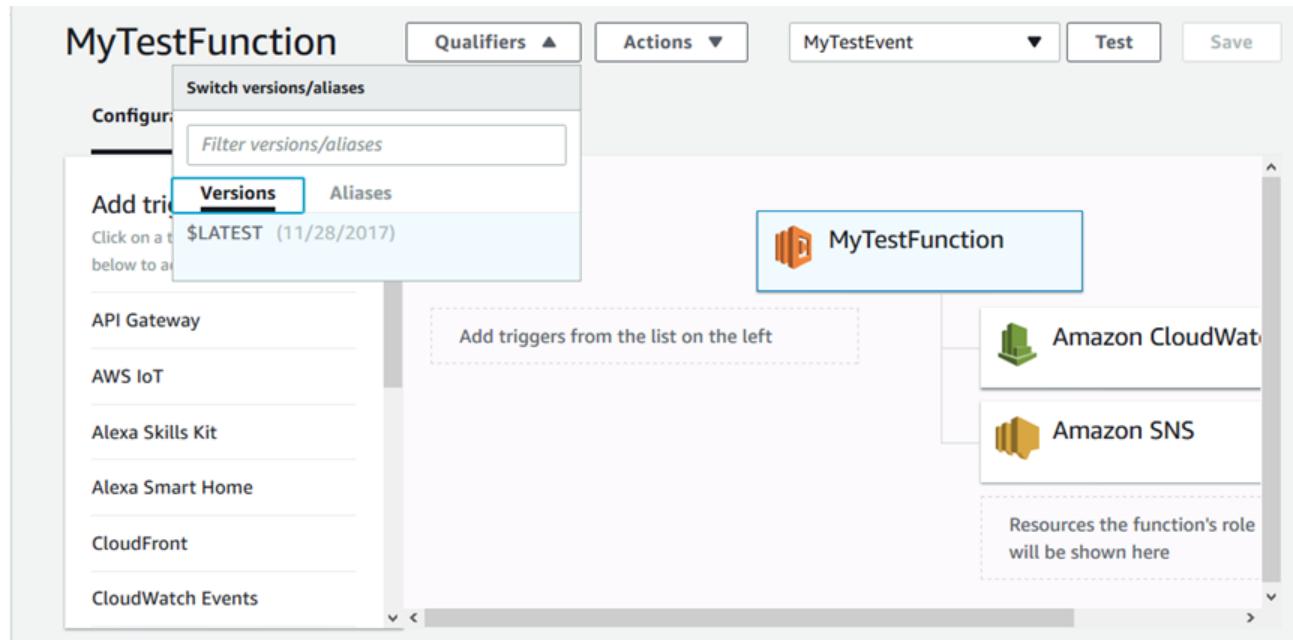
이러한 API 외에도 기존의 관련 API들이 버전 관리와 관련된 작업을 지원합니다.

AWS CLI를 사용하는 방법에 대한 예제는 [자습서: AWS Lambda 별칭 사용 \(p. 54\)](#) 단원을 참조하십시오.

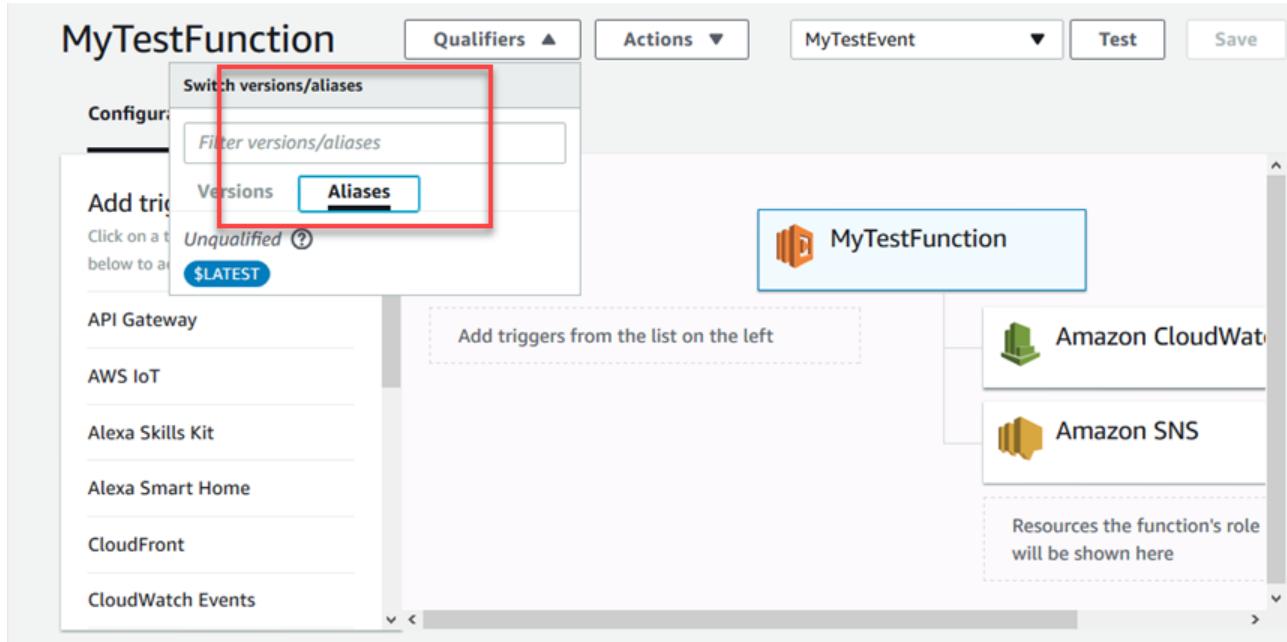
다음 단원에서는 AWS Lambda 콘솔을 사용하여 버전을 관리할 수 있는 방법에 대해 설명합니다. AWS Lambda 콘솔에서 함수를 선택한 다음 한정자를 선택합니다.



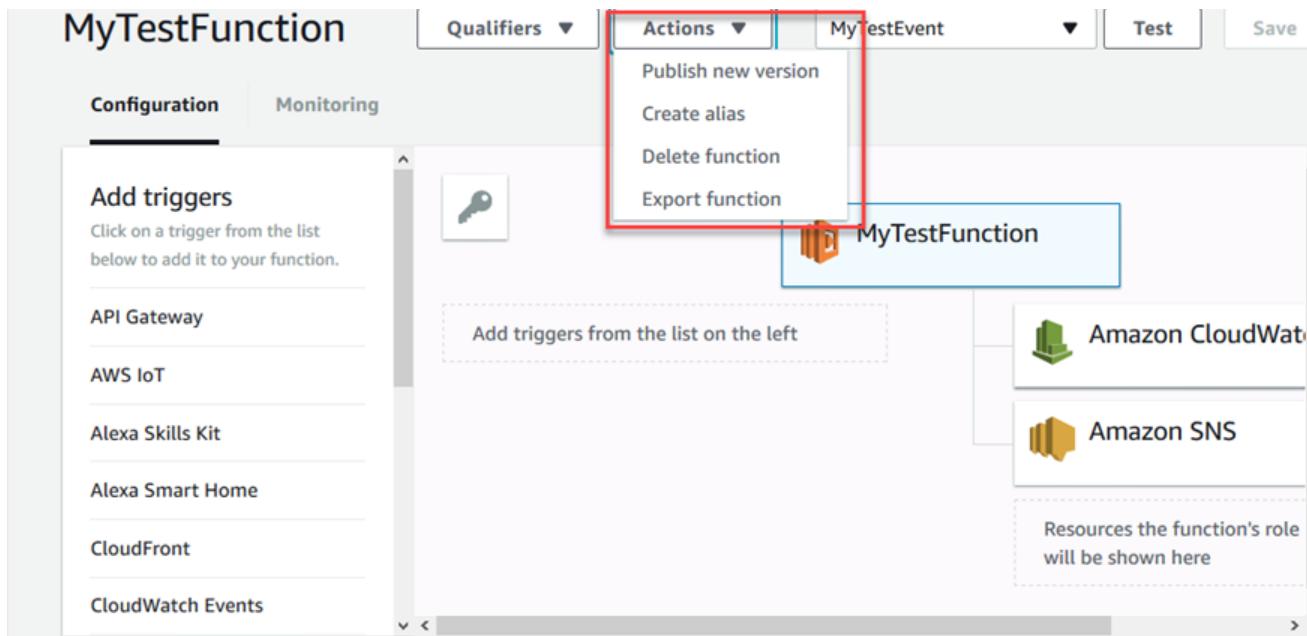
다음 스크린샷에서와 같이 확장된 [Qualifiers] 메뉴에는 [Versions]와 [Aliases] 탭이 표시됩니다. [Versions] 창에서 선택한 함수의 버전 목록을 확인할 수 있습니다. 선택한 함수에 대한 버전을 이전에 게시하지 않은 경우에는 다음과 같이 [Versions] 창에 \$LATEST 버전만 나열됩니다.



[Aliases] 탭을 선택하여 해당 함수의 별칭 목록을 확인합니다. 처음에는 다음 그림에서와 같이 어떤 별칭도 표시되지 않습니다.



이제 작업 메뉴를 사용하여 선택한 Lambda 함수의 버전을 게시하거나 별칭을 생성할 수 있습니다.



버전 관리 및 별칭에 대한 자세한 내용은 [AWS Lambda 함수 버전 관리 및 별칭 \(p. 45\)](#)을 참조하십시오.

## 별칭을 사용한 트래픽 이동

기본적으로 별칭은 단일 Lambda 함수 버전을 가리킵니다. 다른 함수 버전을 가리키도록 별칭을 업데이트하면 수신되는 요청 트래픽이 즉시 업데이트된 버전을 가리킵니다. 따라서 별칭이 새 함수로 인해 야기되는 모든 잠재적 불안정성에 노출됩니다. 이로 인한 영향을 최소화하기 위해 Lambda 별칭의 `routing-config` 파라미터를 구현하여 Lambda 함수의 다른 두 버전을 가리키고 각 버전으로 전송할 수신 트래픽의 비율을 지정할 수 있습니다.

예를 들어 수신 트래픽의 2%만 새 버전으로 라우팅되도록 하여 새 버전의 프로덕션 준비성을 분석하고 나머지 98%는 원래 버전으로 라우팅되도록 지정할 수 있습니다. 새 버전이 더 안정화되면 완전히 확신이 들 때까지 필요에 따라 비율을 점진적으로 업데이트할 수 있습니다. 그런 다음 별칭을 업데이트하여 모든 트래픽을 새 버전으로 라우팅할 수 있습니다.

별칭이 최대 두 개의 Lambda 함수 버전을 가리키도록 설정할 수 있습니다. 또한 다음과 같습니다.

- 두 버전이 동일한 IAM 실행 역할을 가져야 합니다.
- 두 버전이 동일한 AWS Lambda 함수 배달 못한 편지 대기열 (p. 86) 구성을 가지거나 DLQ 구성이 없어야 합니다.
- 별칭이 둘 이상의 버전을 가리키도록 설정할 때 \$LATEST는 가리킬 수 없습니다.

## 별칭을 사용한 트래픽 이동(CLI)

CreateAlias (p. 339) 작업을 사용하여 가중치를 기반으로 두 함수 버전 간에 트래픽을 이동하도록 별칭을 구성하려면 routing-config 파라미터를 구성해야 합니다. 다음 예제에서는 별칭이 다른 두 함수 버전을 가리키도록 설정합니다. 버전 2는 호출 트래픽의 2%를 수신하고 나머지 98%는 버전 1이 수신합니다.

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

UpdateAlias (p. 440) 작업을 사용하여 새 버전(버전 2)으로 수신되는 트래픽의 비율을 업데이트할 수 있습니다. 예를 들어 다음과 같이 새 버전으로 수신되는 호출 트래픽을 5%로 올릴 수 있습니다.

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

모든 트래픽을 버전 2로 라우팅하려면 UpdateAlias 작업을 사용하여 버전 2를 가리키도록 function-version 속성을 변경합니다. 동일한 명령에서 라우팅 구성을 재설정합니다.

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

## 별칭을 사용한 트래픽 이동(콘솔)

아래에서 설명한 대로 Lambda 콘솔을 통해 별칭을 사용한 트래픽 이동을 구성할 수 있습니다.

1. Lambda 함수를 열고 이전에 게시한 버전이 두 개 이상 있는지 확인합니다. 그렇지 않은 경우 AWS Lambda 버전 관리 소개 (p. 47) 단원으로 이동하여 버전 관리에 대해 알아보고 첫 번째 함수 버전을 게시합니다.
2. [Actions] 메뉴에서 [Create alias]를 선택합니다.
3. Create a new alias(새 별칭 생성) 창에서 별칭이 가리킬 Lambda 함수의 이름\*, 설명(선택 사항) 및 버전\* 값을 지정합니다. 여기서는 버전 1을 사용합니다.
4. [Additional version]에서 다음을 지정합니다.
  - a. 두 번째 Lambda 함수 버전을 지정합니다.
  - b. 함수의 가중치 값을 입력합니다. 가중치는 별칭이 호출될 때 해당 버전에 할당되는 트래픽의 비율입니다. 첫 번째 버전에는 남은 가중치가 할당됩니다. 예를 들어 [Additional version]에 10%를 지정할 경우 첫 번째 버전에 자동으로 90%가 할당됩니다.
5. Create를 선택합니다.

## 호출된 버전 확인

별칭이 두 함수 버전 간에 트래픽을 이동하는 경우 호출된 Lambda 함수 버전이 어떤 것인지 두 가지 방법으로 확인할 수 있습니다.

- CloudWatch Logs – 매 함수 호출 시 Lambda에서 호출된 버전 ID가 포함된 START 로그 항목을 자동으로 CloudWatch Logs로 내보냅니다. 예를 들면 다음과 같습니다.

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

Lambda는 Executed Version 차원을 사용하여 실행된 버전을 기준으로 측정치 데이터를 필터링합니다. 별칭 호출에만 적용됩니다. 자세한 내용은 [AWS Lambda CloudWatch 차원 \(p. 225\)](#) 단원을 참조하십시오.

- 응답 페이로드(동기식 호출) – 동기식 함수 호출에 대한 응답에 호출된 함수 버전을 나타내는 x-amz-executed-version 헤더가 포함되어 있습니다.

## AWS Lambda 계층

Lambda 함수는 추가 코드와 콘텐츠를 계층의 형태로 가져오도록 구성할 수 있습니다. 하나의 계층은 라이브러리, [사용자 지정 런타임 \(p. 103\)](#) 또는 그 외 종속성을 포함하는 ZIP 아카이브입니다. 배포 패키지에 라이브러리를 포함시킬 필요 없이 계층을 통해 함수에서 라이브러리를 사용할 수 있습니다.

계층을 사용하면 배포 패키지를 작게 유지할 수 있어 개발이 더 수월합니다. 함수 코드를 사용하여 종속 항목들을 설치하고 패키징할 때 발생할 수 있는 오류를 방지할 수 있습니다. Node.js, Python 및 Ruby 함수의 경우에는 배포 패키지를 3 MB 아래에 유지하는 한, [Lambda 콘솔에서 함수 코드를 개발 \(p. 24\)](#)할 수 있습니다.

### Note

함수는 한 번에 최대 5개 계층을 사용할 수 있습니다. 함수 및 모든 계층의 압축되지 않은 총 크기는 압축 해제된 배포 패키지 크기 제한인 250 MB를 초과할 수 없습니다. 자세한 내용은 [AWS Lambda 한도 \(p. 7\)](#) 단원을 참조하십시오.

여러 계층을 생성하거나 혹은 AWS 및 다른 AWS 고객들이 게시한 계층들을 사용할 수 있습니다. 계층은 특정 AWS 계정, [AWS Organizations](#) 또는 모든 계정에 대한 계층 사용 권한을 부여하기 위한 [리소스 기반 정책 \(p. 66\)](#)을 지원합니다.

계층들은 함수 실행 환경에서 /opt 디렉터리로 추출됩니다. 각 런타임은 언어에 따라 /opt 아래의 다른 위치에 있는 라이브러리를 찾습니다. 추가 구성 없이도 함수 코드가 라이브러리에 액세스할 수 있도록 [계층을 구조화 \(p. 65\)](#)하십시오.

또한 AWS Serverless Application Model(AWS SAM)을 사용하면 여러 계층과 해당 함수의 계층 구성을 각각 관리할 수 있습니다. 자세한 지침은 AWS Serverless Application Model 개발자 안내서에서 [서비스 리소스 선언](#) 단원을 참조하십시오.

### 섹션

- [계층을 사용하기 위한 함수 구성 \(p. 62\)](#)
- [계층 관리 \(p. 63\)](#)
- [계층 내 라이브러리 종속 항목들을 포함 \(p. 65\)](#)
- [계층 권한 \(p. 66\)](#)

## 계층을 사용하기 위한 함수 구성

함수를 생성하는 도중 또는 그 이후에 함수의 구성에서 최대 5개 계층을 지정할 수 있습니다. 사용할 특정 버전의 계층을 선택합니다. 나중에 다른 버전을 사용하려면 함수의 구성을 업데이트하십시오.

함수에 계층을 추가하려면 `update-function-configuration` 명령을 사용하십시오. 다음 예제에서는 2개의 계층 즉, 함수와 동일한 계정의 계층 그리고 다른 계정의 계층을 하나씩 추가합니다.

```
$ aws lambda update-function-configuration --function-name my-function \
--layers arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3
 \
arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2
{
    "FunctionName": "test-layers",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs8.10",
    "Role": "arn:aws:iam::123456789012:role/service-role/lambda-role",
    "Handler": "index.handler",
    "CodeSize": 402,
    "Description": "",
    "Timeout": 5,
    "MemorySize": 128,
    "LastModified": "2018-11-14T22:47:04.542+0000",
    "CodeSha256": "kDHALEY62Ni3OovMwVO8tNvgbRoRa6IOOKqShm7bSWF4=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "81cc64f5-5772-449a-b63e-12330476bcc4",
    "Layers": [
        {
            "Arn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3",
            "CodeSize": 169
        },
        {
            "Arn": "arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2",
            "CodeSize": 169
        }
    ]
}
```

계층 버전의 전체 ARN을 제공하여 사용할 각 계층의 버전을 지정해야 합니다. 계층이 이미 있는 함수에 계층을 추가하면 이전 목록은 새 목록이 덮어씁니다. 계층 구성을 업데이트 할 때마다 모든 계층을 포함하십시오. 모든 계층을 제거하려면 빈 목록을 지정합니다.

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

함수는 `/opt` 디렉터리에서 실행되는 동안 해당 계층의 콘텐츠에 액세스할 수 있습니다. 계층은 지정된 순서대로 적용되며, 동일한 이름의 폴더가 병합됩니다. 동일한 파일이 여러 계층에 나타나면 마지막으로 적용된 계층의 버전이 사용됩니다.

한 계층의 생성자는 사용 중인 해당 계층의 버전을 삭제할 수 있습니다. 이 경우, 마치 해당 계층 버전이 여전히 존재하는 것처럼 함수가 계속 실행됩니다. 다만 계층 구성을 업데이트 할 때에는 삭제된 버전에 대한 참조를 제거해야 합니다.

## 계층 관리

하나의 계층을 생성하려면 이름, 설명, ZIP 아카이브 그리고 해당 계층과 호환되는 [런타임 \(p. 99\)](#) 목록과 함께 `publish-layer-version` 명령을 사용하십시오. 런타임 목록은 선택 사항이며 다만 이 목록을 사용하면 계층을 보다 쉽게 찾을 수 있습니다.

```
$ 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"
    ]
}
}
```

`publish-layer-version`을 호출할 때마다 새 버전을 생성합니다. 계층을 사용하는 함수들은 계층 버전을 직접 참조합니다. 기존의 계층 버전에서는 [권한을 구성 \(p. 66\)](#)할 수 있으며 다만 다른 변경 사항을 적용하려면 새 버전을 만들어야 합니다.

함수의 런타임과 호환되는 계층을 찾으려면 `list-layers` 명령을 사용하십시오.

```
$ aws lambda list-layers --compatible-runtime python3.7
{
    "Layers": [
        {
            "LayerName": "my-layer",
            "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
            "LatestMatchingVersion": {
                "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
                "Version": 2,
                "Description": "My layer",
                "CreatedDate": "2018-11-15T00:37:46.592+0000",
                "CompatibleRuntimes": [
                    "python3.6",
                    "python3.7"
                ]
            }
        }
    ]
}
```

런타임 옵션을 생략하면 모든 계층을 나열할 수 있습니다. 응답의 세부 정보는 최신 버전의 계층을 반영합니다. `list-layer-versions`를 사용하여 계층의 모든 버전을 확인합니다. 버전에 관한 자세한 내용을 확인하려면 `get-layer-version` 단원을 참조하십시오.

```
$ 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",
    "Version": 2
}
```

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

응답 내 링크를 사용하면 계층 아카이브를 다운로드할 수 있으며 이 링크는 10분 동안 유효합니다. 계층 버전을 삭제하려면 `delete-layer-version` 명령을 사용합니다.

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

계층 버전을 삭제하면 이를 사용하기 위한 함수를 더 이상 구성할 수 없습니다. 그러나 해당 버전을 이미 사용 중인 모든 함수는 이 버전에 계속 액세스할 수 있습니다. 버전 번호는 계층 이름에 절대로 재사용되지 않습니다.

## 계층 내 라이브러리 종속 항목들을 포함

런타임 종속 항목들은 하나의 계층에 배치하여 함수 코드 밖으로 옮길 수 있습니다. 여러 계층에 포함된 라이브러리에 대해 함수 코드가 액세스할 수 있도록 /opt 디렉터리의 경로들이 Lambda 런타임에 포함됩니다.

하나의 계층에 여러 라이브러리를 포함시키려면 런타임에서 지원하는 폴더 중 하나에 라이브러리를 배치하십시오.

- Node.js – nodejs/node\_modules, nodejs/node8/node\_modules(NODE\_PATH)

### Example Node.js용 AWS X-Ray SDK

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

- Python – python, python/lib/python3.7/site-packages(사이트 디렉터리)

### Example Pillow

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

- Java – java/lib(classpath)

### Example Jackson

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

- Ruby – ruby/gems/2.5.0(GEM\_PATH), ruby/lib(RUBY\_LIB)

### Example JSON

```
json.zip
# ruby/gems/2.5.0/
| build_info
| cache
| doc
```

```
| extensions
| gems
| # json-2.1.0
# specifications
# json-2.1.0.gemspec
```

- 모두 -bin(PATH), lib(LD\_LIBRARY\_PATH)

#### Example JQ

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

Lambda 실행 환경의 경로 설정에 관한 자세한 내용은 [Lambda 함수에서 사용할 수 있는 환경 변수 \(p. 100\)](#) 단원을 참조하십시오.

## 계층 권한

계층 사용 권한은 리소스에서 관리됩니다. 하나의 계층이 있는 함수를 구성하려면 해당 계층 버전에서 `GetLayerVersion`을 호출할 수 있는 권한이 필요합니다. 계정 내 함수에 대해서는 [사용자 정책 \(p. 13\)](#) 또는 함수의 [리소스 기반 정책 \(p. 10\)](#)에서 이러한 권한을 얻을 수 있습니다. 다른 계정에서 계층을 사용하려면 사용자 정책에 대한 권한이 필요하며, 다른 계정의 소유자는 리소스 기반 정책으로 계정 권한을 부여해야 합니다.

다른 계정에 계층 사용 권한을 부여하려면 `add-layer-version-permission` 명령과 함께 해당 계층 버전의 권한 정책에 명령문을 추가하십시오. 각 명령문에서 단일 계정, 모든 계정 또는 조직을 대상으로 권한을 부여할 수 있습니다.

```
$ 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-5fc0d0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal":"
{"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:u
east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

권한은 단일 버전의 계층에만 적용됩니다. 새 계층 버전을 만들 때마다 해당 절차를 반복합니다.

더 많은 예제는 [계층에 다른 계정에 대한 액세스 권한 부여 \(p. 12\)](#) 단원을 참조하십시오.

## Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성

해당 계정의 가상 클라우드(VPC)에 연결하도록 함수를 구성할 수 있습니다. Amazon Virtual Private Cloud(Amazon VPC)를 사용하여 데이터베이스, 캐시 인스턴스, 내부 서비스 등과 같은 리소스에 대해 프라이빗 네트워크를 생성하십시오. 함수를 VPC에 연결하여 실행 중 프라이빗에 리소스에 액세스합니다.

AWS Lambda는 기본적으로 VPC 내에서 함수 코드를 안전하게 실행합니다. 하지만 Lambda 함수를 프라이빗 VPC 내부의 리소스에 액세스할 수 있게 하려면 AWS Lambda VPC 서브넷 ID 및 보안 그룹 ID가 포함된 추가 VPC 관련 구성 정보를 제공해야 합니다. 이 정보를 사용하여 함수가 프라이빗 VPC 내의 다른 리소스에 안전하게 연결할 수 있게 하는 탄력적 네트워크 인터페이스(ENI)를 설정합니다.

Lambda 함수는 [전용 인스턴스 테넌시](#)를 사용하여 VPC에 직접 연결할 수 없습니다. 전용 VPC의 리소스에 연결하려면, [기본 테넌시를 사용하여 두 번째 VPC에 피어로 연결합니다](#).

## Amazon VPC 액세스를 위한 Lambda 함수 구성

Lambda 함수를 만들 때 `VpcConfig` 파라미터를 사용하여 Lambda 함수 구성에 VPC 정보를 추가합니다([CreateFunction \(p. 347\)](#) 참조). 또는 기존 Lambda 함수 구성에 추가할 수 있습니다([UpdateFunctionConfiguration \(p. 455\)](#) 참조). 다음은 예제입니다.

- `create-function` CLI 명령은 Lambda 함수를 만들 때 VPC 정보를 제공하기 위해 `--vpc-config` 파라미터를 지정합니다.

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

### Note

Lambda 함수 실행 역할에는 ENI를 생성, 설명 및 삭제할 수 있는 권한이 있어야 합니다. AWS Lambda는 역할을 만들 때 사용할 수 있는 필수 EC2 작업(ec2:CreateNetworkInterface, ec2:DescribeNetworkInterfaces 및 ec2:DeleteNetworkInterface)에 대한 권한이 있는 AWSLambdaVPCAccessExecutionRole 권한 정책을 제공합니다. IAM 콘솔에서 정책을 검토할 수 있습니다. 함수 실행 직후에 이 역할을 삭제하지 마십시오. 함수가 실행될 때와 ENI 삭제 사이에 지연이 있습니다. 함수를 실행한 직후에 역할을 삭제하면 ENI를 삭제해야 합니다.

- `update-function-configuration` CLI 명령은 기존 Lambda 함수 구성에 VPC 정보를 추가하기 위해 `--vpc-config` 파라미터를 지정합니다.

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

Lambda 함수 구성에서 VPC 관련 정보를 제거하려면 다음 예제 CLI 명령에 표시된 대로 서브넷 ID 및 보안 그룹 ID의 빈 목록을 제공하여 `UpdateFunctionConfiguration` API를 사용합니다.

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

다음과 같은 추가 고려 사항에 유의합니다.

- Lambda 함수에 VPC 구성을 추가하면 해당 VPC의 리소스에만 액세스할 수 있습니다. 함수가 VPC 리소스와 퍼블릭 인터넷 모두에 액세스해야 하는 경우 VPC는 해당 VPC 내부에 NAT(Network Address Translation) 인스턴스가 있어야 합니다.
- Lambda 함수가 VPC 내에서 실행되도록 구성되면 추가 ENI 시작 페널티가 발생합니다. 즉, 네트워크 리소스에 연결하려고 하면 주소 확인이 지연될 수 있습니다.

## Lambda 함수에 대한 인터넷 액세스

AWS Lambda는 Lambda 함수가 VPC 리소스에 액세스할 수 있게 해주는 ENI를 설정하기 위해 사용자가 제공한 VPC 정보를 사용합니다. 각 ENI에는 지정한 서브넷 내의 IP 주소 범위에서 프라이빗 IP 주소가 할당되지만 퍼블릭 IP 주소는 할당되지 않습니다. 따라서 Lambda 함수에 인터넷 액세스가 필요한 경우(예: NAT 엔

드포인트가 없는 AWS 서비스에 액세스하는 경우) VPC 내부에 NAT 인스턴스를 구성하거나 Amazon VPC NAT 게이트웨이를 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서에서 [NAT 게이트웨이](#)를 참조하십시오. VPC에 연결된 인터넷 게이트웨이는 퍼블릭 IP 주소를 필요로 하기 때문에 사용할 수 없습니다.

Lambda 함수에 인터넷 액세스가 필요한 경우 인터넷 액세스 없이 퍼블릭 서브넷이나 프라이빗 서브넷에 연결하지 마십시오. 대신 NAT 인스턴스 또는 Amazon VPC NAT 게이트웨이를 통해 인터넷 액세스가 가능한 프라이빗 서브넷에만 연결합니다.

## VPC 활성화 Lambda 함수 설정에 대한 지침

Lambda 함수는 처리하는 이벤트 수에 따라 자동으로 조정됩니다. 다음은 조정 동작을 지원하기 위해 VPC 활성화 함수를 설정하기 위한 일반 지침입니다.

- Lambda 함수가 VPC에 액세스하는 경우 VPC에 Lambda 함수의 확장 요구 사항을 지원할 수 있는 충분한 ENI 용량이 있는지 확인해야 합니다. 다음 공식을 사용하여 ENI 필요 용량을 대략적으로 확인할 수 있습니다.

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

여기서 각 항목은 다음과 같습니다.

- 예상되는 최대 동시 실행 계획 – [동시성 관리 \(p. 37\)](#)의 정보를 사용하여 이 값을 결정합니다.
- 메모리 – Lambda 함수에 대해 구성된 메모리의 양입니다.
- 지정한 서브넷에는 ENI 수와 일치하는 충분한 IP 주소가 있어야 합니다.

또한 Lambda 함수 구성에서 각 가용 영역에 하나 이상의 서브넷을 지정하는 것이 좋습니다. 각 가용 영역에서 서브넷을 지정하면 중단되거나 IP 주소가 부족한 경우 함수가 다른 가용 영역에서 실행될 수 있습니다.

VPC에 충분한 ENI 또는 서브넷 IP가 없는 경우, 요청이 늘어나도 Lambda 함수가 확장되지 않으며 `EC2ThrottledException` 등 EC2 오류 유형의 호출 오류가 증가합니다. 비동기식 호출의 경우 해당하는 CloudWatch Logs 없이 오류가 증가하면 오류 응답을 얻기 위해 콘솔에서 Lambda 함수를 동기식으로 호출하십시오.

## 자습서: Amazon VPC에서 Amazon ElastiCache에 액세스하도록 Lambda 함수 구성

이 자습서에서는 다음 작업을 수행합니다.

- 기본 Amazon Virtual Private Cloud에 Amazon ElastiCache 클러스터를 생성합니다. Amazon ElastiCache에 대한 자세한 내용은 [Amazon ElastiCache](#)를 참조하십시오.
- ElastiCache 클러스터에 액세스하기 위해 Lambda 함수를 생성합니다. Lambda 함수를 생성할 때 Amazon VPC 및 VPC 보안 그룹에 서브넷 ID를 제공하여 Lambda 함수가 VPC의 리소스에 액세스할 수 있도록 합니다. 이 자습서의 설명에서는 함수가 UUID를 생성하여 이를 캐시에 기록한 다음, 캐시에서 이를 검색합니다.
- Lambda 함수를 호출하고 VPC의 ElastiCache 클러스터에 액세스했는지 확인합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$)와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 개체 – Lambda.
  - 권한 – AWSLambdaVPCAccessExecutionRole.
  - 역할 이름 – **lambda-vpc-role**.

AWSLambdaVPCAccessExecutionRole은 함수가 VPC에 대한 연결을 관리하는 데 필요한 권한을 가집니다.

## ElastiCache 클러스터 생성

기본 VPC에 ElastiCache 클러스터를 생성합니다.

1. 다음 AWS CLI 명령을 실행하여 Memcached 클러스터를 생성합니다.

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

[Security Groups] 아래 VPC 콘솔에서 기본 VPC 보안 그룹을 조회할 수 있습니다. 예제 함수는 이 클러스터에서 항목을 추가하고 검색합니다.

2. 실행한 캐시 클러스터의 구성 엔드포인트를 적어둡니다. 이 정보는 Amazon ElastiCache 콘솔에서 얻을 수 있습니다. 다음 단원에서는 함수 코드에서 이 값을 지정해 보겠습니다.

## 배포 패키지 만들기

다음의 예제 Python 코드는 항목을 읽어서 ElastiCache 클러스터에 기록합니다.

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 = uuid.uuid4().hex
    #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.decode("utf-8") == 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: " + uuid_obtained.decode("utf-8")
```

## 종속성

- [pymemcache](#) - Lambda 함수 코드가 이 라이브러리를 사용하여 memcache에서 항목을 설정 및 가져올 수 있도록 HashClient 객체를 생성합니다.
- [elasticsearch-auto-discovery](#) - Lambda 함수가 이 라이브러리를 사용하여 Amazon ElastiCache 클러스터에서 노드를 가져옵니다.

Pip를 사용하여 종속성을 설치하고 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지 \(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Lambda 함수 생성

`create-function` 명령을 사용하여 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name AccessMemCache --timeout 30 --memory-size 1024 \
\--zip-file fileb://function.zip --handler app.handler --runtime python3.7 \
--role execution-role-arn \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=default-security-group-id
```

VPC 콘솔에서 VPC의 서브넷 ID와 기본 보안 그룹 ID를 찾을 수 있습니다.

## Lambda 함수 테스트

이 단계에서는 `invoke` 명령을 사용하여 Lambda 함수를 수동으로 호출합니다. 실행 시 Lambda 함수는 UUID를 생성하고 Lambda 코드에 지정했던 ElastiCache 클러스터에 이를 기록합니다. 그런 다음 함수는 캐시에서 해당 항목을 검색합니다.

1. `invoke` 명령을 사용하여 Lambda 함수를 호출합니다.

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

2. 다음과 같이 Lambda 함수가 성공적으로 실행되었는지 확인합니다.

- output.txt 파일을 검토합니다.
- AWS Lambda 콘솔에서 결과를 검토합니다.
- CloudWatch Logs에서 결과를 확인합니다.

VPC에서 ElastiCache 클러스터를 액세스하는 Lambda 함수를 생성하고 나면, 이제 이벤트에 대한 응답으로 함수를 호출할 수 있습니다. 이벤트 소스 구성에 대한 자세한 내용과 예제를 보려면 [다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#) 단원을 참조하십시오.

## 자습서: Amazon VPC에서 Amazon RDS에 액세스하도록 Lambda 함수 구성

이 자습서에서는 다음 작업을 수행합니다.

- 기본 Amazon VPC에서 Amazon RDS MySQL 데이터베이스 엔진 인스턴스를 시작합니다. MySQL 인스턴스에서 샘플 테이블(Employee)이 포함된 데이터베이스(ExampleDB)를 생성합니다. Amazon RDS에 대한 자세한 내용은 [Amazon RDS](#)를 참조하십시오.
- ExampleDB 데이터베이스를 액세스하기 위한 Lambda 함수를 생성하고 테이블(Employee)을 생성한 다음, 몇 가지 레코드를 추가하고 테이블에서 레코드를 검색합니다.
- Lambda 함수를 호출하고 쿼리 결과를 확인합니다. 함수가 VPC에서 RDS MySQL 인스턴스에 액세스할 수 있는지 확인하는 방법입니다.

### 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

### 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 개체 – Lambda.
  - 권한 – AWSLambdaVPCAccessExecutionRole.
  - 역할 이름 – **lambda-vpc-role**.

AWSLambdaVPCAccessExecutionRole은 함수가 VPC에 대한 연결을 관리하는 데 필요한 권한을 가집니다.

## Amazon RDS 데이터베이스 인스턴스 생성

이 자습서에서 예제로 제시되는 Lambda 함수는 테이블(Employee)을 생성하고 몇 가지 레코드를 삽입한 다음, 레코드를 검색합니다. 함수가 생성하는 테이블의 스키마는 다음과 같습니다.

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

여기에서 EmpID는 기본 키입니다. 이제, 이 테이블에 몇 가지 레코드를 추가해야 합니다.

먼저 ExampleDB 데이터베이스의 기본 VPC에서 RDS MySQL 인스턴스를 시작합니다. 기본 VPC에서 RDS MySQL 인스턴스가 이미 실행 중인 경우에는 이 단계를 건너뛰십시오.

다음 방법 중 하나를 사용하여 RDS MySQL 인스턴스를 시작할 수 있습니다.

- Amazon RDS 사용 설명서의 [MySQL DB 인스턴스를 만들고 MySQL DB 인스턴스의 데이터베이스에 연결](#)에 나오는 지침을 따르십시오.
- 다음 AWS CLI 명령을 사용합니다.

```
$ 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
```

데이터베이스 이름, 사용자 이름 및 암호를 적어두십시오. DB 인스턴스의 호스트 주소(엔드포인트)도 필요합니다. 이 주소는 RDS 콘솔에서 확인할 수 있습니다. 인스턴스 상태가 사용 가능으로 되고 엔드포인트 값이 콘솔에 나타날 때까지 기다려야 할 수 있습니다.

## 배포 패키지 만들기

다음에 나오는 Python 코드 예제는 VPC에 생성한 MySQL RDS 인스턴스의 Employee 테이블을 토대로 SELECT 쿼리를 실행합니다. 코드는 ExampleDB 데이터베이스에서 테이블을 생성하고 샘플 레코드를 추가한 다음, 이러한 레코드를 검색합니다.

Example app.py

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

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

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

logger.info("SUCCESS: Connection to RDS MySQL instance succeeded")
```

```
def handler(event, context):
    """
    This function fetches content from MySQL RDS instance
    """

    item_count = 0

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

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

핸들러 외부에서 `pymysql.connect()`를 실행하면 함수가 데이터베이스 연결을 재사용하여 성능을 높일 수 있습니다.

두 번째 파일에는 함수에 대한 연결 정보가 들어 있습니다.

#### Example rds\_config.py

```
#config file containing credentials for RDS MySQL instance
db_username = "username"
db_password = "password"
db_name = "ExampleDB"
```

#### 종속성

- `pymysql` – Lambda 함수 코드는 이 라이브러리를 사용하여 MySQL 인스턴스에 액세스합니다([PyMySQL 참조](#)).

Pip를 사용하여 종속성을 설치하고 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지 \(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Lambda 함수 생성

`create-function` 명령을 사용하여 Lambda 함수를 만듭니다.

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

## Lambda 함수 테스트

이 단계에서는 `invoke` 명령을 사용하여 Lambda 함수를 수동으로 호출합니다. Lambda 함수는 실행 시 RDS MySQL 인스턴스의 Employee 테이블을 토대로 SELECT 쿼리를 실행하고 결과를 인쇄합니다. 이 결과는 CloudWatch Logs로도 전달됩니다.

1. `invoke` 명령을 사용하여 Lambda 함수를 호출합니다.

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

2. 다음과 같이 Lambda 함수가 성공적으로 실행되었는지 확인합니다.

- `output.txt` 파일을 검토합니다.
- AWS Lambda 콘솔에서 결과를 검토합니다.
- CloudWatch Logs에서 결과를 확인합니다.

VPC의 데이터베이스를 액세스하는 Lambda 함수를 생성하면, 이제 이벤트에 대한 응답으로 함수를 호출할 수 있습니다. 이벤트 소스 구성에 대한 자세한 내용과 예제를 보려면 [다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#) 단원을 참조하십시오.

## Lambda 함수 태그 지정

Lambda 함수는 여러 리전의 여러 애플리케이션으로 확장될 수 있습니다. 각 함수 호출의 빈도와 비용을 추적하는 프로세스를 단순화하기 위해 태그를 사용할 수 있습니다. 태그는 보다 효과적인 정리를 위해 AWS 리소스에 연결되는 키-값 페어입니다. 특히 의 경우에 동일한 유형의 리소스로 함수가 많은 경우에 유용합니다. 수백 개의 함수를 가진 고객은 태그를 사용하여 동일한 태그가 포함된 태그를 필터링함으로써 특정 세트에 쉽게 액세스하고 분석할 수 있습니다. 함수의 태그 지정에 대한 두 가지 주요 장점은 다음과 같습니다.

- **그룹화 및 필터링:** 태그를 적용하면 Lambda 콘솔 또는 CLI를 사용하여 특정 애플리케이션 또는 결제 부서에 포함된 Lambda 함수 목록을 분리할 수 있습니다. 자세한 내용은 [태그 지정된 Lambda 함수 필터링 \(p. 75\)](#) 단원을 참조하십시오.
- **비용 할당:** 태그 지정에 대한 Lambda의 지원이 AWS Billing과 통합되어 있기 때문에 결제를 동적 카테고리로 분류하고 함수를 특정 비용 센터에 매핑할 수 있습니다. 예를 들어 모든 Lambda 함수에 "Department"(부서) 키를 태그 지정하면 부서별로 모든 AWS Lambda 비용을 세분화 할 수 있습니다. 그런 다음 "Department 1" 또는 "Department 2"와 같은 개별 부서 값을 제공하여 함수 호출 비용을 해당 비용 센터로 지정할 수 있습니다. 비용 할당은 세부 결제 보고서를 통해 진행되므로 AWS 비용을 쉽게 분류하고 추적할 수 있습니다.

### 주제

- [결제용 Lambda 함수 태그 지정 \(p. 74\)](#)
- [콘솔을 사용하여 Lambda 함수에 태그 적용 \(p. 75\)](#)
- [CLI를 사용하여 Lambda 함수에 태그 적용 \(p. 75\)](#)
- [태그 지정된 Lambda 함수 필터링 \(p. 75\)](#)
- [태그 제한 \(p. 77\)](#)

## 결제용 Lambda 함수 태그 지정

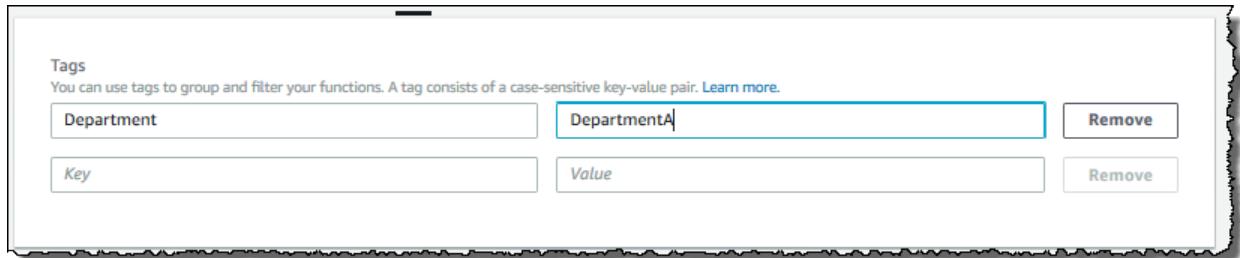
태그를 사용하여 비용 구조를 반영하도록 AWS 청구서를 구성할 수 있습니다. 이렇게 하려면 값이 비용 할당 보고서에 포함될 태그 키를 추가할 수 있습니다. 보고서에서 항목으로 포함되도록 선택한 태그 키가 포함된 비용 할당 보고서 설정에 대한 자세한 정보는 AWS 계정 결제 정보의 [월별 비용 할당 보고서 설정](#)을 참조하십시오.

결합된 리소스의 비용을 확인하려면 동일한 태그 키 값을 가진 함수에 따라 결제 정보를 구성할 수 있습니다. 예를 들어, 특정 애플리케이션 이름으로 여러 Lambda 함수에 태그를 지정한 다음 결제 정보를 구성하여 여러 서비스에 걸친 해당 애플리케이션의 총 비용을 볼 수 있습니다. 자세한 정보는 [AWS Billing and Cost Management 사용 설명서](#)의 비용 할당 태그 사용을 참조하십시오.

AWS Lambda에서 태그 지정할 수 있는 유일한 리소스는 함수입니다. 별칭 또는 특정 함수 버전은 태그 지정 할 수 없습니다. 함수의 별칭 또는 버전 호출은 원래 함수 호출로 청구됩니다.

## 콘솔을 사용하여 Lambda 함수에 태그 적용

[configuration] 탭의 [Tags] 섹션에서 태그를 함수에 추가할 수 있습니다.



기존 함수에서 태그를 제거하려면 함수를 열고 [Tags] 섹션을 선택한 다음 키-값 페어 옆의 [Remove] 버튼을 선택합니다.



## CLI를 사용하여 Lambda 함수에 태그 적용

Lambda 함수를 새로 생성할 때 --tags 옵션을 사용하여 태그를 포함시킬 수 있습니다.

```
$ aws lambda create-function --function-name my-function  
--handler index.js --runtime nodejs8.10 \  
--role role-arn \  
--tags "DEPARTMENT=Department A"
```

기존 함수에 태그를 추가하려면 tag-resource 명령을 사용하십시오.

```
$ aws lambda tag-resource \  
--resource function arn \  
--tags "DEPARTMENT=Department A"
```

태그를 제거하려면 untag-resource 명령을 사용합니다.

```
$ aws lambda untag-resource --resource function arn \  
--tagkeys DEPARTMENT
```

## 태그 지정된 Lambda 함수 필터링

태그를 사용하여 Lambda 함수를 그룹화하면 Lambda 콘솔 또는 AWS CLI에서 제공하는 필터링 기능을 활용하여 특정 요구 사항을 바탕으로 보기 기능을 사용할 수 있습니다.

## 콘솔을 이용하여 Lambda 함수 필터링

Lambda 콘솔에는 태그를 비롯하여 지정된 함수 속성 집합을 기반으로 함수 목록을 필터링할 수 있는 검색 필드가 있습니다. [Department]라는 [Tags] 키를 가진 [MyFunction] 및 [MyFunction2]라는 이름의 두 가지 함

수가 있다고 가정해 보겠습니다. 이러한 함수를 보려면 검색 필드를 선택하고 [Tags] 키 목록을 포함하는 자동 필터링을 확인합니다.

The screenshot shows the AWS Lambda 'Functions' list page with a search bar containing 'tag:Department'. The results table shows three functions:

	Runtime	Code size
mbda function.	Node.js 6.10	333 bytes
mbda function.	Node.js 6.10	333 bytes
mbda function.	Python 2.7	360 bytes

Department(부서) 키를 선택합니다. 그러면 Lambda에서는 해당 키가 포함된 함수를 반환합니다.

이제 [MyFunction] 태그의 키 값이 "Department A"이고 MyFunction2의 키 값이 "Department B"라고 가정합니다. 아래 그림과 같이 [Department] 키(이 경우 [Department A])의 값을 선택하여 검색 범위를 좁힐 수 있습니다.

The screenshot shows the AWS Lambda 'Functions' list page with a search bar containing 'tag:Department : DepartmentA'. The results table shows two functions:

	Runtime	Code size
mbda function.	Node.js 6.10	333 bytes
mbda function.	Node.js 6.10	333 bytes

이는 [MyFunction]만 반환합니다.

[Description], [Function name] 또는 [Runtime] 등 수락된 다른 함수 속성을 포함하여 검색 범위를 더욱 좁힐 수 있습니다.

#### Note

Lambda 함수당 최대 50개의 태그로 제한됩니다. 함수를 삭제하면 연결된 태그 역시 삭제됩니다.

## CLI를 이용하여 Lambda 함수 필터링

특정 Lambda 함수에 적용된 태그를 보려면 다음 Lambda API 명령 중 하나를 사용할 수 있습니다.

- [ListTags \(p. 413\)](#): Lambda 함수 Amazon 리소스 이름(ARN)을 제공하여 이 함수와 연결된 태그 목록을 봅니다.

```
$ aws lambda list-tags --resource function arn
```

- [GetFunction \(p. 374\)](#): Lambda 함수 이름을 제공하여 이 함수와 연결된 태그 목록을 봅니다.

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

또한 AWS 태그 지정 서비스의 [GetResources](#) API를 태그별로 리소스를 필터링할 수도 있습니다. GetResources API는 최대 10개의 필터를 수신하며 각 필터는 태그 키와 최대 10개의 태그 값을 포함합니다.

GetResources에 'ResourceType'을 지정하면 특정 리소스 유형별로 필터링할 수 있습니다. AWS 태그 지정 서비스에 대한 자세한 내용은 [리소스 그룹 작업](#) 단원을 참조하십시오.

## 태그 제한

태그에 적용되는 제한은 다음과 같습니다.

- 리소스당 최대 태그 수 - 50개
- 최대 키 길이 - UTF-8의 유니코드 문자 128자
- 최대 값 길이 - 유니코드 문자 256자(UTF-8)
- 태그 키와 값은 대/소문자를 구분합니다.
- 태그 이름이나 값에서 aws: 접두사는 사용하지 마십시오. 이 단어는 AWS용으로 예약되어 있습니다. 이 접두사가 지정된 태그 이름이나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스 당 태그 수 제한에 포함되지 않습니다.
- 태깅 스키마를 여러 서비스와 리소스에서 사용하게 될 경우 다른 서비스 또한 허용되는 문자에 대한 제한이 있을 수 있음을 유의하십시오. 일반적으로 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 공백 및 숫자 와 특수 문자+ - = . \_ : / @

# AWS Lambda 함수 호출

AWS Lambda에서 애플리케이션 빌드 시 코어 구성 요소는 Lambda 함수와 이벤트 소스입니다. 이벤트 소스는 이벤트를 게시하는 AWS 서비스 또는 사용자 지정 애플리케이션이고, Lambda 함수는 이벤트를 처리하는 사용자 지정 코드입니다. 예시를 위해 다음 시나리오를 고려해 보십시오.

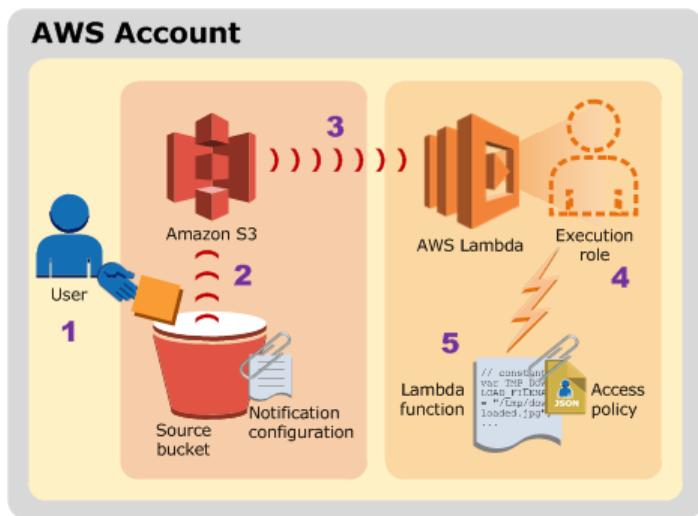
- 파일 처리 – 사진 공유 애플리케이션을 가지고 있다고 가정해봅시다. 사람들이 이 애플리케이션을 사용하여 사진을 업로드하면 애플리케이션은 이러한 사용자 사진을 버킷에 저장합니다. 그런 다음, 애플리케이션은 각 사용자 사진의 썸네일 버전을 만들어서 이를 사용자의 프로필 페이지에 표시합니다. 이 시나리오에서 자동으로 썸네일을 생성하는 Lambda 함수를 만드는 방법을 선택할 수 있습니다. Amazon S3은 객체 생성 이벤트를 게시하고 Lambda 함수를 호출할 수 있는 지원되는 AWS 이벤트 소스 중 하나입니다. 함수 코드는 S3 버킷에서 사진 객체를 읽어서 썸네일 버전을 만든 다음, 이를 또 다른 S3 버킷에 저장할 수 있습니다.
- 데이터 및 분석 – 분석 애플리케이션을 구축하여 DynamoDB 테이블의 원시 데이터를 저장하고 있다고 가정해봅시다. 업데이트를 기록하거나 테이블에서 항목을 삭제할 때 스트림은 테이블에 연결된 스트림에 항목 업데이트 이벤트를 게시할 수 있습니다. 이 경우, 이벤트 데이터는 항목 키, 이벤트 이름(삽입, 업데이트, 삭제) 및 기타 관련 세부 정보를 제공합니다. 원시 데이터를 집계하여 사용자 지정 측정치를 생성하도록 함수를 작성할 수 있습니다.
- 웹사이트 – 웹사이트를 생성하고 있으며 Lambda에 백엔드 로직을 호스팅하고 싶다고 가정해봅시다. HTTP 엔드포인트로 Amazon API Gateway를 사용하여 HTTP를 통해 Lambda 함수를 호출할 수 있습니다. 이제 웹 클라이언트가 API를 호출할 수 있으며, API 게이트웨이는 Lambda에 요청을 라우팅할 수 있습니다.
- 모바일 애플리케이션 – 이벤트를 만드는 사용자 지정 모바일 애플리케이션을 가지고 있다고 가정해봅시다. 함수를 생성하여 사용자 지정 애플리케이션이 게시한 이벤트를 처리할 수 있습니다. 예를 들어 이 시나리오에서는 사용자 지정 모바일 애플리케이션 내의 클릭을 처리하도록 함수를 구성할 수 있습니다.

AWS Lambda는 이벤트 소스로 다수의 AWS 서비스를 지원합니다. 자세한 내용은 [다른 서비스와 함께 AWS Lambda 사용 \(p. 128\)](#) 단원을 참조하십시오. Lambda 함수를 트리거하도록 이들 이벤트 소스를 구성하면 이벤트 발생 시 Lambda 함수가 자동으로 호출됩니다. 이벤트 소스 매핑을 정의하여 추적할 이벤트와 호출할 Lambda 함수를 식별하는 방법을 정합니다.

다음은 이벤트 소스와 전체 경험의 작동 방법을 소개하는 예제입니다.

## 예제 1: Amazon S3가 이벤트를 푸시하고 Lambda 함수를 호출

Amazon S3은 PUT, POST, COPY 및 DELETE 객체 이벤트 같이 상이한 유형의 이벤트를 버킷에 게시할 수 있습니다. 아래 그림에서와 같이 버킷 알림 기능을 사용하여 특정 유형의 이벤트가 발생할 때 Amazon S3에게 Lambda 함수를 호출하도록 지시하는 이벤트 소스 매핑을 구성할 수 있습니다.



아래 그림은 다음 시퀀스를 보여줍니다.

1. 사용자는 버킷에 객체를 생성합니다.
2. Amazon S3가 객체 생성 이벤트를 감지합니다.
3. Amazon S3는 [실행 역할 \(p. 9\)](#)에서 제공한 권한을 사용하여 Lambda 함수를 호출합니다.
4. AWS Lambda는 Lambda 함수를 실행하여 이벤트를 파라미터로 지정합니다.

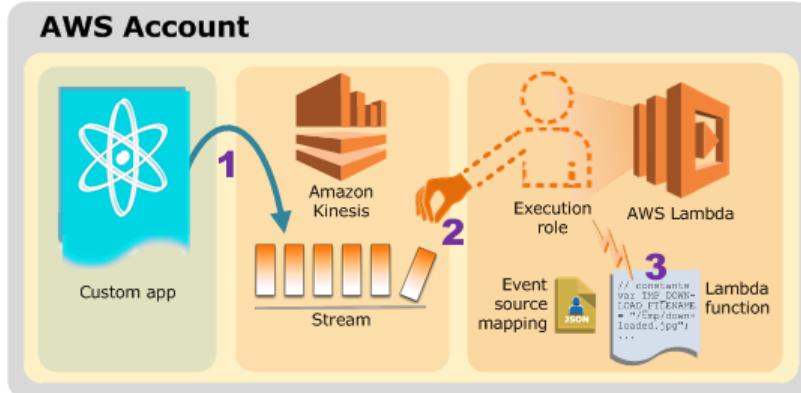
함수를 버킷 알림 작업으로 호출하도록 Amazon S3를 구성할 수 있습니다. 함수를 호출할 수 있는 권한을 Amazon S3에 부여하려면 함수의 [리소스 기반 정책 \(p. 10\)](#)을 업데이트하십시오.

## 예제 2: AWS Lambda가 Kinesis 스트림에서 이벤트를 가져와서 Lambda 함수 호출

풀 기반의 이벤트 소스에서 AWS Lambda는 소스를 폴링한 다음 레코드가 소스에서 감지될 때 Lambda 함수를 호출합니다.

- [CreateEventSourceMapping \(p. 343\)](#)
- [UpdateEventSourceMapping \(p. 444\)](#)

아래 그림은 사용자 지정 애플리케이션이 Kinesis 스트림에 레코드를 어떻게 기록하는지 보여줍니다.



아래 그림은 다음 시퀀스를 보여줍니다.

1. 사용자 지정 애플리케이션은 Kinesis 스트림에 레코드를 기록합니다.
2. AWS Lambda는 스트림을 지속적으로 폴링하고 서비스가 스트림에서 새로운 레코드를 감지할 때 Lambda 함수를 호출합니다. AWS Lambda는 Lambda에서 생성하는 이벤트 소스 매핑에 따라 폴링할 대기열과 호출할 Lambda 함수를 식별합니다.
3. Lambda 함수는 수신 이벤트에서 호출됩니다.

스트림 기반 이벤트 소스를 사용할 경우, AWS Lambda에서 이벤트 소스 매핑을 생성하십시오. Lambda는 함수를 동기적으로 호출하는 스트림에서 항목을 읽습니다. 함수를 호출할 수 있는 권한을 Lambda에 부여할 필요는 없지만, 스트림에서 읽는 권한은 필요합니다.

## 호출 유형

AWS Lambda는 Lambda 함수의 동기식 및 비동기식 호출을 지원합니다. 사용자가 Lambda 함수를 호출할 때만(일명 온디맨드 호출) 호출 유형을 제어할 수 있습니다. 다음 예제들은 요청 시 호출을 보여줍니다.

- 사용자 지정 애플리케이션은 Lambda 함수를 호출합니다.
- 테스트 목적으로 Lambda 함수를 수동으로 호출합니다(예를 들어 AWS CLI 사용).

두 경우에 모두 [Invoke \(p. 392\)](#) 작업을 사용하여 Lambda 함수를 호출합니다. 그리고 동기식 또는 비동기식으로 호출 유형을 지정할 수 있습니다.

AWS 서비스를 트리거로 사용하면 서비스 각각에 대해 호출 유형이 사전 결정됩니다. 사용자는 Lambda 함수를 호출할 때 이러한 이벤트 소스가 사용하는 호출 유형을 제어할 권리가 없습니다.

예를 들어 Amazon S3는 Lambda 함수를 비동기식으로 호출하고 Amazon Cognito는 항상 Lambda 함수를 동기식으로 호출합니다. 폴 기반 AWS 서비스(Amazon Kinesis, Amazon DynamoDB, Amazon Simple Queue Service)의 경우, AWS Lambda는 스트림 또는 메시지 대기열을 폴링하고 동기식으로 Lambda 함수를 호출합니다.

## AWS Lambda 이벤트 소스 매핑

Lambda 함수와 이벤트 소스는 AWS Lambda의 핵심 구성 요소입니다. 이벤트 소스는 이벤트를 게시하는 엔터티이고, Lambda 함수는 이벤트를 처리하는 사용자 지정 코드입니다. 지원되는 이벤트 소스는 AWS Lambda에서 사용할 수 있도록 사전 구성이 가능한 AWS 서비스입니다. 구성은 Lambda 함수에 이벤트 소스를 매핑한다는 점에서 이벤트 소스 매핑이라고도 하는데, 이벤트 발생 시 함수를 자동 호출하도록 할 수 있습니다.

각각의 이벤트 소스 매핑은 게시할 이벤트의 유형과 이벤트 발생 시 호출할 Lambda 함수를 식별합니다. 그러면 특정 Lambda 함수가 파라미터로 이벤트 정보를 수신하고, Lambda 함수 코드가 이벤트를 처리합니다.

또한 AWS 리소스 이벤트를 포함하고 Lambda 함수를 호출하도록 사용자 지정 애플리케이션을 생성할 수 있습니다. 자세한 내용은 [AWS Command Line Interface에서 AWS Lambda 사용 \(p. 87\)](#) 단원을 참조하십시오.

이벤트 매핑 정보를 어디에 저장해야 할지 모르시겠습니까? 이벤트 소스 내에 저장할까요, 아니면 내에 저장할까요? 다음 단원은 이러한 이벤트 소스 카테고리 각각에 대한 이벤트 소스 매핑을 설명합니다. 이들 단원에서는 Lambda 함수가 호출되는 방법과 Lambda 함수를 호출할 수 있도록 권한을 관리하는 방법도 설명합니다.

### 주제

- [AWS 서비스의 이벤트 소스 매핑 \(p. 81\)](#)

- AWS 폴 기반 서비스의 이벤트 소스 매팅 (p. 81)
- 사용자 지정 애플리케이션의 이벤트 소스 매팅 (p. 82)

## AWS 서비스의 이벤트 소스 매팅

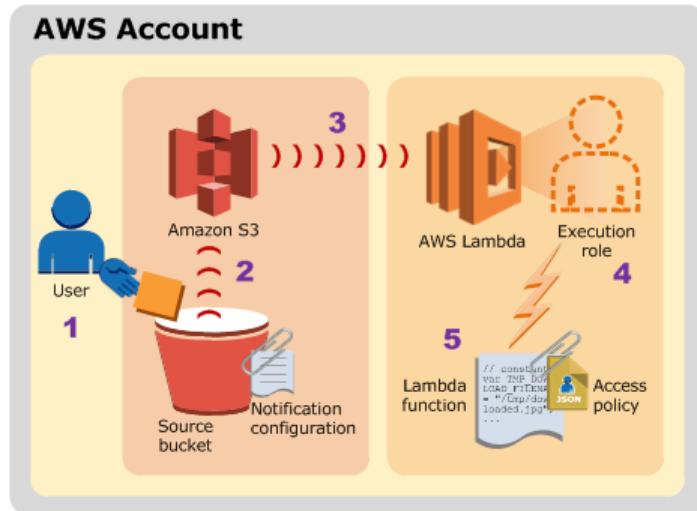
폴 기반의 AWS 서비스(Amazon Kinesis Data Streams 및 DynamoDB 스트림 또는 Amazon SQS 대기열)를 제외하고 다른 지원 AWS 서비스들은 이벤트를 게시하는 것은 물론이고, Lambda 함수를 호출할 수 있습니다(일명 푸시 모델). 푸시 모델에서는 다음 사항에 유의하십시오.

- 이벤트 소스 매팅은 이벤트 소스 내에 유지됩니다. 이벤트 소스에서 해당 API가 지원되기 때문에 이벤트 소스 매팅을 생성 및 관리할 수 있습니다. 예를 들어 Amazon S3는 버킷 알림 구성 API를 제공합니다. 이 API를 사용하여 게시할 버킷 이벤트와 호출할 함수를 식별하는 이벤트 소스 매팅을 구성할 수 있습니다.
- 이벤트 소스가 Lambda 함수를 호출하기 때문에 리소스 기반의 정책을 사용하여 이벤트 소스에 필요한 권한을 부여해야 합니다. 자세한 내용은 [AWS Lambda에서 리소스 기반 정책 사용 \(p. 10\)](#) 단원을 참조하십시오.

다음 예제는 이 모델의 작동 방식을 설명합니다.

Example – Amazon S3가 이벤트를 푸시하고 Lambda 함수를 호출

각각의 객체 생성 버킷 이벤트에 대해 AWS Lambda 함수를 호출하고 싶다고 가정해보십시오. 버킷 알림 구성에 필요한 이벤트 소스 매팅을 추가합니다.



다음 그림은 그 흐름을 보여 줍니다.

1. 사용자는 버킷에 객체를 생성합니다.
2. Amazon S3가 객체 생성 이벤트를 감지합니다.
3. Amazon S3는 버킷 알림 구성에 설명되어 있는 이벤트 소스 매팅에 따라 Lambda 함수를 호출합니다.
4. AWS Lambda는 Lambda 함수에 연결되는 권한 정책을 확인하여 Amazon S3가 필요한 권한을 갖도록 합니다. 권한 정책에 대한 자세한 내용은 [AWS Lambda 권한 \(p. 9\)](#)을 참조하십시오.
5. AWS Lambda는 일단 연결된 권한 정책을 확인하고 Lambda 함수를 실행합니다. 함수는 파라미터로 이벤트를 수신한다는 것을 기억하십시오.

## AWS 폴 기반 서비스의 이벤트 소스 매팅

AWS Lambda는 다음의 폴 기반 서비스를 지원합니다.

## Lambda가 이벤트를 읽는 서비스

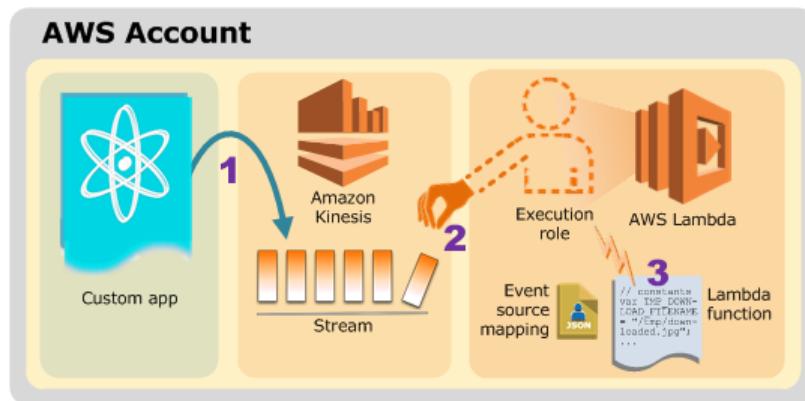
- [Amazon Kinesis \(p. 175\)](#)
- [Amazon DynamoDB \(p. 165\)](#)
- [Amazon Simple Queue Service \(p. 211\)](#)

이벤트 소스 매핑을 구성하면 AWS Lambda가 이벤트 소스를 폴링하고 Lambda 함수를 호출합니다. 이벤트 소스 매핑은 AWS Lambda에서 유지됩니다. AWS Lambda는 이벤트 소스 매핑을 생성 및 관리하기 위한 API를 제공합니다. 자세한 내용은 [CreateEventSourceMapping \(p. 343\)](#) 단원을 참조하십시오.

AWS Lambda에는 Kinesis 및 DynamoDB 스트림 또는 Amazon SQS 대기열을 폴링하고 레코드를 읽을 수 있는 권한이 필요합니다. Lambda 함수를 생성할 때 지정한 역할과 연결되는 권한 정책을 사용하여 [실행 역할 \(p. 9\)](#)을 통해 이러한 권한을 부여합니다. AWS Lambda는 Lambda 함수를 호출할 수 있는 권한이 필요하지 않습니다.

아래 그림은 Kinesis 스트림에 레코드를 기록하는 사용자 지정 애플리케이션과 AWS Lambda가 스트림을 폴링하는 방법을 보여줍니다. 스트림에서 새로운 레코드를 감지하면 AWS Lambda는 Lambda 함수를 호출합니다.

Kinesis 스트림에 레코드를 기록하는 사용자 지정 애플리케이션을 가지고 있다고 가정해보십시오. 스트림에서 새 레코드가 감지될 때 함수를 호출하고 싶을 것입니다. AWS Lambda에서 Lambda 함수와 필요한 이벤트 소스 매핑을 생성합니다.



아래 그림은 다음 시퀀스를 보여줍니다.

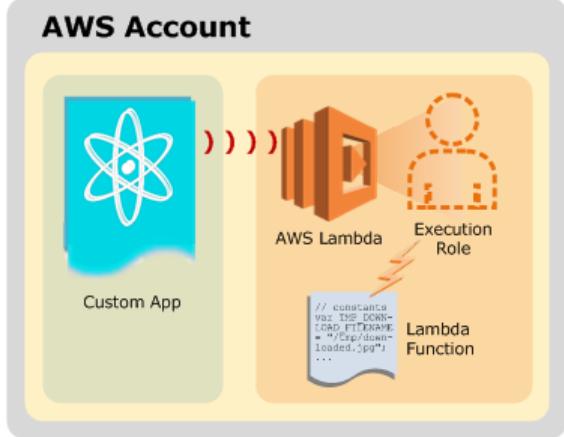
1. 사용자 지정 애플리케이션은 Amazon Kinesis 스트림에 레코드를 기록합니다.
2. AWS Lambda는 스트림을 지속적으로 폴링하고 서비스가 스트림에서 새로운 레코드를 감지하면 Lambda 함수를 호출합니다. AWS Lambda는 AWS Lambda에서 생성하는 이벤트 소스 매핑에 따라 폴링할 대기열과 호출할 Lambda 함수를 식별합니다.
3. AWS Lambda가 Lambda 함수를 실행합니다.

이 예제에서는 Kinesis 스트림을 사용하지만, DynamoDB 스트림을 사용하는 경우도 마찬가지입니다.

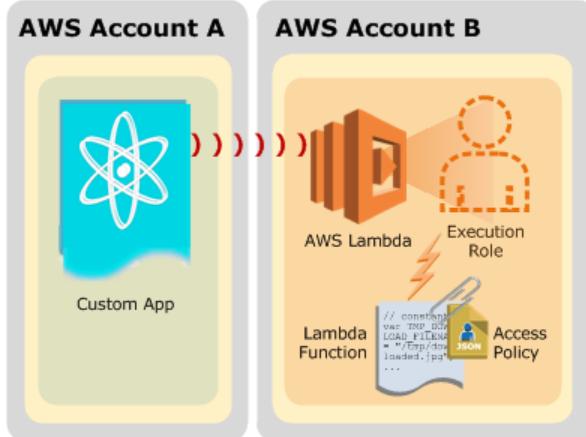
## 사용자 지정 애플리케이션의 이벤트 소스 매핑

이벤트를 게시 및 처리하는 사용자 지정 애플리케이션을 가지고 있는 경우에는 이러한 이벤트를 처리하도록 Lambda 함수를 생성할 수 있습니다. 이 경우에는 사전 구성이 필요하지 않습니다. 즉, 이벤트 소스 매핑을 설정할 필요가 없습니다. 대신에 이벤트 소스가 AWS Lambda Invoke API를 사용합니다. 애플리케이션 및 Lambda 함수를 다른 AWS 계정 여러 개가 소유하고 있는 경우에는 Lambda 함수를 소유하는 AWS 계정이 Lambda 함수에 연결된 권한 정책에서 교차 계정 권한을 허용해야 합니다.

아래 그림은 계정의 사용자 지정 애플리케이션이 어떻게 Lambda 함수를 호출하는지 보여줍니다. 이 예제에서는 사용자 지정 애플리케이션이 함수를 소유하고 있는 계정과 동일한 계정 자격 증명을 사용하고 있기 때문에 함수를 호출하기 위한 추가 권한이 필요하지 않습니다.



다음 예제에서는 사용자 애플리케이션과 Lambda 함수를 다른 AWS 계정 여러 개가 소유하고 있습니다. 이 경우에는 Lambda 함수를 소유하는 AWS 계정이 Lambda 함수에 연결된 권한 정책에서 교차 계정 권한을 허용해야 합니다. 자세한 내용은 [AWS Lambda 권한 \(p. 9\)](#) 단원을 참조하십시오.



## AWS Lambda 재시도 동작

함수 호출 시 몇 가지 이유로 인해 오류가 발생할 수 있습니다. 코드에서 예외가 발생하거나 시간이 초과되거나 메모리가 부족할 수 있습니다. 코드를 실행하는 런타임에서 오류 및 중지가 발생할 수 있습니다. 동시 연결이 부족하여 조절(throttling)이 수행될 수 있습니다.

오류가 발생할 경우 코드가 모두 실행될 수도 있고 부분적으로 실행될 수도 있고 전혀 실행되지 않을 수도 있습니다. 대부분의 경우, 함수를 호출하는 클라이언트나 서비스는 오류가 발생할 경우 재시도하므로 코드는 원치 않는 결과를 가져오지 않도록 동일한 이벤트를 반복 처리할 수 있어야 합니다. 함수에서 리소스를 관리하거나 데이터베이스를 작성할 경우 동일한 요청이 여러 번 처리되는 경우를 처리할 필요가 있습니다.

Lambda는 호출의 출처에 따라 다음 방식으로 재시도를 처리합니다.

- 스트림 기반이 아닌 이벤트 소스 - 이러한 이벤트 소스 중 일부는 Lambda 함수를 동기식으로 호출하도록 설정이 되고, 일부는 비동기식으로 함수를 호출합니다. 따라서 예외는 다음과 같이 처리됩니다.
    - 동기식 호출 - Lambda는 응답 본문에 `FunctionError` 필드를 포함시키고 `x-Amz-Function-Error` 헤더에 오류에 대한 세부 정보를 담습니다. 상태 코드는 함수 오류를 의미하는 200입니다. Lambda는 요

청, 함수 또는 권한에 문제가 있어 핸들러가 이벤트를 처리할 수 없는 경우에만 오류 상태 코드를 반환합니다. 자세한 내용은 [호출 오류 \(p. 394\)](#)를 참조하십시오.

서비스에 따라 [AWS 서비스 트리거 \(p. 128\)](#)를 재시도할 수 있습니다. 애플리케이션에서 직접 Lambda 함수를 호출하는 경우, 재시도 여부를 선택할 수 있습니다.

- 비동기식 호출 – 비동기식 이벤트는 Lambda 함수 호출에 사용되기 전에 대기열로 보내집니다. 각 이벤트를 완벽하게 처리할 수 없는 경우에는 재시도 간에 지연 시간을 두고 자동으로 두 번 호출을 재시도합니다. 함수에 대해 [배달 못한 편지 대기열 \(p. 86\)](#)을 구성하여 3회의 시도가 모두 실패한 요청을 캡처합니다.
- 스트림 기반의 폴 기반 이벤트 소스 - Kinesis Data Streams 또는 DynamoDB로 구성됩니다. Lambda 함수 호출에 실패하면 AWS Lambda는 데이터가 만료될 때까지(최대 7일) 실패한 레코드 배치를 처리하려고 시도합니다.

예외는 차단으로 처리되며, 실패한 레코드 배치가 만료되거나 성공적으로 처리될 때까지 AWS Lambda는 샤프에서 새로운 레코드를 읽어 들이지 않습니다. 이렇게 하면 AWS Lambda에서 순서대로 스트림 이벤트가 처리됩니다.

- 스트림을 기반으로 하지 않는 폴 기반 이벤트 소스 - Amazon Simple Queue Service로 구성됩니다. Amazon SQS 대기열을 이벤트 소스로 구성한 경우, AWS Lambda가 대기열에서 레코드 배치를 폴링하고 Lambda 함수를 호출합니다. 호출이 실패하거나 시간이 초과되면 배치의 모든 메시지가 대기열로 반환되며, [제한 시간 초과](#) 기간이 만료되면 각각 처리될 수 있습니다. (제한 시간 초과는 Amazon Simple Queue Service에서 다른 소비자가 메시지를 수신하고 처리하는 것을 방지하는 기간입니다.)

호출이 배치를 성공적으로 처리하면 해당 배치의 각 메시지가 대기열에서 제거됩니다. 성공적으로 처리되지 않은 메시지는 삭제됩니다. 또는 [Amazon SQS 배달 못한 편지 대기열](#)을 구성한 경우 분석을 위해 여기로 실패 정보를 보냅니다.

이벤트를 순차적으로 처리할 필요가 없는 경우 Amazon SQS 대기열을 사용하면 이전 메시지 호출의 실패 여부에 상관없이 AWS Lambda가 새 메시지를 계속 처리한다는 장점이 있습니다. 즉, 새 메시지의 처리가 차단되지 않습니다.

호출 모드에 대한 자세한 내용은 [AWS Lambda 이벤트 소스 매핑 \(p. 80\)](#) 단원을 참조하십시오.

## 조정 동작 이해

동시 실행이란 특정 시점에 이루어지는 함수 코드의 실행 횟수를 뜻합니다. 동시 실행 횟수는 예상이 가능하지만, Lambda 함수가 폴 기반의 이벤트 소스에서 이벤트를 처리하는지 여부에 따라 다릅니다.

폴 기반이 아닌 이벤트 소스(예: Lambda가 Amazon S3 또는 API 게이트웨이 같은 다른 소스의 모든 이벤트를 처리할 수 있음)의 이벤트를 처리하기 위해 Lambda 함수를 생성한 경우에는 게시된 각 이벤트가 작업 단위가 됩니다(계정 한도에 도달할 때까지 별도로 처리됨). 따라서 이러한 이벤트 소스의 수는 동시 실행에 영향을 줍니다. 이 공식을 사용하여 함수가 사용하는 용량을 예측할 수 있습니다.

invocations per second \* average execution duration in seconds

Amazon S3 이벤트를 처리하는 Lambda 함수를 예로 들어보겠습니다. Lambda 함수 호출에 평균 3초가 걸리고 Amazon S3가 초당 10개의 이벤트를 게시한다고 가정해보고, 그 다음 함수를 30회 동시 실행해보겠습니다.

폴 기반 이벤트 소스의 동시 실행 수는 다음과 같은 추가 요인에 따라 달라집니다.

- 스트림 기반의 폴 기반 이벤트 소스
  - Amazon Kinesis Data Streams
  - Amazon DynamoDB

Kinesis 또는 DynamoDB 스트림을 처리하는 Lambda 함수의 경우, 샤프 수가 동시 실행의 단위입니다. 스트림에 100개의 활성 샤프가 있으면 최대 100개의 함수 호출이 동시에 실행됩니다. 이는 각 샤프의 이벤트를 순차적으로 처리하기 때문입니다.

- 스트림 기반이 아닌 폴 기반 이벤트 소스: Amazon SQS 대기열을 처리하는 Lambda 함수의 경우, 각 메시지 배치가 단일 동시 단위로 간주되는 최대 동시성 레벨에 도달할 때까지 AWS Lambda가 자동으로 대기열에서 폴링 규모를 조정합니다. AWS Lambda의 자동 조정 동작은 대기열이 비었을 때는 폴링 비용을 낮게 유지하고 대기열에 부담이 갈 때는 높은 처리량을 달성하도록 설계되었습니다.

Amazon SQS 이벤트 소스 매핑이 초기에 활성화되면 Lambda는 Amazon SQS 대기열의 긴 폴링을 시작합니다. 긴 폴링을 사용하면 빈 응답의 수를 줄이고 메시지가 도착할 때 최적의 처리 지연 시간을 제공하여 Amazon Simple Queue Service 폴링 비용을 절감할 수 있습니다.

대기열에 대한 메시지 유입이 증가하면 AWS Lambda는 동시 함수 실행 수가 1000개, 계정 동시성 한도 또는 함수 동시성 한도(선택 사항) 중 더 작은 값에 도달할 때까지 폴링 활동을 자동으로 확장합니다. Amazon Simple Queue Service에서는 5개의 동시 함수 호출로 초기 버스트를 지원하고 분당 60회의 동시 호출로 동시성을 높입니다.

#### Note

[계정 수준 제한](#)은 계정의 다른 함수에 영향을 받으며, 함수당 동시성은 함수로 전송한 모든 이벤트에 적용됩니다. 자세한 내용은 [동시성 관리 \(p. 37\)](#) 단원을 참조하십시오.

## 요청 속도

요청 속도는 Lambda 함수가 호출된 속도를 뜻합니다. 폴 기반 서비스를 제외한 모든 서비스에서 요청 속도는 이벤트 소스가 이벤트를 생성하는 속도입니다. 폴 기반 서비스에서 AWS Lambda는 다음과 같이 요청 속도를 계산합니다.

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

예를 들어 스트림에 5개의 활성 샤프가 있고(5개의 Lambda 함수가 동시에 실행) Lambda 함수를 호출하는 데 약 2초가 걸린다고 한다면 요청 속도는 초당 2.5 요청이 됩니다.

## 자동 조정

AWS Lambda는 증가된 트래픽에 응답하여 [동시성 한도 \(p. 7\)](#)까지 함수 실행을 동적으로 확장합니다. 로드가 지속되는 상황에서는 함수의 동시성이 500~3000회의 동시 실행 횟수에서 초기 레벨까지 증가합니다. 이 횟수는 지역에 따라 다릅니다. 최초의 증가 이후 함수의 용량은 로드가 수용될 때까지 또는 리전 내 모든 함수의 동시성 합계가 한도에 도달할 때까지 증가하는데, 매번 동시 실행이 500회씩 추가로 증가합니다.

리전	최초 동시성 증가
미국 동부(오하이오), 미국 서부(캘리포니아 북부 지역), 캐나다 (중부)	500
미국 서부(오레곤), 미국 동부(버지니아 북부)	3000
아시아 태평양(서울), 아시아 태평양(뭄바이), 아시아 태평양(싱가포르), 아시아 태평양(시드니)	500
아시아 태평양(도쿄)	1000
EU(런던), EU(파리), EU(스톡홀름)	500
EU(프랑크푸르트)	1000

리전	최초 동시성 증가
EU(아일랜드)	3000
남아메리카(상파울루)	500
중국(베이징), 중국(닝샤)	500
AWS GovCloud (US-West)	500

#### Note

함수가 VPC에 연결되면 [Amazon VPC 네트워크 인터페이스 제한](#)이 함수의 조정을 방지할 수 있습니다. 자세한 내용은 [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 \(p. 66\)](#) 단원을 참조하십시오.

조정을 제한하기 위해 예약된 동시성을 사용해 함수를 구성할 수 있습니다. 자세한 내용은 [동시성 관리 \(p. 37\)](#) 단원을 참조하십시오.

## AWS Lambda 함수 배달 못한 편지 대기열

비동기로 호출된 모든 Lambda 함수는 두 번 재시도한 뒤에 해당 이벤트를 무시하게 됩니다. 재시도에 실패했는데 이유를 모르는 경우, DLQ(배달 못한 편지 대기열)를 사용하여 처리되지 않은 이벤트를 Amazon SQS 대기열 또는 Amazon SNS 주제로 보내 결함을 분석해 볼 수 있습니다.

AWS Lambda는 처리할 수 없는 이벤트를 지정된 [Amazon SNS 주제](#) 또는 [Amazon SQS 대기열](#)로 보냅니다. DLQ를 지정하지 않은 함수는 재시도 횟수가 끝난 뒤 이벤트를 폐기합니다. 재시도에 대한 자세한 내용은 [AWS Lambda 재시도 동작 \(p. 83\)](#) 단원을 참조하십시오.

Lambda 함수의 `DeadLetterConfig` 파라미터에 Amazon 리소스 이름 `TargetArn` 값을 지정하여 DLQ를 구성할 수 있습니다.

```
{
  "Code": {
    "ZipFile": blob,
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string"
  },
  "Description": "string",
  "FunctionName": "string",
  "Handler": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
  "Publish": bool,
  "DeadLetterConfig": {
    "TargetArn": "string"
  }
}
```

또한 미처리 이벤트가 어떤 서비스로 전달되는지에 따라 Lambda 함수의 [실행 역할 \(p. 9\)](#)에 권한을 추가로 부여해야 합니다.

- Amazon SQS의 경우: [SendMessage](#)
- Amazon SNS의 경우: [Publish](#)

DLQ 대상 ARN에 기록된 페이로드는 메시지 본문을 수정하지 않은 원본 이벤트 페이로드입니다. 메시지 속성에는 이벤트가 처리되지 않은 이유를 이해하는 데 도움이 되는 정보가 포함되어 있습니다.

### DLQ 메시지 속성

이름	유형	값
RequestID	문자열	고유 요청 식별자
ErrorCode	번호	3자리 HTTP 오류 코드
ErrorMessage	문자열	오류 메시지(1KB로 잘림)

권한 문제로 인해 또는 메시지 총 크기가 대상 대기열이나 주제의 한도를 초과할 경우 DLQ 메시지가 대상에 전송되지 못할 수 있습니다. 예를 들어 본문이 256 KB에 근접한 Amazon SNS 알림이 오류를 발생시키는 함수를 트리거할 경우, Amazon SNS가 추가하고 Lambda가 추가한 속성과 결합된 추가 이벤트로 인해 메시지가 DLQ에 허용된 최대 크기를 초과할 수 있습니다. DLQ에 쓸 수 없을 경우, Lambda는 이벤트를 삭제하고 [DeadLetterErrors \(p. 223\)](#) 측정치를 내보냅니다.



이벤트 소스로 Amazon SQS를 사용 중인 경우 Lambda 함수가 아닌 Amazon SQS 대기열 자체에서 DLQ를 구성하십시오. 자세한 내용은 [Using AWS Lambda with Amazon SQS \(p. 211\)](#) 단원을 참조하십시오.

## AWS Command Line Interface에서 AWS Lambda 사용

AWS Command Line Interface를 사용하여 함수 및 기타 AWS Lambda 리소스를 관리할 수 있습니다. AWS CLI는 AWS SDK for Python (Boto)을 사용하여 Lambda API와 상호 작용합니다. 이를 통해 API에 대해 배운 후 그 지식을 바탕으로 AWS SDK에서 Lambda를 사용하는 애플리케이션을 빌드할 수 있습니다.

이 자습서에서는 AWS CLI를 사용하여 Lambda 함수를 관리하고 호출합니다.

### 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaBasicExecutionRole.
  - 역할 이름 – **lambda-cli-role**.

AWSLambdaBasicExecutionRole 정책은 함수가 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 함수 만들기

다음은 입력으로 이벤트를 수신하여, 수신 이벤트 데이터의 일부를 CloudWatch Logs에 로깅합니다.

Example index.js

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

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

함수를 만들려면

1. 샘플 코드를 index.js 파일에 복사합니다.
2. 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

3. `create-function` 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name helloworld \
--zip-file file://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/lambda-cli-role
{
    "FunctionName": "helloworld",
    "CodeSize": 351,
    "MemorySize": 128,
    "FunctionArn": "function-arn",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
    "Timeout": 3,
    "LastModified": "2015-04-07T22:02:58.854+0000",
    "Runtime": "nodejs8.10",
    "Description": ""
}
```

invoke 명령을 사용하여 Lambda 함수를 호출합니다.

```
$ aws lambda invoke --function-name helloworld --log-type Tail \
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}' \
outputfile.txt
{
    "LogResult": "base64-encoded-log",
    "StatusCode": 200
}
```

또한 이 명령은 --log-type 파라미터를 지정하여 함수에서 만든 로그의 테일 끝을 요청합니다. 응답의 로그 데이터는 base64 인코딩 형식입니다. base64 프로그램을 사용하여 로그를 해독하십시오.

```
$ echo base64-encoded-log | base64 --decode
START RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801      value1 = value1
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801      value2 = value2
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801      value3 = value3
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801      result: "value1"
END RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
REPORT RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
Duration: 13.35 ms      Billed Duration: 100 ms      Memory Size: 128 MB
Max Memory Used: 9 MB
```

기본 호출 유형(RequestResponse)을 사용하여 함수를 호출했기 때문에, 연결은 실행이 완료될 때까지 열린 상태로 유지됩니다. Lambda는 출력 파일에 응답을 씁니다.

## 계정의 Lambda 함수 목록 표시

다음 AWS CLI list-functions 명령을 실행하여, 생성한 함수의 목록을 검색할 수 있습니다.

```
$ aws lambda list-functions --max-items 10
{
    "Functions": [
        {
            "FunctionName": "helloworld",
            "MemorySize": 128,
            "CodeSize": 412,
            "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
            "Handler": "ProcessKinesisRecords.handler",
            "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
            "Timeout": 3,
            "LastModified": "2015-02-22T21:03:01.172+0000",
            "Runtime": "nodejs6.10",
            "Description": ""
        },
        {
            "FunctionName": "ProcessKinesisRecords",
            "MemorySize": 128,
            "CodeSize": 412,
            "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
            "Handler": "ProcessKinesisRecords.handler",
            "Role": "arn:aws:iam::account-id:role/lambda-execute-test-kinesis",
            "Timeout": 3,
            "LastModified": "2015-02-22T21:03:01.172+0000",
            "Runtime": "nodejs6.10",
            "Description": ""
        },
        ...
    ]
}
```

```
],
    "NextMarker": null
}
```

응답에서 Lambda는 최대 10개의 함수로 이루어진 목록을 반환합니다. 검색할 수 있는 함수가 추가로 있는 경우, `NextMarker`는 다음 `list-functions` 요청에서 사용할 수 있도록 마커를 제공합니다. 그렇지 않으면 값이 `null`이 됩니다. 다음의 `list-functions` AWS CLI 명령은 `--marker` 파라미터를 보여주는 예제입니다.

```
$ aws lambda list-functions --max-items 10 \
--marker value-of-NextMarker-from-previous-response
```

## Lambda 함수 검색

Lambda CLI `get-function` 명령은 함수의 배포 패키지를 다운로드할 때 사용할 수 있도록 Lambda 함수 메타데이터와 미리 서명된 URL을 반환합니다.

```
$ aws lambda get-function --function-name helloworld
{
    "Code": {
        "RepositoryType": "S3",
        "Location": "pre-signed-url"
    },
    "Configuration": {
        "FunctionName": "helloworld",
        "MemorySize": 128,
        "CodeSize": 287,
        "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
        "Timeout": 3,
        "LastModified": "2015-04-07T22:02:58.854+0000",
        "Runtime": "nodejs8.10",
        "Description": ""
    }
}
```

자세한 내용은 [GetFunction \(p. 374\)](#) 단원을 참조하십시오.

## 정리

다음 `delete-function` 명령을 실행하여 `helloworld` 함수를 삭제합니다.

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

IAM 콘솔에서 생성한 IAM 역할을 삭제합니다. 역할 삭제에 대한 자세한 내용은 IAM 사용 설명서의 [역할 또는 인스턴스 프로파일 삭제](#)를 참조하십시오.

# Android용 AWS Mobile SDK에서 Lambda 함수 호출

모바일 애플리케이션에서 Lambda 함수를 호출할 수 있습니다. 비즈니스 로직을 함수에 추가하여 개발 수명 주기와 프런트 엔드 클라이언트를 분리하면 모바일 애플리케이션을 보다 간편하게 개발하고 유지 관리할 수 있습니다. Android용 Mobile SDK에서 [Amazon Cognito](#)를 사용하여 사용자를 인증하고 요청에 권한을 부여 (p. 91)합니다.

모바일 애플리케이션에서 함수를 호출할 경우 이벤트 구조, [호출 유형 \(p. 80\)](#) 및 권한 모델을 선택합니다. [별칭 \(p. 50\)](#)을 사용하여 함수 코드에 대한 원활한 업데이트를 활성화할 수 있지만, 함수와 애플리케이션이 긴밀하게 결합됩니다. 함수를 추가할 때, API 계층을 생성하여 프런트 엔드 클라이언트로부터 함수 코드를 분리하고 성능을 개선할 수 있습니다.

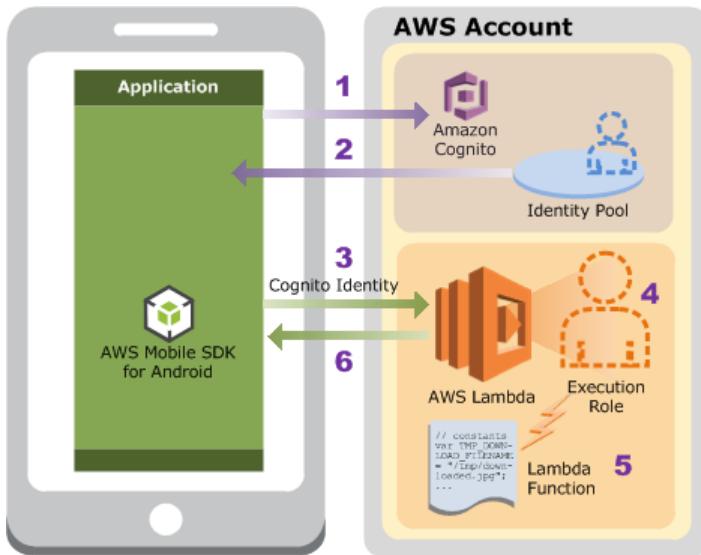
모바일 및 웹 애플리케이션을 위한 정식 기능의 웹 API를 생성하려면 Amazon API Gateway를 사용하십시오. API 게이트웨이를 사용하여 사용자 지정 권한 부여자를 추가하고, 요청을 조절하며, 모든 함수의 결과를 캐싱할 수 있습니다. 자세한 내용은 [Using AWS Lambda with Amazon API Gateway \(p. 131\)](#) 단원을 참조하십시오.

#### 주제

- [자습서: Android용 Mobile SDK에서 AWS Lambda 사용 \(p. 91\)](#)
- [샘플 함수 코드 \(p. 96\)](#)

## 자습서: Android용 Mobile SDK에서 AWS Lambda 사용

이 자습서에서는 Amazon Cognito를 사용하여 자격 증명을 받고 Lambda 함수를 호출하는 간단한 Android 모바일 애플리케이션을 생성합니다.



모바일 애플리케이션은 Amazon Cognito 자격 증명 풀에서 AWS 자격 증명을 검색한 후 이를 사용하여 요청 데이터를 포함하는 이벤트를 통해 Lambda 함수를 호출합니다. 이 함수는 요청을 처리하고 프런트 엔드에 응답을 반환합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셸 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaBasicExecutionRole.
  - 역할 이름 – **lambda-android-role**.

AWSLambdaBasicExecutionRole 정책은 함수가 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 함수 만들기

다음 예제는 데이터를 사용하여 문자열 응답을 생성합니다.

Note

다른 언어로 작성된 샘플 코드는 [샘플 함수 코드 \(p. 96\)](#) 단원을 참조하십시오.

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);
}
```

함수를 만들려면

1. 샘플 코드를 index.js 파일에 복사합니다.
2. 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

3. create-function 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name AndroidBackendLambdaFunction \
--zip-file file://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/lambda-android-role
```

## Lambda 함수 테스트

샘플 이벤트 데이터를 사용하여 수동으로 함수를 호출합니다.

### Lambda 함수를 테스트하려면(AWS CLI)

1. 다음과 같은 샘플 이벤트 JSON을 파일(input.txt)로 저장합니다.

```
{ "firstName": "first-name", "lastName": "last-name" }
```

2. 다음 invoke 명령을 실행합니다.

```
$ aws lambda invoke --function-name AndroidBackendLambdaFunction \
--payload file://file-path/input.txt outputfile.txt
```

## Amazon Cognito 자격 증명 풀 생성

이 단원에서는 Amazon Cognito 자격 증명 풀을 생성합니다. 자격 증명 풀은 두 가지 역할을 가집니다. 인증되지 않은 사용자를 위한 IAM 역할을 업데이트하고 AndroidBackendLambdaFunction Lambda 함수를 실행하기 위한 권한을 부여합니다.

IAM 역할에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할](#)을 참조하십시오. Amazon Cognito 서비스에 대한 자세한 내용은 [Amazon Cognito](#) 제품 세부 정보 페이지를 참조하십시오.

### 자격 증명 풀을 생성하려면

1. [Amazon Cognito 콘솔](#)을 엽니다.
2. JavaFunctionAndroidEventHandlerPool라는 새로운 자격 증명 풀을 생성합니다. 자격 증명 풀을 생성하기 위한 절차를 따르기 전에 다음에 유의하십시오.
  - 예제 모바일 애플리케이션에서는 사용자 로그인이 필요하지 않기 때문에 생성할 자격 증명 풀은 인증되지 않은 자격 증명에 대한 액세스를 허용해야 합니다. 따라서 인증되지 않은 자격 증명에 대한 액세스 활성화 옵션을 선택해야 합니다.
  - 인증되지 않은 자격 증명에 연결된 권한 정책에 다음 명령문을 추가합니다.

```
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda:us-
east-1:123456789012:function:AndroidBackendLambdaFunction"
    ]
}
```

그 결과 나온 정책은 다음과 같습니다.

```
{
    "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"  
    ]  
}  
}
```

자격 증명 풀을 생성하는 방법에 대한 자침은 [Amazon Cognito 콘솔](#)에 로그인하여 새 자격 증명 풀 마법사를 따르십시오.

- 자격 증명 풀 ID를 메모합니다. 다음 단원에서 생성하는 모바일 애플리케이션에서 이 ID를 지정합니다. 앱은 임시 보안 자격 증명을 요청하기 위해 Amazon Cognito에 요청을 전송할 때 이 ID를 사용합니다.

## Android 애플리케이션 만들기

이제 이벤트를 생성하는 간단한 Android 모바일 애플리케이션을 생성하고 이벤트 데이터를 파라미터로 전달하여 Lambda 함수를 호출합니다.

Android Studio를 사용하여 다음 지침들을 확인했습니다.

- 다음 구성을 사용하여 `AndroidEventGenerator`라는 Android 프로젝트를 새로 생성합니다.
  - [Phone and Tablet] 플랫폼을 선택합니다.
  - [Blank Activity]를 선택합니다.
- `build.gradle(Module:app)` 파일에서 `dependencies` 섹션에 다음을 추가합니다.

```
compile 'com.amazonaws:aws-android-sdk-core:2.2.+'  
compile 'com.amazonaws:aws-android-sdk-lambda:2.2.+'
```

- 필요에 따라 종속 프로그램을 다운로드할 수 있도록 프로젝트를 빌드합니다.
- Android 애플리케이션 매니페스트(`AndroidManifest.xml`)에서 애플리케이션이 인터넷에 연결되도록 다음과 같은 권한을 추가합니다. `</manifest>` 엔드 태그 바로 앞에 이들을 추가할 수 있습니다.

```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- `MainActivity`에 다음과 같은 가져오기를 추가합니다.

```
import com.amazonaws.mobileconnectors.lambdainvoker.*;  
import com.amazonaws.auth.CognitoCachingCredentialsProvider;  
import com.amazonaws.regions.Regions;
```

- package 섹션에서 두 가지 클래스(`RequestClass` 및 `ResponseClass`)를 추가합니다. POJO는 이전 섹션에서 함수에서 생성한 POJO와 동일합니다.
  - `RequestClass` 이 클래스의 인스턴스는 이름과 성으로 이루어진 이벤트 데이터에서 POJO(Plain Old Java Object) 역할을 합니다. 이전 단원에서 생성한 Lambda 함수에서 Java 예제를 사용하고 있는 경우, 이 POJO는 Lambda 함수 코드에서 생성한 POJO와 동일합니다.

```
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. 같은 패키지에서 AndroidBackendLambdaFunction Lambda 함수를 호출하기 위해 MyInterface라는 인터페이스를 생성합니다.

```
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);
}
```

}

코드의 `@LambdaFunction` 주석은 같은 이름의 Lambda 함수에 특정 클라이언트 메서드를 매핑합니다. 이 주석에 대한 자세한 내용은 Android용 AWS Mobile SDK Developer Guide의 [AWS Lambda](#)를 참조하십시오.

8. 애플리케이션을 단순하게 유지하기 위해 `onCreate()` 이벤트 핸들러에서 Lambda 함수를 호출할 수 있는 코드를 추가합니다. `MainActivity`에서 `onCreate()` 코드 끝에 다음과 같은 코드를 추가합니다.

```
// 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. 코드를 실행하고 다음과 같이 이를 확인합니다.

- `Toast.makeText()`는 반환된 응답을 표시합니다.
- CloudWatch Logs가 Lambda 함수에서 생성된 로그를 표시하는지 확인합니다. 이벤트 데이터(이름과 성)를 표시해야 합니다. 콘솔에서 이를 확인할 수도 있습니다.

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

주제

- [Node.js \(p. 97\)](#)
- [Java \(p. 97\)](#)

## Node.js

다음 예제는 데이터를 사용하여 문자열 응답을 생성합니다.

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);
}
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Java

다음 예제는 데이터를 사용하여 문자열 응답을 생성합니다.

코드에서 handler(myHandler)는 입력 및 출력에 RequestClass 및 ResponseClass 유형을 사용합니다. 코드는 이들 유형을 구현합니다.

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;
        }
    }

    public String handleRequest(RequestClass request, Context context) {
        String response = "Hello " + request.getFirstName() + " " + request.getLastName();
        return response;
    }
}
```

```
        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);
}
}
```

## 종속성

- aws-lambda-java-core

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

# AWS Lambda 런타임

AWS Lambda는 런타임 사용을 통해 여러 언어를 지원합니다. 함수를 생성할 때 런타임을 선택하며, 함수의 구성은 업데이트하여 런타임을 변경할 수 있습니다. 기본 실행 환경은 함수 코드에서 액세스할 수 있는 추가 라이브러리와 [환경 변수 \(p. 100\)](#)를 제공합니다.

## Amazon Linux

- AMI – [amzn-ami-hvm-2018.03.0.20181129-x86\\_64-gp2](#)
- Linux 커널 – 4.14.114-93.126.amzn2.x86\_64 or 4.14.114-83.126.amzn1.x86\_64

### Note

Lambda는 Amazon Linux 2018.03으로 업그레이드됩니다. 자세한 내용은 [Upcoming updates to the AWS Lambda and AWS Lambda@Edge execution environment\(AWS Lambda 및 AWS Lambda@Edge 실행 환경에 대한 예정된 업데이트\)](#)를 참조하십시오.

## Amazon Linux 2

- AMI – [amzn2-ami-hvm-2.0.20190313-x86\\_64-gp2](#)
- Linux 커널 – 4.14.114-93.126.amzn2.x86\_64

함수가 호출되면 Lambda는 이전 호출의 실행 환경을 사용할 수 있을 경우 그 실행 환경을 재사용하려고 시도합니다. 이렇게 하면 실행 환경을 준비하는 시간이 절약되고, [실행 콘텍스트 \(p. 101\)](#)의 데이터베이스 연결 및 임시 파일 등과 같은 리소스를 저장하여 함수가 실행될 때마다 그러한 리소스가 생성되는 상황을 막을 수 있습니다.

런타임은 단일 버전의 언어, 여러 버전의 언어 또는 여러 언어를 지원할 수 있습니다. 언어 또는 프레임워크 버전이 수명 종료되면 해당 버전의 런타임이 [더 이상 사용되지 않습니다 \(p. 102\)](#).

## Node.js 런타임

이름	식별자	Node.js 버전	JavaScript용 AWS SDK	운영 체제
Node.js 10	<code>nodejs10.x</code>	10.15	2.437.0	Amazon Linux 2
Node.js 8.10	<code>nodejs8.10</code>	8.10	2.290.0	Amazon Linux

## Python 런타임

이름	식별자	Python용 AWS SDK	운영 체제
Python 3.6	<code>python3.6</code>	boto3-1.7.74 botocore-1.10.74	Amazon Linux
Python 3.7	<code>python3.7</code>	boto3-1.9.42 botocore-1.12.42	Amazon Linux
Python 2.7	<code>python2.7</code>	해당 사항 없음	Amazon Linux

### Ruby 런타임

이름	식별자	운영 체제
Ruby 2.5	ruby2.5	Amazon Linux

### Java 런타임

이름	식별자	JDK	운영 체제
Java 8	java8	java-1.8.0-openjdk	Amazon Linux

### Go 런타임

이름	식별자	운영 체제
Go 1.x	go1.x	Amazon Linux

### .NET 런타임

이름	식별자	언어	운영 체제
.NET Core 2.1	dotnetcore2.1	C# PowerShell Core 6.0	Amazon Linux
.NET Core 1.0	dotnetcore1.0	C#	Amazon Linux

Lambda에서 다른 언어를 사용하기 위해 [사용자 지정 런타임 \(p. 103\)](#)을 구현할 수 있습니다. Lambda 실행 환경은 호출 이벤트를 받고 응답을 보내기 위한 [런타임 인터페이스 \(p. 105\)](#)를 제공합니다. 사용자 지정 런타임은 함수 코드와 함께 배포하거나 하나의 [계층 \(p. 62\)](#)에서 배포할 수 있습니다.

#### 주제

- [Lambda 함수에서 사용할 수 있는 환경 변수 \(p. 100\)](#)
- [AWS Lambda 실행 콘텍스트 \(p. 101\)](#)
- [런타임 지원 정책 \(p. 102\)](#)
- [사용자 지정 AWS Lambda 런타임 \(p. 103\)](#)
- [AWS Lambda 런타임 인터페이스 \(p. 105\)](#)
- [자습서 – 사용자 지정 런타임 게시 \(p. 107\)](#)

## Lambda 함수에서 사용할 수 있는 환경 변수

다음은 AWS Lambda 실행 환경의 일부이자 Lambda 함수에서 사용할 수 있는 환경 변수 목록입니다. 아래 표는 AWS Lambda가 예약하여 변경이 불가능한 환경 변수를 비롯해 Lambda 함수를 생성할 때 설정할 수 있는 환경 변수를 보여줍니다. Lambda 함수에서 환경 변수를 사용하는 방법에 대한 자세한 내용은 [AWS Lambda 환경 변수 \(p. 39\)](#) 단원을 참조하십시오.

#### Lambda 환경 변수

키	예약	값
_HANDLER	예	함수에 대해 구성된 핸들러 위치.

키	예약	값
AWS_REGION	예	Lambda 함수가 실행되는 AWS 리전입니다.
AWS_EXECUTION_ENV	예	런타임 ID (p. 99)이며 앞에 AWS_Lambda_가 붙습니다. 예: AWS_Lambda_java8.
AWS_LAMBDA_FUNCTION_NAME	예	; 함수의 이름.
AWS_LAMBDA_FUNCTION_MEMORY_SIZE	예	함수에 사용 가능한 총 스왑 메모리 양(MB).
AWS_LAMBDA_FUNCTION_VERSION	예	실행할 함수의 버전.
AWS_LAMBDA_LOG_GROUP_NAME	예	함수의 Amazon CloudWatch Logs 그룹 및 스트림 이름.
AWS_LAMBDA_LOG_STREAM_NAME		
AWS_ACCESS_KEY_ID	예	함수의 실행 역할 (p. 9)에서 가져온 액세스 키.
AWS_SECRET_ACCESS_KEY		
AWS_SESSION_TOKEN		
LANG	아니요	en_US.UTF-8입니다. 이것은 런타임의 로캘입니다.
TZ	예	환경의 표준 시간대(UTC). 실행 환경에서는 NTP를 사용하여 시스템 클록을 동기화합니다.
LAMBDA_TASK_ROOT	예	Lambda 함수 코드의 경로.
LAMBDA_RUNTIME_DIR	예	런타임 라이브러리의 경로.
PATH	아니요	/usr/local/bin:/usr/bin/:/bin:/opt/bin
LD_LIBRARY_PATH	아니요	/lib64:/usr/lib64:\$LAMBDA_RUNTIME_DIR:\$LAMBDA_RUNTIME_DIR/lib:\$LAMBDA_TASK_ROOT:\$LAMBDA_TASK_ROOT/lib:/opt/lib
NODE_PATH	아니요	(Node.js) /opt/nodejs/node8/node_modules/:/opt/nodejs/node_modules:\$LAMBDA_RUNTIME_DIR/node_modules
PYTHONPATH	아니요	(Python) \$LAMBDA_RUNTIME_DIR.
GEM_PATH	아니요	(Ruby) \$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0.
AWS_LAMBDA_RUNTIME_API	예	(사용자 지정 런타임) 런타임 API (p. 105)의 호스트 및 포트입니다.

## AWS Lambda 실행 콘텍스트

AWS Lambda는 Lambda 함수를 실행할 때 Lambda 함수를 실행하는 데 필요한 리소스를 프로비저닝하고 관리합니다. Lambda 함수를 생성할 때 Lambda 함수에서 허용하고 싶은 메모리 용량 및 최대 실행 시간 같은 구성 정보를 지정합니다. Lambda 함수가 호출되면 AWS Lambda는 사용자가 제공하는 구성 설정에 따라 실행 콘텍스트를 시작합니다. 실행 콘텍스트는 데이터베이스 연결 또는 HTTP 엔드포인트와 같은 Lambda 함수 코드의 외부 종속성을 초기화하는 임시 런타임 환경에 속합니다. 이렇게 하면 후속 호출 시 성능이 더 향상되는데 그 이유는 아래에 설명한 것처럼 외부 종속성을 "콜드 스타트(cold-start)"하거나 초기화할 필요가 없기 때문입니다.

실행 콘텍스트를 설정하고 필요한 "부트스트래핑"을 실행하기까지 다소 시간이 소요되기 때문에 Lambda 함수가 호출될 때마다 약간의 지연 시간이 추가될 수 있습니다. 이러한 지연 시간은 보통 Lambda 함수가 처음으로 호출될 때나 업데이트된 이후에 발생합니다. 왜냐하면 AWS Lambda가 Lambda 함수를 후속 호출할 때 실행 콘텍스트를 재사용하려고 하기 때문입니다.

Lambda 함수가 실행되고 나면 AWS Lambda는 또 다른 Lambda 함수 호출을 예상하여 실행 콘텍스트를 일정 시간 동안 유지합니다. 실제로, 이 서비스는 Lambda 함수가 완료된 이후에 실행 콘텍스트를 일시 중지했다가 Lambda 함수가 다시 호출될 때 AWS Lambda가 콘텍스트 재사용을 선택하면 콘텍스트를 다시 가동시켜 재사용합니다. 이렇게 실행 콘텍스트를 재사용하는 접근 방식은 다음과 같은 장점이 있습니다.

- Lambda 함수 코드의 모든 선언(handler 코드가 아닌 경우는 [프로그래밍 모델 \(p. 31\)](#) 참조)은 함수가 다시 호출될 때 추가로 최적화가 되도록 초기화된 상태로 유지됩니다. 예를 들어 함수가 연결을 재설정하는 대신 데이터베이스 연결을 설정하면 원래 연결이 후속 호출에 사용됩니다. 코드에 로직을 추가하여 생성에 앞서 연결이 이미 존재하는지 확인하는 것이 좋습니다.
- 각 실행 콘텍스트는 /tmp 디렉터리의 추가 디스크 공간의 512 MB를 제공합니다. 디렉터리 콘텐츠는 실행 콘텍스트가 일시 중지되어도 그대로 유지되기 때문에 일시적인 캐시를 여러 호출에서 사용할 수 있습니다. 코드를 추가하여 캐시에 저장한 데이터가 포함되어 있는지 확인할 수 있습니다. 배포 한도에 대한 자세한 내용은 [AWS Lambda 한도 \(p. 7\)](#)을 참조하십시오.
- 함수 종료 시 완료되지 않아서 Lambda 함수에서 초기화된 백그라운드 프로세스나 콜백은 AWS Lambda가 실행 콘텍스트 재사용을 선택한 경우에 재개됩니다. 코드가 존재하려면 먼저 코드의 백그라운드 프로세스나 콜백(Node.js의 경우)이 완료되어야 합니다.

Lambda 함수 코드를 기록할 때 AWS Lambda가 이후 함수 호출을 위해 실행 콘텍스트를 자동으로 재사용한다고 가정해서는 안 됩니다. 다른 요인으로 인해 AWS Lambda가 새 실행 콘텍스트를 생성해야 할 수 있으며 이 경우 데이터베이스 연결 실패 같은 예기치 못한 결과가 발생할 수 있습니다.

## 런타임 지원 정책

AWS Lambda 런타임은 유지 관리 및 보안 업데이트가 적용되는 운영 체제, 프로그래밍 언어 및 소프트웨어 라이브러리의 조합을 기반으로 구축됩니다. 런타임 구성 요소를 보안 업데이트가 더 이상 지원하지 않을 경우 Lambda는 런타임을 사용 중단합니다.

사용 중단은 두 단계로 수행됩니다. 첫 단계에서는 사용 중단된 런타임을 사용하는 함수를 더 이상 생성할 수 없습니다. 최소한 30일 동안은 사용 중단된 런타임을 사용하는 기존 함수를 계속 업데이트할 수 있습니다. 이 기간이 지나면 함수 생성 및 업데이트가 모두 영구적으로 비활성화됩니다. 하지만 호출 이벤트를 처리하는 데 함수를 계속 사용할 수 있습니다.

### 사용 중단 일정

이름	식별자	수명 종료	사용 중단(생성)	사용 중단(업데이트)
Node.js 0.10	nodejs	2016년 10월 31일	2016년 10월 31일	2016년 10월 31일
Node.js 4.3	nodejs4.3	2018년 30월 4일	2018년 12월 15일	2019년 4월 30일
	nodejs4.3-edge			
Node.js 6.10	nodejs6.10	2019년 4월 30일	2019년 4월 30일	2019년 6월 30일
.NET Core 2.0	dotnetcore2.0	2019년 4월 30일	2019년 4월 30일	2019년 5월 30일

대부분의 경우 언어 버전 또는 운영 체제의 수명 종료는 미리 공개됩니다. 앞으로 60일 내에 사용 중단될 런타임에서 실행하는 함수가 있을 경우, Lambda는 지원되는 런타임으로 함수를 마이그레이션하여 준비해야

한다고 이메일로 알려줍니다. 이전 버전과 호환되는 업데이트가 필요한 보안 문제나 LTS(장기 지원) 일정을 지원하지 않는 소프트웨어 등과 같은 일부 경우에는 사전 통지가 불가능할 수 있습니다.

런타임이 사용 중단되면 Lambda는 호출을 비활성화하여 언제든지 런타임의 사용을 완전히 중단시킬 수 있습니다. 사용 중단된 런타임은 보안 업데이트 기술 지원을 받을 수 없습니다. 런타임 수명이 종료되기 전에 Lambda는 해당 고객에게 추가로 알림을 보냅니다. 현재, 사용 중단이 예정된 런타임이 없습니다.

## 사용자 지정 AWS Lambda 런타임

AWS Lambda 런타임은 모든 프로그래밍 언어로 구현할 수 있습니다. 런타임은 함수가 호출될 때 Lambda 함수의 핸들러 메서드를 실행하는 프로그램입니다. 런타임은 `bootstrap`이라는 실행 파일의 형태로 함수의 배포 패키지에 포함될 수 있습니다.

런타임은 함수의 설정 코드를 실행하고, 환경 변수에서 핸들러 이름을 읽으며 Lambda 런타임 API에서 호출 이벤트를 읽는 기능을 담당합니다. 런타임은 이벤트 데이터를 함수 핸들러에 전달하고 핸들러의 응답을 Lambda에 다시 게시합니다.

사용자 지정 런타임은 표준 Lambda 실행 환경 (p. 99)에서 실행됩니다. 이 런타임은 셀 스크립트, Amazon Linux에 포함된 언어로 된 스크립트 또는 Amazon Linux에 컴파일된 바이너리 실행 파일일 수 있습니다.

사용자 지정 런타임을 시작하려면 [자습서 – 사용자 지정 런타임 게시 \(p. 107\)](#) 단원을 참조하십시오. GitHub에서 C++로 구현된 사용자 지정 런타임을 [awslabs/aws-lambda-cpp](#)에서 탐색할 수도 있습니다.

### 주제

- [사용자 지정 런타임 사용 \(p. 103\)](#)
- [사용자 지정 런타임 빌드 \(p. 103\)](#)

## 사용자 지정 런타임 사용

사용자 지정 런타임을 사용하려면 함수의 런타임을 `provided`로 설정하십시오. 런타임은 함수의 배포 패키지 또는 [계층 \(p. 62\)](#)에 포함될 수 있습니다.

### Example function.zip

```
.
### bootstrap
### function.sh
```

배포 패키지에 `bootstrap`이라는 이름의 파일이 있을 경우, Lambda이 이 파일을 실행합니다. 그렇지 않은 경우, Lambda은 함수의 계층에서 런타임을 찾습니다. 부트스트랩 파일을 찾을 수 없거나 실행할 수 없는 경우, 함수는 호출 시 오류를 반환합니다.

## 사용자 지정 런타임 빌드

사용자 지정 런타임의 진입점은 `bootstrap`이라는 이름의 실행 파일입니다. 부트스트랩 파일은 런타임일 수 있으며 혹은 런타임을 생성하는 다른 파일을 호출할 수도 있습니다. 다음 예제에서는 번들 버전의 Node.js를 사용하여 `runtime.js`라는 별개의 파일에서 JavaScript 런타임을 실행합니다.

### Example 부트스트랩

```
#!/bin/sh
```

```
cd $LAMBDA_TASK_ROOT
./node-v11.1.0-linux-x64/bin/node runtime.js
```

런타임 코드는 일부 초기화 작업을 완료하는 역할을 담당합니다. 그런 다음, 이 코드는 종료될 때까지 호출 이벤트를 루프에서 처리합니다. 초기화 작업은 [함수의 인스턴스 당 \(p. 101\)](#) 한 번씩 실행되어 호출을 처리할 수 있는 환경을 준비합니다.

### 초기화 작업

- 설정 검색 – 환경 변수를 읽어 함수 및 환경에 관한 세부 정보를 확인합니다.
  - \_HANDLER – 함수의 구성에서 핸들러에 대한 위치입니다. 표준 형식은 `file.method`이며, 여기서 `file`은 확장명이 없는 파일의 이름이고 `method`은 파일에 정의된 메서드 또는 함수의 이름입니다.
  - LAMBDA\_TASK\_ROOT – 함수 코드가 포함된 디렉터리입니다.
  - AWS\_LAMBDA\_RUNTIME\_API – 런타임 API의 호스트 및 포트입니다.

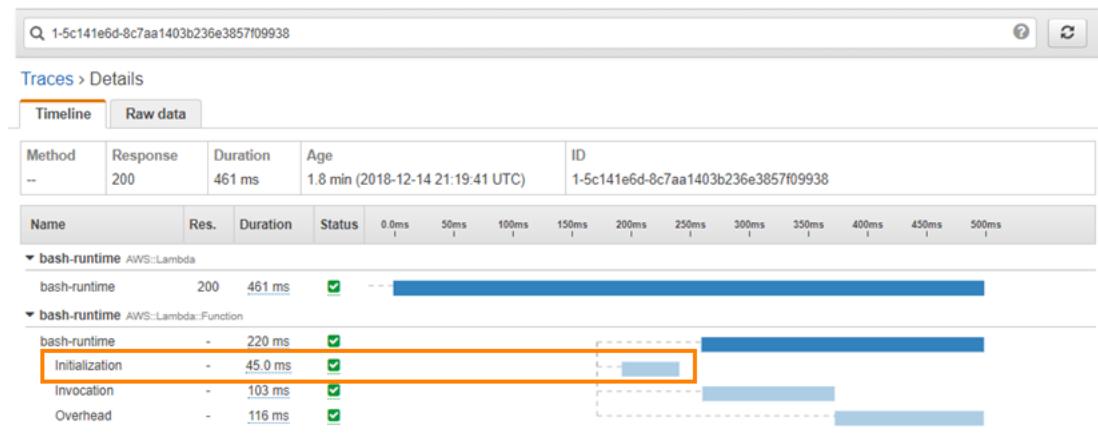
사용 가능한 변수의 전체 목록은 [Lambda 함수에서 사용할 수 있는 환경 변수 \(p. 100\)](#) 단원을 참조하십시오.

- 함수 초기화 – 핸들러 파일을 로드하고 이 파일에 포함된 전역적 또는 정적 코드를 실행합니다. 함수는 SDK 클라이언트 및 데이터베이스 연결과 같은 정적 리소스를 한 번 생성해야 하며, 다중 호출 시 그러한 리소스를 다시 사용해야 합니다.
- 오류 처리 – 오류가 발생하면 [초기화 오류 \(p. 107\)](#) API를 호출하고 즉시 종료하십시오.

초기화 횟수는 청구 대상인 실행 시간 및 시간 초과에 반영됩니다. 실행으로 인해 새 함수 인스턴스의 초기화가 트리거되면 로그 및 [AWS X-Ray 추적 \(p. 226\)](#)에서 초기화 시간을 볼 수 있습니다.

### Example 로그

```
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
```



런타임은 실행 중에 [Lambda 런타임 인터페이스 \(p. 105\)](#)를 사용하여 수신 이벤트를 관리하고 오류를 보고합니다. 초기화 작업을 완료한 후, 런타임은 수신 이벤트를 루프에서 처리합니다.

### 작업 처리

- 이벤트 가져오기 – 다음 이벤트를 가져오려면 [다음 호출 \(p. 105\)](#) API를 호출하십시오. 응답 본문에는 이벤트 데이터가 포함됩니다. 응답 헤더에는 요청 ID 및 기타 정보가 포함됩니다.

- 트레이스 헤더 전파 – API 응답의 `Lambda-Runtime-Trace-Id` 헤더에서 X-Ray 트레이스 헤더를 가져옵니다. X-Ray SDK가 사용할 동일한 값으로 `_X_AMZN_TRACE_ID` 환경 변수를 설정하십시오.
- 컨텍스트 객체 생성 – API 응답에서 환경 변수 및 헤더의 컨텍스트 정보를 사용하여 객체를 생성합니다.
- 함수 핸들러 호출 – 이벤트 및 컨텍스트 객체를 핸들러에 전달합니다.
- 응답 처리 – [호출 응답 \(p. 106\)](#) API를 호출하여 핸들러에서 응답을 게시합니다.
- 오류 처리 – 오류가 발생하면 [호출 오류 \(p. 106\)](#) API를 호출하십시오.
- 정리 – 사용하지 않은 리소스를 릴리스하거나 다른 서비스로 데이터를 전송하거나 혹은 다음 이벤트를 가져오기 전에 추가 작업을 수행하십시오.

함수의 배포 패키지에 런타임을 포함하거나 함수 계층에 런타임을 별도로 배포할 수 있습니다. 예제 연습을 보려면 [??? \(p. 107\)](#) 단원을 참조하십시오.

## AWS Lambda 런타임 인터페이스

AWS Lambda은 [사용자 지정 런타임 \(p. 103\)](#)에 HTTP API를 제공하여 Lambda에서 호출 이벤트를 수신하고 Lambda 실행 환경 ([p. 99](#))내에서 응답 데이터를 다시 전송합니다.

런타임 API 버전 2018-06-01에 대한 OpenAPI 사양은 [runtime-api.zip](#)에서 사용할 수 있습니다.

런타임은 `AWS_LAMBDA_RUNTIME_API` 환경 변수에서 엔드포인트를 가져와 API 버전을 추가하고 다음의 리소스 경로를 사용하여 API와 상호 작용합니다.

### Example 요청

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

#### 리소스

- [다음 호출 \(p. 105\)](#)
- [호출 응답 \(p. 106\)](#)
- [호출 오류 \(p. 106\)](#)
- [초기화 오류 \(p. 107\)](#)

## 다음 호출

경로 – `/runtime/invocation/next`

메서드 – GET

호출 이벤트를 검색합니다. 응답 본문에는 호출의 페이로드가 포함되어 있습니다. 이 페이로드는 함수 트리거의 이벤트 데이터를 포함하는 JSON 문서입니다. 응답 헤더에는 호출에 대한 추가 데이터가 포함되어 있습니다.

#### 응답 헤더

- `Lambda-Runtime-Aws-Request-Id` – 함수 호출을 트리거한 요청을 식별하는 요청 ID입니다.

예: `8476a536-e9f4-11e8-9739-2dfe598c3fcfd`.

- `Lambda-Runtime-Deadline-Ms` – 함수가 Unix 시간 형식에 따른 밀리초 단위의 시간 제한에 도달하는 날짜입니다.

예: `1542409706888`.

- **Lambda-Runtime-Invoked-Function-Arn** – 호출에 지정된 Lambda 함수, 버전 또는 별칭의 ARN입니다.  
예: arn:aws:lambda:us-east-2:123456789012:function:custom-runtime.
- **Lambda-Runtime-Trace-Id** – [AWS X-Ray 트레이스 헤더](#)입니다.  
예: Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1.
- **Lambda-Runtime-Client-Context** – AWS Mobile SDK 호출에서 클라이언트 애플리케이션 및 디바이스에 관한 데이터입니다.
- **Lambda-Runtime-Cognito-Identity** – AWS Mobile SDK 호출에서 Amazon Cognito 자격 증명 공급자에 관한 데이터입니다.

요청 ID는 Lambda 내에서 호출을 추적합니다. 응답을 전송할 때에는 이 ID를 사용하여 호출을 지정합니다.

트레이스 헤더에는 추적 ID, 상위 ID 및 샘플링 결정이 포함되어 있습니다. 요청이 샘플링될 경우, 이 요청은 Lambda 또는 업스트림 서비스에 의해 샘플링된 것입니다. 런타임은 헤더의 값을 사용해 `_X_AMZN_TRACE_ID`를 설정해야 합니다. X-Ray SDK는 이 값을 판독하여 ID를 가져온 다음, 요청을 추적할지 여부를 결정합니다.

## 호출 응답

경로 – /runtime/invocation/[AwsRequestId](#)/response

메서드 – POST

Lambda으로 호출 응답을 전송합니다. 런타임은 함수 핸들러를 호출한 후 함수의 응답을 호출 응답 경로에 게시합니다. 동기식 호출의 경우, Lambda은 응답을 클라이언트로 다시 보냅니다.

Example 성공 요청

```
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"
```

## 호출 오류

경로 – /runtime/invocation/[AwsRequestId](#)/error

메서드 – POST

함수가 오류를 반환하면 런타임은 JSON 문서로 오류를 포맷하고 호출 오류 경로에 게시합니다.

Example 요청 본문

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException"
}
```

Example 오류 요청

```
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"
```

## 초기화 오류

경로 – /runtime/init/error

메서드 – POST

런타임에서 초기화 도중에 오류가 발생하면 초기화 오류 경로에 오류 메시지가 게시됩니다.

Example 초기화 오류 요청

```
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"
```

## 자습서 – 사용자 지정 런타임 게시

이 자습서에서는 사용자 지정 런타임이 있는 Lambda 함수를 생성합니다. 먼저 함수의 배포 패키지에 런타임을 포함시킵니다. 그런 다음, 이 런타임을 함수와는 별도로 관리하는 계층으로 마이그레이션합니다. 마지막으로 리소스 기반 권한 정책을 업데이트하여 런타임 계층을 다른 모든 사용자와 공유합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

Lambda 함수를 생성하려면 IAM 역할이 필요합니다. 이 역할에는 로그를 CloudWatch Logs로 보내고 함수에서 사용하는 AWS 제품에 액세스하는 권한이 필요합니다. 함수 개발을 위한 역할이 없는 경우 이 역할을 하나 생성합니다.

### 실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – Lambda.
  - 권한 – AWSLambdaBasicExecutionRole.

- 역할 이름 – **lambda-role**.

AWSLambdaBasicExecutionRole 정책은 함수가 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가지고 있습니다.

## 함수 만들기

사용자 지정 런타임이 있는 Lambda 함수를 생성합니다. 이 예제는 2개의 파일 즉, 런타임 bootstrap 파일과 함수 핸들러를 포함합니다. 두 파일은 모두 Bash에서 구현됩니다.

런타임은 배포 패키지에서 함수 스크립트를 로드합니다. 런타임은 두 변수를 사용하여 스크립트를 찾습니다. LAMBDA\_TASK\_ROOT는 패키지가 추출된 위치를 알려주며 \_HANDLER에는 스크립트의 이름이 포함됩니다.

### Example 부트스트랩

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event
    EVENT_DATA=$(curl -ss -LD "$HEADERS" -X GET "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")
    REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d '[:space:]' | cut -d: -f2)

    # Execute the handler function from the script
    RESPONSE=$(($(_HANDLER" | cut -d. -f2) "$EVENT_DATA"))

    # Send the response
    curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/response" -d "$RESPONSE"
done
```

스크립트를 로드한 후, 런타임은 이벤트를 루프에서 처리합니다. 그리고 런타임 API를 사용하여 Lambda에서 호출 이벤트를 검색하고 이벤트를 핸들러로 전달하며 응답을 Lambda에 다시 게시합니다. 요청 ID를 가져오기 위해 런타임은 API 응답의 헤더를 임시 파일에 저장하고 이 파일에서 Lambda-Runtime-Aws-Request-Id 헤더를 읽습니다.

### Note

런타임은 오류 처리를 포함한 추가적인 책임이 있으며, 핸들러에 컨텍스트 정보를 제공합니다. 자세한 내용은 [사용자 지정 런타임 빌드 \(p. 103\)](#) 단원을 참조하십시오.

스크립트는 이벤트 데이터를 취하여 이를 stderr에 기록한 후 반환하는 핸들러 함수를 정의합니다.

### Example function.sh

```
function handler () {
    EVENT_DATA=$1
    echo "$EVENT_DATA" 1>&2;
    RESPONSE="Echoing request: '$EVENT_DATA'"
```

```
    echo $RESPONSE
}
```

두 파일을 모두 `runtime-tutorial`이라는 이름의 프로젝트 디렉터리에 저장합니다.

```
runtime-tutorial
# bootstrap
# function.sh
```

실행 파일들을 생성하여 이를 ZIP 아카이브에 추가합니다.

```
runtime-tutorial$ chmod 755 function.sh bootstrap
runtime-tutorial$ zip function.zip function.sh bootstrap
  adding: function.sh (deflated 24%)
  adding: bootstrap (deflated 39%)
```

`bash-runtime`이라는 이름의 함수를 생성합니다.

```
runtime-tutorial$ aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime provided \
--role arn:aws:iam::123456789012:role/lambda-role
{
  "FunctionName": "bash-runtime",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:bash-runtime",
  "Runtime": "provided",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "CodeSize": 831,
  "Description": "",
  "Timeout": 3,
  "MemorySize": 128,
  "LastModified": "2018-11-28T06:57:31.095+0000",
  "CodeSha256": "mv/xRv84LPCxdpcbKvmwuuFzwo7sLwUO1VxcUv3wKlM=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "2e1d51b0-6144-4763-8e5c-7d5672a01713"
}
```

함수를 호출하고 응답을 확인합니다.

```
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"}'
```

## 계층 생성

함수 코드에서 런타임 코드를 분리하려면 런타임만 포함하는 계층을 생성하십시오. 계층을 사용하면 함수의 종속 항목들을 독립적으로 개발할 수 있으며, 여러 함수가 있는 동일한 계층을 사용할 때 스토리지 사용량을 줄일 수 있습니다.

`bootstrap` 파일이 포함된 계층 아카이브를 생성합니다.

```
runtime-tutorial$ zip runtime.zip bootstrap
adding: bootstrap (deflated 39%)
```

publish-layer-version 명령을 사용하여 계층을 생성합니다.

```
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
fileb://runtime.zip
{
    "Content": {
        "Location": "https://awslambda-us-west-2-layers.s3.us-west-2.amazonaws.com/
snapshots/123456789012/bash-runtime-018c209b...",
        "CodeSha256": "bXVLhHi+D3H1QbDARUVPrDwlC7bssPxySQqt1QZqusE=",
        "CodeSize": 584,
        "UncompressedCodeSize": 0
    },
    "LayerArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime",
    "LayerVersionArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1",
    "Description": "",
    "CreatedDate": "2018-11-28T07:49:14.476+0000",
    "Version": 1
}
```

그러면 해당 계층의 첫 번째 버전이 생성됩니다.

## 함수 업데이트

함수와 함께 런타임 계층을 사용하려면 이 계층을 사용하도록 함수를 구성하고 함수에서 런타임 코드를 제거하십시오.

계층을 가져 오도록 함수 구성을 업데이트하십시오.

```
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
        }
    ]
    ...
}
```

이렇게 하면 런타임이 /opt 디렉터리 내 함수에 추가됩니다. Lambda은 함수의 배포 패키지에서 제거된 이러한 런타임만 사용합니다. 핸들러 스크립트만 포함하도록 함수 코드를 업데이트하십시오.

```
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
        }
    ]
}
```

```
        }
    ]
    ...
}
```

함수를 호출하여 런타임 계층에서 작동하는지 확인하십시오.

```
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"}'
```

## 런타임 업데이트

실행 환경에 관한 정보를 기록하려면 환경 변수를 출력하도록 런타임 스크립트를 업데이트하십시오.

### Example 부트스트랩

```
#!/bin/sh

set -euo pipefail

echo "## Environment variables:"
env

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"
...
```

새 코드를 사용해 계층의 두 번째 버전을 생성합니다.

```
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
```

새 버전의 계층을 사용하도록 함수를 구성합니다.

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:2
```

## 계층 공유

런타임 계층에 권한 설명문을 추가하여 이를 다른 계정과 공유하십시오.

```
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"
```

}

단일 계정, 조직 내 계정 또는 모든 계정에 권한을 부여하는 다중 문을 추가할 수 있습니다.

## 정리

각 버전의 계층을 삭제합니다.

```
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
```

이 함수는 계층의 버전 2에 대한 참조가 있으므로 Lambda 내에 여전히 존재합니다. 함수는 계속 작동하며, 다만 삭제된 버전을 사용하도록 함수를 구성할 수는 없습니다. 그런 다음, 함수의 계층 목록을 수정하면 새 버전을 지정하거나 삭제된 계층을 생략해야 합니다.

`delete-function` 명령을 사용해 자습서 함수를 삭제합니다.

```
runtime-tutorial$ aws lambda delete-function --function-name bash-runtime
```

# AWS Lambda 애플리케이션

AWS Lambda 애플리케이션은 작업을 수행하는 데 함께 작동하는 Lambda 함수, 이벤트 소스 및 기타 리소스의 조합입니다. AWS CloudFormation 및 기타 도구를 사용해 애플리케이션의 구성 요소를 하나의 리소스로 배포해 관리할 수 있는 단일 패키지로 수집할 수 있습니다. 애플리케이션은 Lambda 프로젝트를 이동하기 쉽게 만들어 AWS CodePipeline, AWS CodeBuild 및 AWS Serverless Application Model 명령줄 인터페이스 (SAM CLI) 등과 같은 추가 개발자 도구와 통합이 가능하게 합니다.

[AWS Serverless Application Repository](#)는 몇 번만 클릭하여 간편하게 계정에 배포할 수 있는 Lambda 애플리케이션 모음을 제공합니다. 리포지토리에는 바로 사용할 수 있는 애플리케이션과 프로젝트의 시작 지점으로 사용할 수 있는 샘플이 들어 있습니다. 또한 자체 프로젝트를 제출해 포함할 수도 있습니다.

[AWS CloudFormation](#)에서는 애플리케이션의 리소스를 정의하는 템플릿을 생성할 수 있고 애플리케이션을 스택으로 관리할 수 있습니다. 애플리케이션 스택에서는 리소스를 더욱 안전하게 추가 또는 수정할 수 있습니다. 한 부분이라도 업데이트에 실패하면 AWS CloudFormation에서는 이전 구성으로 자동으로 롤백합니다. AWS CloudFormation 파라미터를 사용해 동일한 템플릿에서 애플리케이션을 위한 여러 환경을 생성할 수 있습니다.

[AWS Serverless Application Model \(p. 116\)](#)(AWS SAM)은 더 높은 수준에서 서비스 애플리케이션을 정의할 수 있는 AWS CloudFormation 템플릿 언어를 위한 확장입니다. 이 모델은 함수 역할 생성 등과 같은 일반적인 작업을 추출해 템플릿 작성을 더욱 손쉽게 만듭니다. AWS SAM은 AWS CloudFormation에서 직접 지원되며 AWS CLI 및 AWS SAM CLI를 통해 추가 함수를 포함할 수 있습니다.

[AWS CLI](#) 및 [AWS SAM CLI](#)는 Lambda 애플리케이션 스택을 관리하기 위한 명령줄 도구입니다. AWS CloudFormation API를 사용해 애플리케이션 스택을 관리하기 위한 명령 이외에 AWS CLI는 배포 패키지 업로드 및 템플릿 업데이트 등과 같은 작업을 간소화하는 상위 수준 명령을 지원합니다. AWS SAM CLI는 템플릿 확인 및 로컬 테스트를 포함하여 추가 함수를 제공합니다.

## 주제

- [AWS Lambda 콘솔에서 애플리케이션 관리 \(p. 113\)](#)
- [AWS 서비스 애플리케이션 모델\(AWS SAM\) 사용 \(p. 116\)](#)
- [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#)
- [AWS CodePipeline를 사용하여 Lambda 애플리케이션에 대해 지속적 전송\(CD\) 파이프라인 구축 \(p. 119\)](#)
- [AWS Lambda 함수 작업의 모범 사례 \(p. 124\)](#)

## AWS Lambda 콘솔에서 애플리케이션 관리

AWS Lambda 콘솔에서는 [Lambda 애플리케이션 \(p. 113\)](#)을 모니터링 및 관리할 수 있습니다. 애플리케이션 메뉴에는 Lambda 함수와 함께 AWS CloudFormation 스택이 나열됩니다. 이 메뉴에는 AWS CloudFormation 콘솔, AWS Serverless Application Repository, AWS CLI 또는 AWS SAM CLI를 사용하여 AWS CloudFormation에서 실행한 스택이 포함되어 있습니다.

Lambda 애플리케이션을 보려면

1. [Lambda 콘솔](#)을 엽니다.
2. 애플리케이션을 선택합니다.

### 3. 애플리케이션을 선택합니다.

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

개요에는 애플리케이션에 대한 다음 정보가 표시됩니다.

- AWS CloudFormation 템플릿 또는 SAM 템플릿 – 애플리케이션을 정의하는 템플릿
- 리소스 – 애플리케이션 템플릿에서 정의하는 AWS 리소스. 애플리케이션의 Lambda 함수를 관리하려면 목록에서 함수 이름을 선택합니다.

## 애플리케이션 모니터링

모니터링 탭에는 애플리케이션의 리소스에 대한 집계 지표와 함께 Amazon CloudWatch 대시보드가 표시됩니다.

Lambda 애플리케이션을 모니터링하려면

1. [Lambda 콘솔](#)을 엽니다.
2. 애플리케이션을 선택합니다.
3. [Monitoring]을 선택합니다.

기본적으로 Lambda 콘솔에는 기본 대시보드가 표시됩니다. 애플리케이션 템플릿에서 사용자 지정 대시보드를 정의해 이 페이지를 사용자 지정할 수 있습니다. 템플릿에 대시보드가 한 개 이상 포함되어 있는 경우 이 페이지에는 기본 대시보드가 아니라 사용자 지정 대시보드가 표시됩니다. 페이지 오른쪽 상단에 있는 드롭다운 메뉴를 사용해 대시보드 간에 전환할 수 있습니다.

## 사용자 지정 대시보드

[AWS::CloudWatch::Dashboard](#) 리소스 유형을 사용해 애플리케이션 템플릿에 Amazon CloudWatch 대시보드를 한 개 이상 추가하여 애플리케이션 모니터링 페이지를 사용자 지정합니다. 다음 예에서는 `my-function` 함수의 호출 횟수를 그래프로 보여주는 단일 위젯이 있는 대시보드를 만듭니다.

Example 함수 대시보드 템플릿

```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
          "widgets": [
            {
              "type": "metric",
              "width": 12,

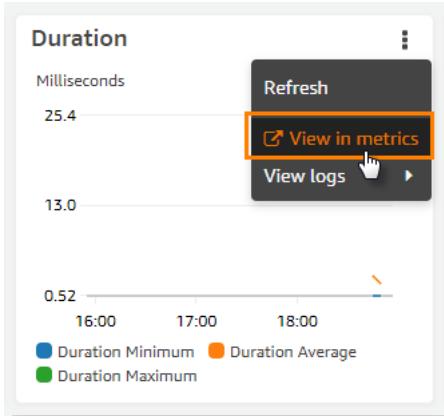
```

```
"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
}
]
```

CloudWatch 콘솔의 기본 모니터링 대시보드에서 위젯에 대한 정의를 확인할 수 있습니다.

#### 위젯 정의를 보려면

1. [Lambda 콘솔](#)을 엽니다.
2. 애플리케이션을 선택합니다.
3. 표준 대시보드가 있는 애플리케이션을 선택합니다.
4. [Monitoring]을 선택합니다.
5. 위젯의 드롭다운 메뉴에서 지표에서 보기 선택합니다.



6. [Source]를 선택합니다.

CloudWatch 대시보드 및 위젯 작성에 대한 자세한 내용은 Amazon CloudWatch API 참조의 [대시보드 본문 구조 및 구문](#)을 참조하십시오.

## AWS 서비스 애플리케이션 모델(AWS SAM) 사용

AWS Serverless Application Model(AWS SAM)은 AWS에서 서비스 애플리케이션을 빌드하는 데 사용할 수 있는 오픈 소스 프레임워크입니다. 이 프레임워크는 서비스 애플리케이션을 정의하는 데 사용하는 AWS SAM 템플릿 사양과, 서비스 애플리케이션을 빌드 및 테스트하고 배포하는 데 사용하는 AWS SAM 명령줄 인터페이스(AWS SAM CLI)로 구성됩니다.

AWS SAM에 대한 자세한 내용은 [AWS SAM이란 무엇인가?](#)를 참조하십시오.

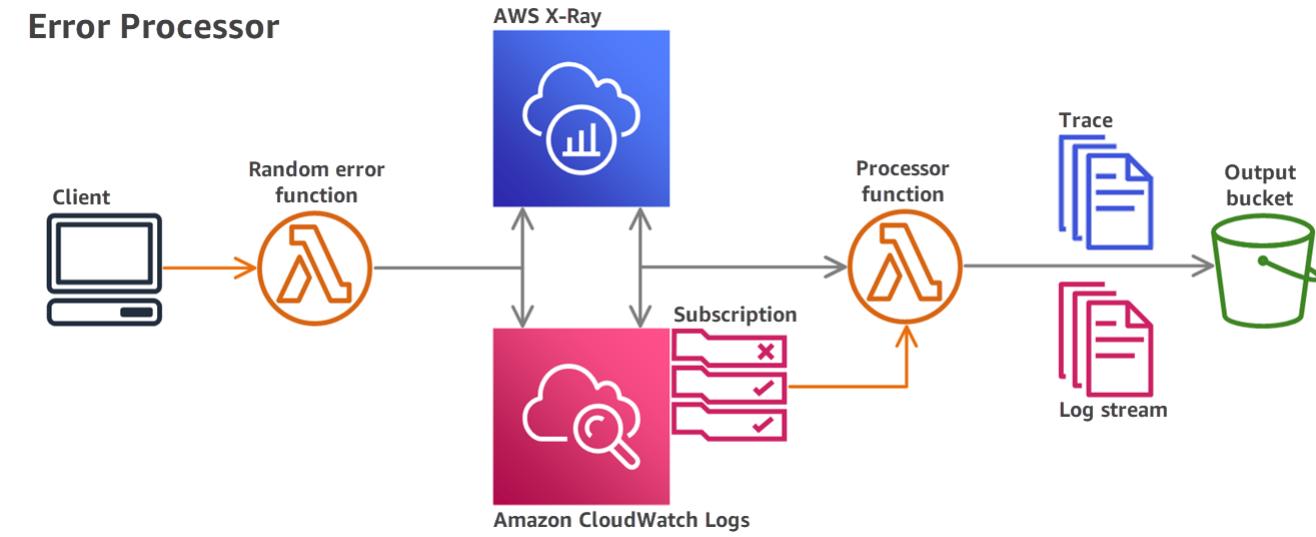
AWS SAM 템플릿에 대한 자세한 내용은 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

Lambda 함수를 로컬로 테스트하는 방법은 [서비스 애플리케이션 테스트 및 디버그](#)를 참조하십시오.

서비스 애플리케이션의 자동 배포에 대한 자세한 내용은 [서비스 애플리케이션 배포](#)를 참조하십시오.

## AWS Lambda용 오류 처리자 샘플 애플리케이션

오류 처리자 샘플 애플리케이션은 AWS Lambda를 사용하여 [Amazon CloudWatch Logs 구독](#) (p. 158)의 이벤트를 처리하는 것을 보여 줍니다. CloudWatch Logs를 사용하여 로그 항목이 패턴과 일치할 때 Lambda 함수를 호출할 수 있습니다. 이 애플리케이션의 구독은 함수 로그 그룹에서 ERROR라는 단어를 포함하는 항목을 모니터링합니다. 그리고 응답으로 프로세서 Lambda 함수를 호출합니다. 처리자 함수는 전체 로그 스트림을 검색하여 오류를 야기한 요청의 데이터를 추적하고, 나중에 사용하기 위해 이를 저장합니다.



함수 코드는 다음 파일에서 사용할 수 있습니다.

- 무작위 오류 – [random-error/index.js](#)
- 처리자 – [processor/index.js](#)

AWS CLI와 AWS CloudFormation을 사용하여 단 몇 분 내에 샘플을 배포할 수 있습니다. [README](#)의 지침에 따라 샘플을 다운로드하고 구성하여 해당 계정에 배포하십시오.

### 섹션

- [아키텍처 및 이벤트 구조](#) (p. 117)
- [AWS X-Ray를 사용하여 계측](#) (p. 117)
- [AWS CloudFormation 템플릿과 추가 리소스](#) (p. 118)

## 아키텍처 및 이벤트 구조

샘플 애플리케이션은 다음과 같은 AWS 서비스를 사용합니다.

- AWS Lambda – 함수 코드를 실행하여 CloudWatch Logs로 전송하고 추적 데이터를 X-Ray로 전송합니다.
- Amazon CloudWatch Logs – 로그를 수집하고, 로그 항목이 필터 패턴과 일치하면 함수를 호출합니다.
- AWS X-Ray – 추적 데이터를 수집하고, 검색을 위해 추적을 인덱스화하고, 서비스 맵을 생성합니다.
- Amazon Simple Storage Service(Amazon S3) – 배포 결과물과 애플리케이션 출력을 저장합니다.
- AWS CloudFormation – 애플리케이션 리소스를 생성하고 함수 코드를 배포합니다.

애플리케이션의 Lambda 함수는 무작위로 오류를 생성합니다. CloudWatch Logs가 함수의 로그에서 ERROR라는 단어를 탐지하면 처리를 위해 이벤트를 처리자 함수로 전송합니다.

Example – CloudWatch Logs 메시지 이벤트

```
{  
    "awslogs": {  
        "data": "H4sIAAAAAAAHWQTO/DMAzFv0vEkbLYcdJkt4qVXmCDteIAm1DbZKjs  
+kdpB0Jo350MhsQFyVLsZ+unl/fJWjeO5asrPgbH5..."  
    }  
}
```

디코딩할 경우, 데이터에는 로그 이벤트에 대한 세부 정보가 포함됩니다. 함수는 이 세부 정보를 사용하여 그 스트림을 식별하고, 로그 메시지를 구문 분석하여 오류를 야기한 요청의 ID를 확인합니다.

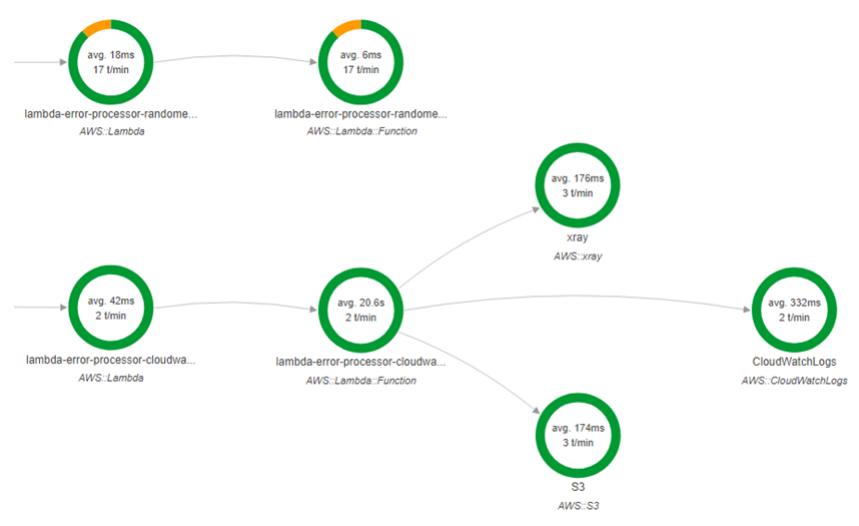
Example – 디코딩된 CloudWatch Logs 이벤트 데이터

```
{  
    "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\t1d2c1444-efd1-43ec-b16e-8fb2d37508b8\TERROR\n"  
        }  
    ]  
}
```

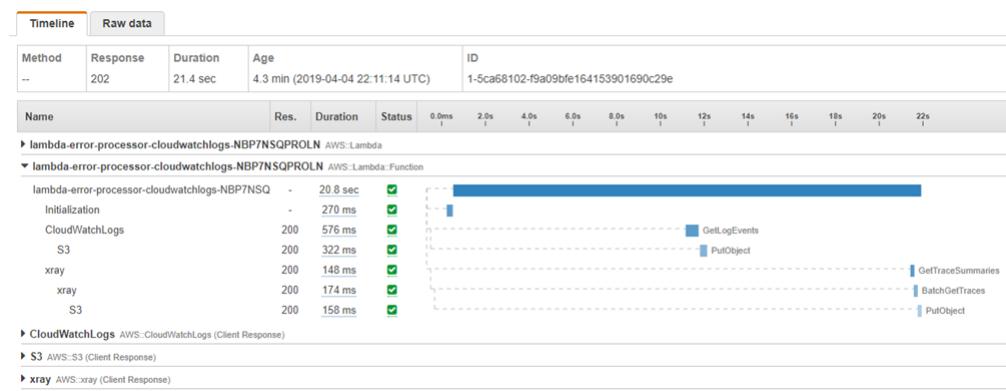
처리자 함수는 CloudWatch Logs 이벤트의 정보를 사용하여, 오류를 야기한 요청에 대한 전체 로그 스트림과 X-Ray 추적을 다운로드하여 Amazon S3 버킷에 저장합니다. 로그 스트림과 추적 시간이 끝날 수 있도록, 함수는 데이터에 액세스하기 전에 잠깐 기다립니다.

## AWS X-Ray를 사용하여 계측

이 애플리케이션은 [AWS X-Ray \(p. 226\)](#)를 사용하여 함수 호출 및 AWS 서비스에 대한 함수의 호출을 추적합니다. X-Ray는 함수로부터 받은 추적 데이터를 사용하여 오류 식별에 도움이 되는 서비스 맵을 생성합니다. 다음 서비스 맵은 일부 요청에 대한 오류를 생성하는 무작위 오류 함수를 보여줍니다. 또한 X-Ray, CloudWatch Logs, Amazon S3(이)라고 하는 처리자 함수도 확인할 수 있습니다.



두 가지 Node.js 함수는 템플릿에서 활성 추적으로 구성되며, Node.js용 AWS X-Ray SDK를 사용하여 코드로 계측화됩니다. 활성 추적을 통해, Lambda 태그는 수신되는 요청에 추적 헤더를 추가하고, 추적을 시간 정보와 함께 X-Ray로 전송합니다. 또한 무작위 오류 함수는 X-Ray SDK를 사용하여 요청 ID와 사용자 정보를 주석에 기록합니다. 주석은 추적에 첨부되며, 이를 이용해 특정 요청에 대한 추적 위치를 찾을 수 있습니다.



처리자 함수는 CloudWatch Logs 이벤트로부터 요청 ID를 받은 후 AWS SDK for JavaScript를 사용하여 해당 요청에 대한 X-Ray를 검색합니다. 그리고 X-Ray SDK를 통해 계측한 AWS SDK 클라이언트를 사용하여 추적 및 로그 스트림을 다운로드합니다. 그런 다음 이를 출력 버킷에 저장합니다. X-Ray SDK는 이러한 호출을 기록하며, 기록된 호출은 추적에 하위 세그먼트로 나타납니다.

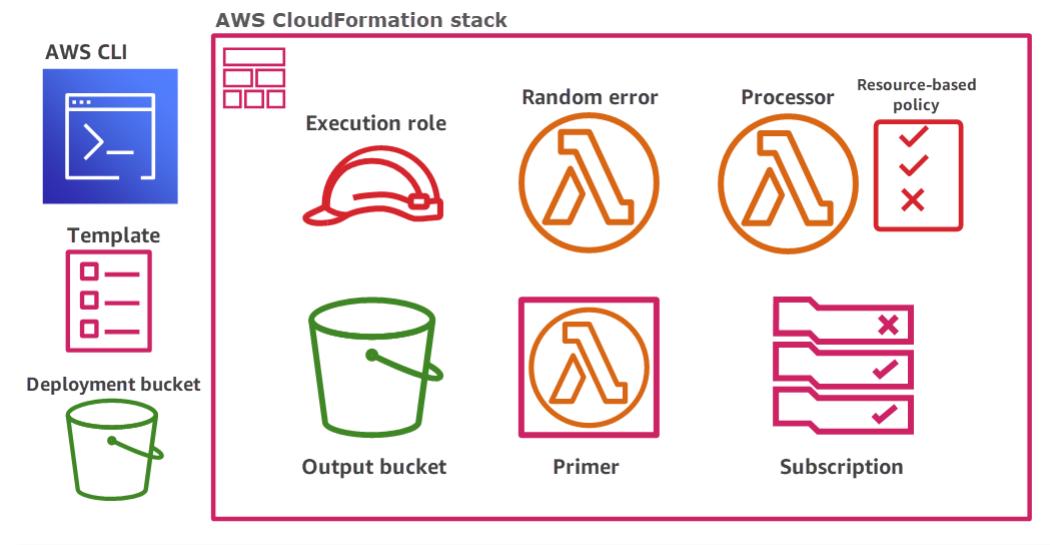
## AWS CloudFormation 템플릿과 추가 리소스

애플리케이션은 두 개의 Node.js 모듈—AWS CloudFormation 템플릿 및 지원하는 셀 스크립트로 구현됩니다. 템플릿은 처리자 함수, 무작위 오류 함수 및 다음과 같은 지원 리소스를 생성합니다.

- 실행 역할 – 다른 AWS 서비스에 대한 액세스 권한을 함수에 부여하는 IAM 역할입니다.
- 프라이머(Primer) 함수 – 무작위 오류 함수를 호출하여 로그 그룹을 생성하는 추가 함수입니다.
- 사용자 지정 리소스 – AWS CloudFormation 사용자 지정 리소스는 배포 중 프라이머 함수를 호출하여 그 그룹이 존재하는지 확인합니다.
- CloudWatch Logs 구독 – ERROR라는 단어가 로깅되면 처리자 함수를 트리거하는 로그 스트림에 대한 구독입니다.
- 리소스 기본 정책 – CloudWatch Logs가 함수를 호출할 수 있도록 허용하는 처리자 함수의 권한 명령문입니다.

- Amazon S3 버킷 – 처리자 함수의 출력 저장 위치입니다.

템플릿은 GitHub의 [error-processor.yaml](#)에서 보십시오.



Lambda와 AWS CloudFormation의 통합 과정에서 발생하는 제한 문제를 해결하기 위해, 이 템플릿은 배포 중 실행하는 추가 함수를 생성합니다. 모든 Lambda 함수에는 함수 실행의 출력을 저장하는 CloudWatch Logs 로그 그룹이 제공됩니다. 하지만 로그 그룹은 함수가 처음 호출되기 전에는 생성되지 않습니다.

로그 그룹의 존재 여부에 따라 구독을 생성하기 위해 애플리케이션은 세 번째 Lambda 함수를 사용하여 무작위 오류 함수를 호출합니다. 템플릿에는 프라이머 함수에 대한 코드가 인라인으로 포함되어 있습니다. AWS CloudFormation 사용자 지정 리소스는 배포 중에 이를 호출합니다. DependsOn 속성은 로그 스트림과 리소스 기반 정책이 구독 중에 생성되도록 보장합니다.

## AWS CodePipeline를 사용하여 Lambda 애플리케이션에 대해 지속적 전송(CD) 파이프라인 구축

AWS CodePipeline를 사용하여 Lambda 애플리케이션에 대해 지속적 전송 파이프라인을 생성할 수 있습니다. CodePipeline은 소스 컨트롤과 빌드 및 배포 리소스를 결합하여 애플리케이션의 소스 코드가 변경될 때마다 실행하는 파이프라인을 생성합니다.

이 자습서에서는 다음 리소스를 생성합니다.

- 리포지토리 – AWS CodeCommit의 Git 리포지토리. 변경 사항을 푸시하면 파이프라인이 소스 코드를 Amazon S3 버킷으로 복사하고 빌드 프로젝트에 전달합니다.
- 빌드 프로젝트 – 파이프라인으로부터 소스 코드를 받아 애플리케이션을 패키징하는 AWS CodeBuild 빌드입니다. 소스에는 빌드 사양과 함께 종속 항목을 설치하고 배포용 AWS Serverless Application Model(AWS SAM) 템플릿을 준비하는 명령이 들어 있습니다.
- 배포 구성 – 파이프라인의 배포 단계는 빌드 출력에서 AWS SAM 템플릿을 가져와 AWS CloudFormation에서 변경 세트를 생성하고 변경 세트를 실행하여 애플리케이션의 AWS CloudFormation 스택을 업데이트하는 일련의 작업을 정의합니다.
- 역할 – 파이프라인과 빌드와 배포는 각각 AWS 리소스를 관리할 수 있는 서비스 역할을 가지고 있습니다. 리소스를 생성하면 콘솔이 파이프라인과 빌드 역할을 생성합니다. AWS CloudFormation이 애플리케이션 스택을 관리할 수 있도록 허용하는 역할을 생성합니다.

파이프라인은 리포지토리의 한 브랜치를 하나의 AWS CloudFormation 스택에 매핑합니다. 동일한 리포지토리에 다른 브랜치를 위한 환경을 추가하기 위해 추가 파이프라인을 생성할 수 있습니다. 파이프라인에 테스트, 스테이징, 수동 승인을 위한 단계를 추가할 수도 있습니다. AWS CodePipeline에 대한 자세한 내용은 [AWS CodePipeline란?](#)을 참조하십시오.

#### 섹션

- [사전 조건 \(p. 120\)](#)
- [AWS CloudFormation 역할 생성 \(p. 120\)](#)
- [리포지토리 설정 \(p. 121\)](#)
- [파이프라인 만들기 \(p. 122\)](#)
- [빌드 단계 역할 업데이트 \(p. 123\)](#)
- [배포 단계 완료 \(p. 123\)](#)

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$)와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

빌드 단계에서 빌드 스크립트는 결과물을 Amazon Simple Storage Service(Amazon S3)에 업로드합니다. 파이프라인에 기존 버킷을 사용하거나 새 버킷을 만들 수 있습니다. AWS CLI를 사용하여 브랜치를 만듭니다.

```
$ aws s3 mb s3://lambda-deployment-artifacts-123456789012
```

## AWS CloudFormation 역할 생성

AWS 리소스에 액세스할 수 있는 권한을 AWS CloudFormation에 부여하는 역할을 만듭니다.

#### AWS CloudFormation 역할을 생성하려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS CloudFormation
  - 권한 – AWSLambdaExecute.
  - 역할 이름 – **cfn-lambda-pipeline**
4. 역할을 엽니다. 권한 탭에서 인라인 정책 추가를 선택합니다.
5. 정책 생성에서 JSON 탭을 선택한 후 다음 정책을 추가합니다.

```
{  
    "Statement": [  
        {
```

```
    "Action": [
        "apigateway:*",
        "codedeploy:*",
        "lambda:*",
        "cloudformation>CreateChangeSet",
        "iam:GetRole",
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:PutRolePolicy",
        "iam:AttachRolePolicy",
        "iam:DeleteRolePolicy",
        "iam:DetachRolePolicy",
        "iam:PassRole",
        "s3:GetObjectVersion",
        "s3:GetBucketVersioning"
    ],
    "Resource": "*",
    "Effect": "Allow"
}
],
"Version": "2012-10-17"
}
```

## 리포지토리 설정

프로젝트 파일을 저장할 AWS CodeCommit 리포지토리를 생성합니다. 자세한 내용은 CodeCommit 사용 설명서의 [설정을](#) 참조하십시오.

리포지토리를 만들려면

1. [개발자 도구 콘솔](#)을 엽니다.
2. 소스에서 리포지토리를 선택합니다.
3. [Create repository]를 선택합니다.
4. 지침에 따라 **lambda-pipeline-repo**라는 리포지토리를 생성하고 복제합니다.

리포지토리 폴더에 다음 파일을 만듭니다.

Example index.js

현재 시간을 반환하는 Lambda 함수입니다.

```
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.yaml

애플리케이션을 정의하는 [SAM 템플릿 \(p. 116\)](#)입니다.

```
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: nodejs8.10  
    CodeUri: ./  
    Events:  
      MyTimeApi:  
        Type: Api  
        Properties:  
          Path: /TimeResource  
          Method: GET
```

#### Example buildspec.yml

필요한 패키지를 설치하고 배포 패키지를 Amazon S3에 업로드하는 [AWS CodeBuild 빌드 사양](#)입니다. 강조 표시된 텍스트를 버킷의 이름으로 바꿉니다.

```
version: 0.2  
phases:  
  install:  
    commands:  
      - npm install time  
      - export BUCKET=lambda-deployment-artifacts-123456789012  
      - aws cloudformation package --template-file template.yaml --s3-bucket $BUCKET --  
        output-template-file outputtemplate.yaml  
  artifacts:  
    type: zip  
    files:  
      - template.yaml  
      - outputtemplate.yaml
```

커밋하고 파일을 CodeCommit에 푸시합니다.

```
~/lambda-pipeline-repo$ git add .  
~/lambda-pipeline-repo$ git commit -m "project files"  
~/lambda-pipeline-repo$ git push
```

## 파이프라인 만들기

애플리케이션을 배포하는 파이프라인을 생성합니다. 이 파이프라인은 리포지토리의 변경을 모니터링하고, AWS CodeBuild 빌드를 실행하여 배포 패키지를 생성하고, AWS CloudFormation을 사용하여 애플리케이션을 배포합니다. 파이프라인 생성 과정에서 AWS CodeBuild 빌드 프로젝트를 생성할 수도 있습니다.

### 파이프라인을 생성하려면

1. [개발자 도구 콘솔](#)을 엽니다.
2. 파이프라인에서 파이프라인을 선택합니다.
3. [Create pipeline]을 선택합니다.
4. 파이프라인 설정을 구성하고 다음을 선택합니다.
  - 파이프라인 이름 – **lambda-pipeline**
  - 서비스 역할 – 새 서비스 역할
  - 아티팩트 스토어 – 기본 위치
5. 소스 단계 설정을 구성하고 다음을 선택합니다.
  - 소스 공급자 – AWS CodeCommit
  - 리포지토리 이름 – lambda-pipeline-repo

- 브랜치 이름 – master
  - 변경 탐지 옵션 – Amazon CloudWatch Events
6. 빌드 공급자에서 AWS CodeBuild를 선택한 후 프로젝트 생성을 선택합니다.
7. 빌드 프로젝트 설정을 구성하고 CodePipeline으로 계속을 선택합니다.
- 프로젝트 이름 – **lambda-pipeline-build**
  - 운영 체제 – Ubuntu
  - 런타임 – Node.js
  - 런타임 버전 – aws/codebuild/nodejs:8.11.0
  - 이미지 버전 – 최신
  - 빌드 사양 이름 – **buildspec.yml**
8. [Next]를 선택합니다.
9. 배포 단계 설정을 구성하고 다음을 선택합니다.
- 배포 공급자 – AWS CloudFormation
  - 작업 모드 – 변경 사항 세트 생성 또는 교체
  - 스택 이름 – lambda-pipeline-stack
  - 변경 세트 이름 – lambda-pipeline-changeset
  - 템플릿 – **BuildArtifact::outputtemplate.yaml**
  - 기능 – CAPABILITY\_IAM
  - 역할 이름 – cfn-lambda-pipeline
10. [Create pipeline]을 선택합니다.

파이프라인을 처음 실행하면 추가 권한이 필요하기 때문에 실패합니다. 다음 단원에서는 빌드 단계를 위해 생성된 역할에 권한을 추가합니다.

## 빌드 단계 역할 업데이트

빌드 단계 중에 AWS CodeBuild는 빌드 출력을 Amazon S3 버켓에 업로드할 수 있는 권한이 필요합니다.

역할을 업데이트 하려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. code-build-lambda-pipeline-service-role을 선택합니다.
3. 정책 연결을 선택합니다.
4. AmazonS3FullAccess를 연결합니다.

## 배포 단계 완료

배포 단계에는 Lambda 애플리케이션을 관리하는 AWS CloudFormation 스택에 대한 변경 세트를 생성하는 작업이 있습니다. 변경 세트를 실행하는 두 번째 작업을 추가하여 배포를 완료합니다.

배포 단계를 업데이트 하려면

1. [개발자 도구 콘솔](#)에서 파이프라인을 엽니다.
2. [Edit]를 선택합니다.
3. 배포 옆의 단계 편집을 선택합니다.
4. 작업 그룹 추가를 선택합니다.
5. 배포 단계 설정을 구성하고 다음을 선택합니다.

- 작업 이름 – **execute-changeset**
  - 배포 공급자 – AWS CloudFormation
  - 입력 아티팩트 – BuildArtifact
  - 작업 모드 – 변경 세트 실행
  - 스택 이름 – lambda-pipeline-stack
  - 변경 세트 이름 – lambda-pipeline-changeset
6. Save를 선택합니다.
  7. Done을 선택합니다.
  8. Save를 선택합니다.
  9. 변경 사항 배포를 선택하여 파이프라인을 실행합니다.

파이프라인이 준비되었습니다. 마스터 브랜치에 변경 사항을 푸시하여 배포를 트리거합니다.

## AWS Lambda 함수 작업의 모범 사례

다음은 AWS Lambda 사용에 대한 권장 모범 사례입니다.

### 주제

- [함수 코드 \(p. 124\)](#)
- [함수 구성 \(p. 125\)](#)
- [경보 및 지표 \(p. 125\)](#)
- [스트림 이벤트 호출 \(p. 126\)](#)
- [비동기 호출 \(p. 126\)](#)
- [Lambda VPC \(p. 126\)](#)

## 함수 코드

- 핵심 로직에서 Lambda 핸들러(진입점)를 분리합니다. 이를 통해 단위 테스트를 수행할 수 있는 더 많은 함수를 만들 수 있습니다. Node.js에서 이는 다음과 같을 수 있습니다.

```
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
}
```

- 실행 컨텍스트 재사용을 활용하여 함수 성능을 향상시킵니다. 코드가 검색하는 외부화된 구성 또는 종속성이 초기 실행 후에 로컬에 저장되고 참조되는지 확인합니다. 모든 호출에서 변수/객체의 재초기화를 제한합니다. 대신 정적 초기화/생성자, 전역/정적 변수 및 singleton을 사용합니다. 이전 호출 중에 설정된 연결(HTTP, 데이터베이스 등)을 유지하고 다시 사용합니다.
- [AWS Lambda 환경 변수 \(p. 39\)](#)를 사용하여 작업 파라미터를 함수에 전달합니다. 예를 들어, Amazon S3 버킷에 기록하는 경우 기록하고 있는 버킷 이름을 하드 코딩하는 대신 환경 변수로 구성합니다.
- 함수 배포 패키지의 종속성을 제어합니다. AWS Lambda 실행 환경에는 Node.js 및 Python 런타임용 AWS SDK와 같은 여러 라이브러리가 포함되어 있습니다(전체 목록은 [AWS Lambda 런타임 \(p. 99\)](#)에서

찾을 수 있음). 최신 기능 및 보안 업데이트를 활성화하려면 가 주기적으로 이러한 라이브러리를 업데이트 해야 합니다. 이러한 업데이트는 함수의 동작에 사소한 변화를 가져올 수 있습니다. 함수에서 사용하는 종속성을 완전히 제어하려면 모든 종속성을 배포 패키지로 패키징하는 것이 좋습니다.

- 배포 패키지 크기를 런타임 필요에 따라 최소화합니다. 이렇게 하면 호출 전에 배포 패키지를 다운로드하고 압축을 풀 때 걸리는 시간이 단축됩니다. Java 또는 .NET Core에서 작성된 함수의 경우 배포 패키지의 일부로 전체 AWS SDK 라이브러리를 업로드하지 마십시오. 대신, 필요한 SDK의 구성 요소를 선택하는 모듈을 선택적으로 활용합니다(예: DynamoDB, Amazon S3 SDK 모듈 및 [Lambda 핵심 라이브러리](#)).
- 종속성 .jar 파일을 별도의 /lib 디렉토리에 배치하여 Java에서 생성된 배포 패키지의 압축을 푸는데 Lambda가 소요하는 시간을 단축합니다. 이는 많은 수의 .class 파일이 있는 단일 jar에 모든 함수 코드를 배치하는 것보다 빠릅니다.
- 종속성의 복잡성을 최소화합니다. [실행 콘텍스트](#) 시작 시 빠르게 로드되는 더 단순한 프레임워크를 선호합니다. 예를 들어 [Spring Framework](#)와 같은 더 복잡한 프레임워크보다는 [Dagger](#) 또는 [Guice](#) 같은 더 단순한 Java 종속성 주입(IoC) 프레임워크를 선호합니다.
- 일부 임의 기준이 충족될 때까지 함수가 자동으로 자체 호출이 되는 리커시브 코드를 Lambda 함수에서 사용하지 않도록 합니다. 리커시브 코드를 사용할 경우, 의도하지 않은 함수 호출이 증가하고 비용이 상승할 수 있습니다. 함수 동시 실행 한도를 설정하는 경우 코드를 업데이트하는 동안 즉시 0으로 설정하여 해당 함수에 대한 모든 호출을 조절합니다.

## 함수 구성

- Lambda 함수를 테스트하는 성능은 최적의 메모리 크기 구성 선택할 때 매우 중요합니다. 메모리 크기가 증가하면 함수에 사용 가능한 상용하는 CPU 사용이 증가합니다. 함수의 메모리 사용은 호출마다 결정되며 [AWS CloudWatch Logs](#)에서 볼 수 있습니다. 매 호출마다 아래와 같이 REPORT: 항목이 만들어집니다.

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Max Memory Used: 필드를 분석함으로써 함수가 더 많은 메모리를 필요로 하는지 또는 함수의 메모리 크기가 과다 프로비저닝되었는지 확인할 수 있습니다.

- Lambda 함수를 로드 테스트하여 최적의 제한 시간 값을 확인합니다. 함수의 실행 시간을 분석하여 함수의 동시성을 기대 이상으로 높일 수 있는 종속성 서비스를 통해 문제를 더 효과적으로 확인하는 것이 중요합니다. 이는 Lambda 함수가 Lambda의 조정을 처리하지 못할 수 있는 리소스에 대한 네트워크 호출을 수행할 때 특히 중요합니다.
- IAM 정책을 설정할 때 가장 제한적인 권한을 사용합니다. Lambda 함수에 필요한 리소스와 작업을 이해하고 실행 역할을 이러한 권한으로 제한합니다. 자세한 내용은 [AWS Lambda 권한 \(p. 9\)](#) 단원을 참조하십시오.
- [AWS Lambda 한도 \(p. 7\)](#) 단원의 내용을 숙지합니다. 페이지 크기, 파일 설명자 및 /tmp 공간은 런타임 리소스 제한을 결정할 때 자주 간과됩니다.
- 더 이상 사용하지 않는 Lambda 함수를 삭제합니다. 삭제하면 사용되지 않는 함수는 배포 패키지 크기 제한에 불필요하게 포함되지 않습니다.
- Amazon Simple Queue Service를 이벤트 소스로 사용 중인 경우, 함수의 예상 실행 시간의 값이 [대기열 제한 시간](#) 값을 넘지 않도록 합니다. 이 사항은 [CreateFunction \(p. 347\)](#) 및 [UpdateFunctionConfiguration \(p. 455\)](#)에 모두 적용됩니다.
  - CreateFunction의 경우 AWS Lambda가 함수 생성 프로세스에 실패합니다.
  - UpdateFunctionConfiguration의 경우, 함수가 중복 호출될 수 있습니다.

## 경보 및 지표

- Lambda 함수 코드 내에서 지표를 생성하거나 업데이트하는 대신 [AWS Lambda 지표 \(p. 223\)](#) 및 [CloudWatch 경보](#)를 사용합니다. 이는 함수의 상태를 파악하는 훨씬 더 효율적인 방법이며, 개발 프로세스

초기에 문제를 파악할 수 있습니다. 예를 들어, 함수 코드로 인한 병목 현상이나 자연 시간을 해결하기 위해 예상되는 함수 실행 시간을 기준으로 경보를 구성할 수 있습니다.

- 로깅 라이브러리 및 [AWS Lambda 지표 및 차원](#)을 사용하여 앱 오류를 파악합니다(ERR, ERROR, WARNING 등).

## 스트림 이벤트 호출

- 서로 다른 배치 및 레코드 크기로 테스트하여 각 이벤트 소스의 폴링 빈도를 조정하고 해당 함수가 얼마나 빨리 작업을 완료할 수 있는지 확인합니다. [BatchSize](#)는 각 호출에서 함수로 보낼 수 있는 최대 레코드 수를 제어합니다. 배치 크기가 클수록 더 큰 레코드 세트에서 호출 오버헤드를 더 효율적으로 수용하여 처리량을 늘릴 수 있습니다.

### Note

대기하는 대신, 처리할 레코드가 충분하지 않은 경우 스트림 처리 함수는 더 적은 수의 레코드로 호출됩니다.

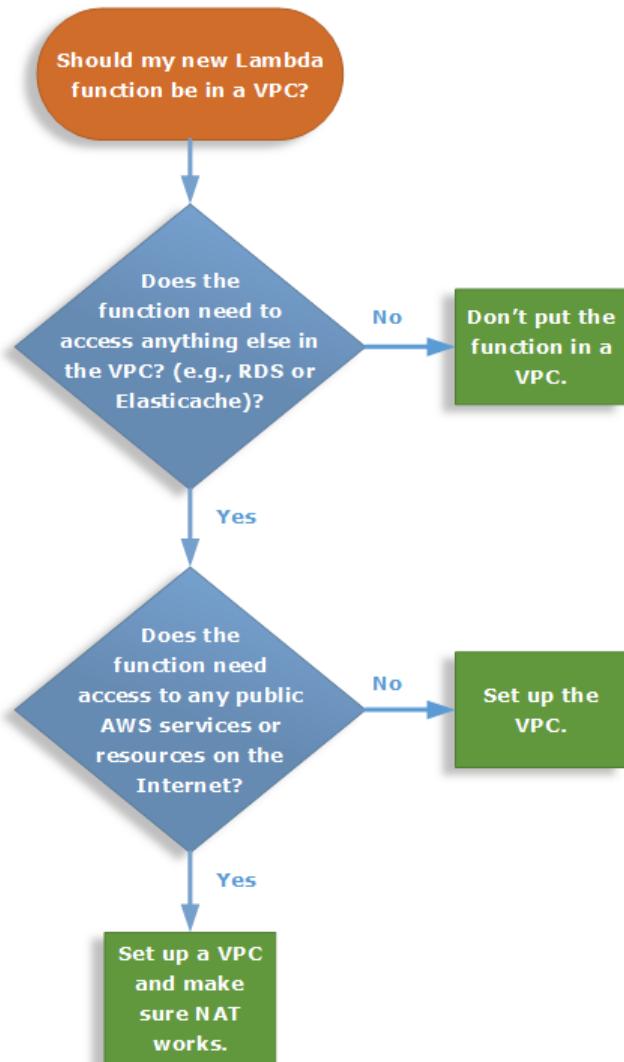
- 샤드를 추가하여 Kinesis 스트림 처리량을 늘립니다. Kinesis 스트림은 1개 이상의 샤드로 구성되며, Lambda는 최대 한 개의 동시 호출로 각 샤드를 폴링합니다. 예를 들어 스트림에 100개의 활성 샤드가 있으면 최대 100개의 함수 호출이 동시에 실행됩니다. 샤드 수를 늘리면 Lambda 함수의 최대 동시 호출 수가 증가하고 Kinesis 스트림 처리량이 증가할 수 있습니다. Kinesis 스트림에서 샤드 수를 늘리는 경우 데이터에 대해 적절한 파티션 키를 선택하여([파티션 키](#) 참조) 관련 레코드가 동일한 샤드에서 끝나며 데이터가 잘 분산되도록 해야 합니다.
- IteratorAge에 대해 [Amazon CloudWatch](#)를 사용하여 Kinesis 스트림이 처리 중인지 확인합니다. 예를 들어 최대 값을 30000(30초)으로 설정하여 CloudWatch 경보를 구성합니다.

## 비동기 호출

- [AWS Lambda](#) 함수 배달 못한 편지 대기열 (p. 86)를 생성하고 사용 하여 비동기 함수 오류를 해결하고 재생합니다.

## Lambda VPC

- 다음 다이어그램은 VPC(Virtual Private Cloud)를 사용해야 하는지 여부에 대한 판단 트리를 안내합니다.



- 반드시 필요한 경우가 아닌 한 VPC에 Lambda 함수를 배치하지 않습니다. 프라이빗 Amazon 관계형 데이터베이스 인스턴스처럼 공개할 수 없는 리소스에 액세스할 때 이 기능을 사용하는 것 이외에 아무런 도움이 되지 않습니다. 같은 서비스는 액세스 정책이 있는 IAM을 통해 보안이 유지될 수 있으므로, 엔드포인트를 공개해도 안전하며 VPC에서 함수를 실행하여 보안을 유지할 필요가 없습니다.
- Lambda는 VPC에 탄력적 네트워크 인터페이스(ENI)를 생성해 내부 리소스에 액세스합니다. 동시성 증가를 요청하기 전에 ENI 용량(이에 대한 공식은 [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 \(p. 66\)](#)에서 찾을 수 있음) 및 IP 주소 공간이 충분한지 확인해야 합니다. ENI 용량이 충분하지 않은 경우 증가를 요청해야 합니다. IP 주소 공간이 충분하지 않은 경우 더 큰 서브넷을 만들어야 할 수 있습니다.
- VPC에서 전용 Lambda 서브넷 만들기:
  - 이렇게 하면 다른 프라이빗/퍼블릭 서브넷을 변경하지 않고 NAT 게이트웨이 트래픽에 대한 사용자 지정 라우팅 테이블을 보다 쉽게 적용할 수 있습니다. 자세한 내용은 [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 \(p. 66\)](#) 단원을 참조하십시오.
  - 또한 다른 리소스와 공유하지 않고 Lambda에 주소 공간을 할당할 수도 있습니다.

# 다른 서비스와 함께 AWS Lambda 사용

AWS Lambda는 함수를 호출하기 위해 다른 AWS 서비스와 통합됩니다. 리소스 수명 주기 이벤트에 대응하여 함수를 호출하거나, 수신 HTTP 요청에 응답하거나, 대기열에서 이벤트를 사용하거나, [일정에 따라 실행](#) (p. 153)하도록 트리거를 구성할 수 있습니다.

Lambda와 통합되는 각 서비스는 JSON 형식의 함수에 데이터를 이벤트로 전송합니다. 이벤트 문서의 구조는 각 이벤트 유형에 따라 다르며, 함수를 트리거한 리소스 또는 요청에 대한 데이터가 들어 있습니다. Lambda 런타임은 이벤트를 객체로 변환한 후 함수로 전달합니다.

다음 예제는 `/lambda?query=1234ABCD`에 대한 GET 요청을 나타내는 [Application Load Balancer](#) (p. 129)의 테스트 이벤트를 보여줍니다.

Example Application Load Balancer의 이벤트

```
{
  "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=1-5c536348-3d683b8b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http",
    "x-imforwards": "20"
  },
  "body": "",
  "isBase64Encoded": false
}
```

## Note

Lambda 런타임은 이벤트 문서를 객체로 변환한 후 [함수 핸들러](#) (p. 31)에 전달합니다. 컴파일된 언어의 경우 Lambda는 라이브러리의 이벤트 유형에 대한 정의를 제공합니다. 자세한 내용은 다음 주제 단원을 참조하십시오.

- [Java를 사용하여 Lambda 함수 빌드](#) (p. 257)
- [Go를 사용하여 Lambda 함수 빌드](#) (p. 283)

- C#을 사용하여 Lambda 함수 빌드 (p. 295)

대기열 또는 데이터 스트림을 생성하는 서비스의 경우, Lambda에서 [이벤트 소스 매핑 \(p. 80\)](#)을 생성하고 실행 역할 (p. 9)에서, 다른 서비스에 액세스할 수 있는 권한을 Lambda에 부여합니다. Lambda는 다른 서비스로부터 데이터를 읽고, 이벤트를 생성하고, 함수를 호출합니다.

#### Lambda가 이벤트를 읽는 서비스

- Amazon Kinesis (p. 175)
- Amazon DynamoDB (p. 165)
- Amazon Simple Queue Service (p. 211)

다른 서비스는 해당 함수를 직접 호출합니다. 함수의 [리소스 기반 정책 \(p. 10\)](#)에서 다른 서비스에 권한을 부여한 후, 다른 서비스가 이벤트를 생성하고 함수를 호출하도록 구성합니다. 서비스에 따라 호출은 동기적으로 또는 비동기적으로 수행될 수 있습니다. 동기 호출의 경우 다른 서비스는 함수로부터 응답을 기다리고 [오류 시 재시도 \(p. 83\)](#)할 수 있습니다.

#### Lambda 함수를 동기적으로 호출하는 서비스

- Elastic Load Balancing(Application Load Balancer) (p. 129)
- Amazon Cognito (p. 163)
- Amazon Lex (p. 188)
- Amazon Alexa (p. 131)
- Amazon API Gateway (p. 131)
- Amazon CloudFront(Lambda@Edge) (p. 161)
- Amazon Kinesis Data Firehose (p. 187)

비동기 호출의 경우 Lambda는 이벤트를 함수에 전달하기 전에 대기열에 추가합니다. 다른 서비스는 이벤트가 대기열에 추가되는 즉시 성공 결과를 받고, 그 후에 무슨 일이 일어나는지 알지 못합니다. 오류가 발생할 경우 Lambda는 [재시도 \(p. 83\)](#)를 처리하고, 구성된 [배달 못한 편지 대기열 \(p. 86\)](#)로 실패한 이벤트를 보냅니다.

#### Lambda 함수를 비동기적으로 호출하는 서비스

- Amazon Simple Storage Service (p. 188)
- Amazon Simple Notification Service (p. 205)
- Amazon Simple Email Service (p. 203)
- AWS CloudFormation (p. 159)
- Amazon CloudWatch Logs (p. 158)
- Amazon CloudWatch Events (p. 153)
- AWS CodeCommit (p. 163)
- AWS Config (p. 164)

각 서비스에 대한 자세한 내용과 함수를 테스트하는 데 사용할 수 있는 이벤트 예제에 대한 자세한 내용은 [이장의 단원](#)을 참조하십시오.

## Application Load Balancer에서 AWS Lambda 사용

Lambda 함수를 사용하면 Application Load Balancer의 요청을 처리할 수 있습니다. Elastic Load Balancing은 Application Load Balancer에 대한 대상으로 Lambda 함수를 지원합니다. 로드 밸런서 규칙을 사용하여 경

로 또는 헤더 값을 기반으로 HTTP 요청을 함수에 라우팅합니다. 요청을 처리하고 Lambda 함수에서 HTTP 응답을 반환하십시오.

Elastic Load Balancing은 요청 본문 및 메타데이터가 포함된 이벤트와 동기적으로 Lambda 함수를 호출합니다.

### Example Application Load Balancer 요청 이벤트

```
{  
    "requestContext": {  
        "elb": {  
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambda-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  
}
```

함수는 이벤트를 처리하고 JSON의 로드 밸런서에 대한 응답을 반환합니다. Elastic Load Balancing은 이 응답을 HTTP로 변환하여 사용자에게 반환합니다.

### Example 응답 형식

```
{  
    "statusCode": 200,  
    "statusDescription": "HTTP OK",  
    "isBase64Encoded": false,  
    "headers": {  
        "Content-Type": "text/html"  
    },  
    "body": "<h1>Hello from Lambda!</h1>"  
}
```

Application Load Balancer를 함수 트리거로 구성하려면 함수 실행 권한을 Elastic Load Balancing에 부여하고 해당 요청을 함수로 라우팅하는 대상 그룹을 만든 다음, 대상 그룹에 요청을 보내는 로드 밸런서에 하나의 규칙을 추가하십시오.

`add-permission` 명령을 사용하여 권한 설명문을 함수의 리소스 기반 정책에 추가하십시오.

```
$ 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\"}"
}
```

Application Load Balancer리스너 및 대상 그룹 구성에 관한 지침은 Application Load Balancer 사용 설명서에서 [대상으로서 Lambda 함수](#) 단원을 참조하십시오.

## Alexa에서 AWS Lambda 사용

Lambda 함수를 사용하여 Amazon Echo의 음성 도우미인 Alexa에게 새로운 기술을 제공하는 서비스를 구축할 수 있습니다. Alexa Skills Kit는 함수로 실행되는 자체 서비스를 통해 이러한 새로운 기술을 개발할 수 있는 API, 도구 및 설명서를 제공합니다. Amazon Echo 사용자는 Alexa에게 질문을 하거나 요청을 하여 새로운 기술에 액세스할 수 있습니다.

Alexa Skills Kit는 GitHub에서 구할 수 있습니다.

- [Node.js용 Alexa Skills Kit SDK](#)
- [Java용 Alexa Skills Kit SDK](#)

### Example Alexa Smart Home Event

```
{
  "header": {
    "payloadVersion": "1",
    "namespace": "Control",
    "name": "SwitchOnOffRequest"
  },
  "payload": {
    "switchControlAction": "TURN_ON",
    "appliance": {
      "additionalApplianceDetails": {
        "key2": "value2",
        "key1": "value1"
      },
      "applianceId": "sampleId"
    },
    "accessToken": "sampleAccessToken"
  }
}
```

자세한 내용은 [Alexa Skills Kit 시작하기](#)를 참조하십시오.

## Using AWS Lambda with Amazon API Gateway

HTTPS를 통해 AWS Lambda 함수를 호출할 수 있습니다. [Amazon API Gateway](#)를 사용하여 사용자 정의 REST API 및 엔드포인트를 정의한 다음 GET 및 PUT과 같은 개별 메서드를 특정 Lambda 함수에 매핑하여 이를 수행할 수 있습니다. 또는 ANY라는 특수 메서드를 추가하여 지원되는 모든 메서드(GET, POST, PATCH, DELETE)를 사용자의 Lambda 함수에 매핑할 수 있습니다. API 엔드포인트에 HTTPS 요청을 보내면 Amazon API Gateway 서비스가 해당 Lambda 함수를 호출합니다. ANY 메서드에 대한 자세한 내용은 [Lambda 및 API 게이트웨이를 사용하여 단순 마이크로서비스 생성](#) (p. 144) 단원을 참조하십시오.

## Example Amazon API Gateway 메시지 이벤트

```
{  
    "path": "/test/hello",  
    "headers": {  
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",  
        "Accept-Encoding": "gzip, deflate, lzma, sdch, br",  
        "Accept-Language": "en-US,en;q=0.8",  
        "CloudFront-Forwarded-Proto": "https",  
        "CloudFront-Is-Desktop-Viewer": "true",  
        "CloudFront-Is-Mobile-Viewer": "false",  
        "CloudFront-Is-SmartTV-Viewer": "false",  
        "CloudFront-Is-Tablet-Viewer": "false",  
        "CloudFront-Viewer-Country": "US",  
        "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",  
        "Upgrade-Insecure-Requests": "1",  
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",  
        "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",  
        "X-Amz-Cf-Id": "nBsWBOrSHMgnaROZJK1wGCZ9PcRcSpq_oSXZNQwQ100TzL4cimZo3g==",  
        "X-Forwarded-For": "192.168.100.1, 192.168.1.1",  
        "X-Forwarded-Port": "443",  
        "X-Forwarded-Proto": "https"  
    },  
    "pathParameters": {  
        "proxy": "hello"  
    },  
    "requestContext": {  
        "accountId": "123456789012",  
        "resourceId": "us4z18",  
        "stage": "test",  
        "requestId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9",  
        "identity": {  
            "cognitoIdentityPoolId": "",  
            "accountId": "",  
            "cognitoIdentityId": "",  
            "caller": "",  
            "apiKey": "",  
            "sourceIp": "192.168.100.1",  
            "cognitoAuthenticationType": "",  
            "cognitoAuthenticationProvider": "",  
            "userArn": "",  
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",  
            "user": ""  
        },  
        "resourcePath": "/{proxy+}",  
        "httpMethod": "GET",  
        "apiId": "wt6mne2s9k"  
    },  
    "resource": "/{proxy+}",  
    "httpMethod": "GET",  
    "queryStringParameters": {  
        "name": "me"  
    },  
    "stageVariables": {  
        "stageVarName": "stageVarValue"  
    }  
}
```

또한 Amazon API Gateway는 애플리케이션 사용자와 앱 로직 사이에 다음 항목을 활성화하는 계층을 추가합니다.

- 개별 사용자나 요청을 제한할 수 있습니다.

- DDoS(분산 서비스 거부) 공격을 차단합니다.
- Lambda 함수에서 응답을 캐시에 저장할 수 있도록 캐싱 계층을 제공합니다.

Amazon API Gateway 및 AWS Lambda 통합이 작동하는 방법에 대해 다음 사항을 참고합니다.

- 이벤트 푸시 모델 – Amazon API Gateway가 Lambda 함수에 파라미터로 요청 본문의 데이터를 전달하여 Lambda 함수를 호출하는 모델입니다([AWS Lambda 이벤트 소스 매핑 \(p. 80\)](#) 참조).
- 동기식 호출 – Amazon API Gateway는 RequestResponse 를 호출 유형으로 지정하여 Lambda 함수를 호출하고 실시간으로 응답을 얻을 수 있습니다. 호출 유형에 대한 내용은 [호출 유형 \(p. 80\)](#)을 참조하십시오.
- 이벤트 구조 – Lambda 함수가 수신하는 이벤트는 Amazon API Gateway가 수신하는 HTTPS 요청의 본문이며 Lambda 함수는 특정 이벤트 유형을 처리하기 위해 작성된 사용자 지정 코드입니다.

종합적인 경험을 설정하는 데 사용하는 권한 정책은 두 가지 유형이 있습니다.

- Lambda 함수에 대한 권한 – Lambda 함수를 호출하는 것과 관계없이 AWS Lambda는 Lambda 함수를 만들 때 지정하는 IAM 역할(실행 역할)을 가정하여 함수를 실행합니다. 이 역할과 연결된 권한 정책을 사용하여 함수에 필요한 권한을 부여합니다. 예를 들어, Lambda 함수가 객체를 읽어야 하는 경우, 권한 정책에서 관련 Amazon S3 작업에 대한 권한을 부여합니다. 자세한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.
- Amazon API Gateway가 Lambda 함수를 호출할 수 있는 권한 – Amazon API Gateway는 사용자의 허가 없이 Lambda 함수를 호출할 수 없습니다. 사용자는 함수와 연결된 권한 정책을 통해 이 권한을 부여합니다.

## 자습서: Amazon API Gateway과 함께 AWS Lambda 사용

이 예제에서는 Amazon API Gateway를 사용하여 간단한 API를 생성합니다. Amazon API Gateway는 리소스와 메서드 모음입니다. 이 자습서에서는 이에 대해 하나의 리소스(DynamoDBManager)를 생성하고 하나의 메서드(POST)를 정의합니다. 해당 메서드는 Lambda 함수 기반입니다(LambdaFunctionOverHttps). 즉, HTTPS 엔드포인트를 통해 API를 호출하면 Amazon API Gateway가 Lambda 함수를 호출합니다.

DynamoDBManager 리소스에 대한 POST 메서드는 다음 DynamoDB 작업을 지원합니다.

- 항목을 생성, 업데이트 및 삭제합니다.
- 항목을 읽습니다.
- 항목을 스캔합니다.
- 테스트에 사용할 수 있고 DynamoDB와 관련이 없는 다른 작업(echo, ping)

POST 요청에서 보내는 요청 페이로드는 DynamoDB 작업을 식별하고 필요한 데이터를 제공합니다. 예:

- 다음은 DynamoDB 항목 만들기 작업에 대한 샘플 요청 페이로드입니다.

```
{  
    "operation": "create",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Item": {  
            "id": "1",  
            "name": "Bob"  
        }  
    }  
}
```

```
}
```

- 다음은 DynamoDB 항목 읽기 작업에 대한 샘플 요청 페이로드입니다.

```
{  
    "operation": "read",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Key": {  
            "id": "1"  
        }  
    }  
}
```

- 다음은 echo 작업에 대한 샘플 요청 페이로드입니다. 요청 본문에 다음 데이터를 사용하여 HTTP POST 요청을 엔드포인트로 보냅니다.

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  

```

#### Note

API 게이트웨이는 다음과 같은 고급 기능을 제공합니다.

- 전체 요청 전달 – Lambda 함수는 요청 본문 대신 전체 HTTP 요청을 수신하고 AWS\_PROXY 통합 유형을 사용하여 응답 본문 대신 HTTP 응답을 설정할 수 있습니다.
- catch-all 메서드 – ANY catch-all 메서드를 사용하여 API 리소스의 모든 메서드를 단일 매핑을 통해 단일 Lambda 함수에 매핑합니다.
- Catch-all 리소스 – 새 경로 파라미터({proxy+})를 사용하여 추가 구성 없이 리소스의 모든 하위 경로를 Lambda 함수에 매핑합니다.

API Gateway 기능에 대한 자세한 정보는 [프록시 리소스에 대한 프록시 통합 구성](#)을 참조하십시오.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

## 실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 개체 – Lambda.
  - 역할 이름 – **lambda-apigateway-role**.
  - 권한 – DynamoDB 및 CloudWatch Logs에 대한 권한을 포함하는 사용자 지정 정책입니다.

```
{  
    "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"  
        }  
    ]  
}
```

이 사용자 지정 정책에는 함수가 DynamoDB에 항목을 쓰고 로그를 업로드하는 데 필요한 권한이 있습니다. 나중에 사용할 수 있도록 역할의 Amazon 리소스 이름(ARN)을 적어둡니다.

## 함수 만들기

다음은 API 게이트웨이 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 함수 코드 \(p. 141\)](#) 단원을 참조하십시오.

### 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:  
 * - event (required)  
 * - context (optional)  
 * - callback (optional)  
 */
```

```
/*
 * - 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}');
    }
};
```

### 함수를 만들려면

- 샘플 코드를 index.js 파일에 복사합니다.
- 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

- create-function 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

### Lambda 함수 테스트

샘플 이벤트 데이터를 사용하여 수동으로 함수를 호출합니다. 콘솔 UI는 실행 요약, 코드로 작성된 로그 및 함수에서 반환한 결과를 포함하여 실행 결과를 검토하기 위한 사용자 친화적 인터페이스를 제공하므로, 콘솔을 사용하여 함수를 호출하는 것이 좋습니다(콘솔은 항상 동기식 실행을 수행하기 때문에 — 호출 유형을 사용하여 Lambda 함수를 호출함).

## Lambda 함수를 테스트하려면

1. 다음 JSON을 파일에 복사하고 `input.txt`로 저장합니다.

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

2. 다음 `invoke` 명령을 실행합니다.

```
$ aws lambda invoke --function-name LambdaFunctionOverHttps \  
--payload fileb://input.txt outfile.txt
```

## Amazon API Gateway을 사용하여 API 생성

이 단계에서는 Amazon API Gateway를 사용하여 생성한 API의 메서드와 Lambda 함수를 연결하고 종합적 경험을 테스트합니다. 즉, HTTP 요청이 API 메서드로 전송되면 Amazon API Gateway가 Lambda 함수를 호출합니다.

먼저, 하나의 리소스(DynamoDBManager) 및 하나의 메서드(POST)를 통해 Amazon API Gateway를 사용하여 API(DynamoDBOperations)를 생성합니다. POST 메서드를 Lambda 함수와 연결합니다. 그런 다음 종합적 경험을 테스트합니다.

### API 생성

다음 `create-rest-api` 명령을 실행하여 이 자습서용 DynamoDBOperations API를 생성합니다.

```
$ aws apigateway create-rest-api --name DynamoDBOperations  
{  
    "id": "bs8fqo6bp0",  
    "name": "DynamoDBOperations",  
    "createdDate": 1539803980,  
    "apiKeySource": "HEADER",  
    "endpointConfiguration": {  
        "types": [  
            "EDGE"  
        ]  
    }  
}
```

이후 명령에 사용하기 위해 API ID를 저장합니다. 또한 API 루트 리소스의 ID가 필요합니다. ID를 얻으려면 `get-resources` 명령을 실행합니다.

```
$ API=bs8fqo6bp0  
$ aws apigateway get-resources --rest-api-id $API  
{  
    "items": [  
        {  
            "path": "/",  
            "id": "e8kitthgdb"  
        }  
    ]  
}
```

이 시점에서는 루트 리소스만 있지만 다음 단계에서 리소스를 추가합니다.

## API에서 리소스 생성

이전 단원에서 만든 API에서 다음 `create-resource` 명령을 실행하여 리소스(DynamoDBManager)를 생성합니다.

```
$ aws apigateway create-resource --rest-api-id $API --path-part DynamoDBManager \
--parent-id e8kitthgdb
{
    "path": "/DynamoDBManager",
    "pathPart": "DynamoDBManager",
    "id": "resource-id",
    "parentId": "e8kitthgdb"
}
```

응답의 ID를 기록해 둡니다. 생성한 DynamoDBManager 리소스의 ID입니다.

## 리소스에 POST 메서드 생성

다음 `put-method` 명령을 실행하여 API의 DynamoDBManager 리소스에 POST 메서드를 생성합니다.

```
$ 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"
}
```

--authorization-type 파라미터에 대해 NONE을 지정합니다. 이는 이 메서드에 대해 인증되지 않은 요청이 지원된다는 점을 의미합니다. 테스팅 환경에서는 상관없지만 프로덕션 환경에서는 키 기반 인증이나 역할 기반 인증을 사용해야 합니다.

## POST 메서드의 대상으로 Lambda 함수 설정

다음 명령을 실행하여 POST 메서드에 대한 통합 응답으로 Lambda 함수를 설정합니다. POST 메서드 앤드포인트에 대해 HTTP 요청을 실행하면 Amazon API Gateway가 이 메서드를 호출합니다. 이 명령과 다른 명령은 해당 계정 ID와 리전을 포함하는 ARN을 사용합니다. 이를 변수에 저장합니다. 계정 ID는 함수를 생성하는 데 사용한 역할 ARN에서 확인할 수 있습니다.

```
$ 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-method는 API 게이트웨이가 AWS Lambda와 통신하는 데 사용하는 메서드입니다. --uri는 Amazon API Gateway가 요청을 보낼 수 있는 앤드포인트의 고유 ID입니다.

다음과 같이 POST 메서드 응답 및 통합 응답의 content-type을 JSON에 설정합니다.

- 다음 명령을 실행하여 POST 메서드 응답을 JSON에 설정합니다. 이는 API 메서드가 반환하는 응답 유형입니다.

```
$ 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"
    }
}
```

- 다음 명령을 실행하여 POST 메서드 통합 응답을 JSON에 설정합니다. 이는 Lambda 함수가 반환하는 응답 유형입니다.

```
$ 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
    }
}
```

## API 배포

이 단계에서는 생성한 API를 prod라는 단계에 배포합니다.

```
$ aws apigateway create-deployment --rest-api-id $API --stage-name prod
{
    "id": "20vgpsz",
    "createdDate": 1539820012
}
```

## API에 호출 권한 부여

Amazon API Gateway를 사용하여 생성한 API를 배포했으므로 이제 테스트할 수 있습니다. 우선, HTTP 요청을 POST 메서드로 보낼 때 Amazon API Gateway가 Lambda 함수를 호출할 수 있도록 권한을 추가해야 합니다.

이렇게 하려면 Lambda 함수와 연결된 정책에 권한을 추가해야 합니다. 다음 add-permission AWS Lambda 명령을 사용하여 Amazon API Gateway 서비스 보안 주체(apigateway.amazonaws.com)에게 Lambda 함수(LambdaFunctionOverHttps)를 호출할 수 있는 권한을 부여합니다.

```
$ 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\"}}}"
}
```

테스트를 활성화하려면 이 권한을 부여해야 합니다(Amazon API Gateway로 이동하고 테스트를 선택해서 API 메서드를 테스트하는 경우 이 권한이 필요함). --source-arn은 와일드카드 문자(\*)를 단계 값(테스팅만 나타냄)으로 지정합니다. 이를 통해 API를 배포하지 않고도 테스트할 수 있습니다.

이제 같은 명령을 다시 실행하지만 이번에는 배포된 API 권한에 Lambda 함수를 호출하는 권한을 부여합니다.

```
$ 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\"}}}"
}
```

배포된 API가 Lambda 함수를 호출할 수 있는 권한을 갖도록 이 권한을 부여합니다. --source-arn은 API를 배포할 때 사용한 단계 이름인 `prod`를 지정합니다.

## Amazon DynamoDB 테이블 생성

Lambda 함수가 사용하는 DynamoDB 테이블을 생성합니다.

DynamoDB 테이블을 만들려면

1. [DynamoDB 콘솔](#)을 엽니다.
2. 테이블 만들기를 선택합니다.
3. 다음 설정을 사용하여 테이블을 생성합니다.
  - 테이블 이름 – `lambda-apigateway`
  - 기본 키 – `id`(문자열)
4. 생성을 선택합니다.

## HTTP 요청을 사용하여 함수 트리거

이 단계에서는 HTTP 요청을 POST 메서드 앤드포인트로 보낼 준비가 되었습니다. Curl 또는 Amazon API Gateway에서 제공하는 메서드(`test-invoke-method`)를 사용할 수 있습니다.

Amazon API Gateway CLI 명령을 사용하여 HTTP POST 요청을 리소스(DynamoDBManager) 앤드포인트로 보낼 수 있습니다. Amazon API Gateway를 배포했으므로 Curl을 사용하여 동일한 작업에 대한 메서드를 호출할 수 있습니다.

Lambda 함수는 `create` 작업을 지원하여 DynamoDB 테이블에 항목을 생성합니다. 이 작업을 요청하려면 다음 JSON을 사용합니다.

Example `create-item.json`

```
{
    "operation": "create",
    "tableName": "lambda-apigateway",
    "payload": {
        "Item": {
            "id": "1234ABCD",
            "number": 5
        }
    }
}
```

```
}
```

테스트 입력을 `create-item.json` 파일에 저장합니다. `test-invoke-method` Amazon API Gateway 명령을 실행하여 HTTP POST 메서드 요청을 리소스(DynamoDBManager) 엔드포인트에 보냅니다.

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://create-item.json
```

또는 다음 Curl 명령을 사용할 수 있습니다.

```
$ curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-apigateway\",
\"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}" https://$API.execute-api.
$REGION.amazonaws.com/prod/DynamoDBManager
```

Lambda 함수가 지원하는 echo 작업에 대한 요청을 보내려면 다음 요청 페이로드를 사용할 수 있습니다.

Example `echo.json`

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
}
```

테스트 입력을 `echo.json` 파일에 저장합니다. `test-invoke-method` Amazon API Gateway CLI 명령을 실행하여 요청 본문의 이전 JSON을 사용하여 리소스(DynamoDBManager) 엔드포인트에 HTTP POST 메서드 요청을 보냅니다.

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://echo.json
```

또는 다음 Curl 명령을 사용할 수 있습니다.

```
$ curl -X POST -d "{\"operation\":\"echo\",\"payload\":{\"somekey1\":\"somevalue1\",
\"somekey2\":\"somevalue2\"}}" https://$API.execute-api.$REGION.amazonaws.com/prod/
DynamoDBManager
```

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

주제

- [Node.js \(p. 141\)](#)
- [Python 3 \(p. 142\)](#)
- [Go \(p. 143\)](#)

## Node.js

다음 예제는 API 게이트웨이의 메시지를 처리하고, 요청 방법에 따라 DynamoDB 문서를 관리합니다.

### 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}');
    }
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Python 3

다음 예제는 API 게이트웨이의 메시지를 처리하고, 요청 방법에 따라 DynamoDB 문서를 관리합니다.

### 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))
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Go

다음 예제는 API 게이트웨이의 메시지를 처리하고, 요청에 대한 정보를 로깅합니다.

### 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
}
```

go build를 사용하여 실행 파일을 빌드하고 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지 \(Go\) \(p. 283\)](#) 단원을 참조하십시오.

## Lambda 및 API 게이트웨이를 사용하여 단순 마이크로 서비스 생성

이 자습서에서는 Lambda 콘솔을 사용하여 Lambda 함수를 생성하고 Amazon API Gateway 엔드포인트를 사용하여 해당 함수를 트리거합니다. 어떤 메서드(GET, POST, PATCH 등)로도 엔드포인트를 호출하여 Lambda 함수를 트리거할 수 있습니다. 엔드포인트가 호출되면 전체 요청이 Lambda 함수로 패스스루됩니다. 함수 작업은 엔드포인트를 호출한 메서드에 따라 다릅니다.

- DELETE: DynamoDB 테이블에서 항목 삭제
- GET: 테이블을 스캔하고 모든 항목들을 반환
- POST: 항목 생성
- PUT: 항목 업데이트

### Amazon API Gateway을 사용하여 API 생성

이 단원의 단계에 따라 트리거를 위한 새로운 Lambda 함수와 API 게이트웨이 엔드포인트를 생성합니다.

API를 생성하려면

1. AWS Management Console에 로그인하고 AWS Lambda 콘솔을 엽니다.
2. Create Lambda function을 선택합니다.
3. 블루프린트를 선택합니다.
4. 검색줄에 **microservice**를 입력합니다. microservice-http-endpoint 블루프린트를 선택한 후 구성을 선택합니다.
5. 다음 설정을 구성합니다.
  - 이름 – **lambda-microservice**.
  - 역할 – 1개 이상의 템플릿으로 새 역할 생성
  - 역할 이름 – **lambda-apigateway-role**.
  - 정책 템플릿 – 단순 마이크로서비스 권한
  - API – 새 API 생성
  - 보안 – 개방

함수 생성을 선택합니다.

마법사를 완료하고 함수를 생성하면 Lambda가 선택한 API 이름 아래에 `lambda-microservice`라는 이름의 프록시 리소스를 생성합니다. 프록시 리소스에 대한 자세한 정보는 [프록시 리소스에 대한 프록시 통합 구성](#)을 참조하십시오.

프록시 리소스는 AWS\_PROXY 통합 유형과 catch-all 메서드 ANY를 포함하고 있습니다. AWS\_PROXY 통합 유형은 기본 매핑 템플릿을 적용하여 Lambda 함수로 전체 요청을 패스스루하고 Lambda 함수에서 HTTP 응답으로 출력을 변환합니다. ANY 메서드는 GET, POST, PATCH, DELETE 등 지원되는 모든 메서드에 대해 동일한 통합 설정을 정의합니다.

### HTTPS 요청 전송 테스트

이 단계에서는 콘솔을 사용하여 Lambda 함수를 테스트합니다. 뿐만 아니라 curl 명령을 실행하여 전체 경험을 테스트할 수 있습니다. 즉, HTTPS 요청을 API 메서드로 전송하여 Amazon API Gateway가 Lambda 함수를 호출하도록 합니다. 단계를 완료하려면 테이블을 생성하고 "MyTable"이라고 이름을 명명했는지 확인합니다. 자세한 내용은 [스트림을 활성화하여 DynamoDB 테이블 생성 \(p. 170\)](#) 단원을 참조하십시오.

### API를 테스트하려면

1. MyLambdaMicroService 함수가 콘솔에 여전히 열려 있으면 [Actions] 탭을 선택한 다음, [Configure test event]를 선택합니다.
2. 기존 텍스트를 다음으로 바꿉니다.

```
{  
    "httpMethod": "GET",  
    "queryStringParameters": {  
        "TableName": "MyTable"  
    }  
}
```

3. 위의 텍스트를 입력한 후 [Save and test]를 선택합니다.

## API 게이트웨이 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 \(p. 133\)](#)의 Lambda 애플리케이션을 위한 샘플 AWS SAM 템플릿입니다. 아래 텍스트를 .yaml 파일로 복사하고 이전에 만든 ZIP 패키지 옆에 저장합니다. Handler 및 Runtime 파라미터 값은 이전 단원에서 함수를 생성할 때 사용한 것과 일치해야 합니다.

#### Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  LambdaFunctionOverHttps:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: index.handler  
      Runtime: nodejs8.10  
      Policies: AmazonDynamoDBFullAccess  
      Events:  
        HttpPost:  
          Type: Api  
          Properties:  
            Path: '/DynamoDBOperations/DynamoDBManager'  
            Method: post
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

## Using AWS Lambda with AWS CloudTrail

AWS CloudTrail은 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스입니다. CloudTrail은 API 호출을 이벤트로 캡처합니다. AWS 계정의 이벤트 기록을 보유하려면 추적을 생성하십시오. 추적은 CloudTrail이 Amazon S3 버킷으로 이벤트의 로그 파일을 전송할 수 있도록 합니다.

Amazon S3의 버킷 알림 기능을 활용하고 Amazon S3가 직접 객체 생성 이벤트를 AWS Lambda에 게시하게 할 수 있습니다. CloudTrail이 로그를 S3 버킷에 쓸 때마다 Amazon S3는 파라미터로서 Amazon S3 객체 생

성 이벤트를 전달하여 Lambda 함수를 호출할 수 있습니다. S3 이벤트는 이 생성한 로그 객체의 버킷 이름 및 키 이름을 포함한 정보를 제공합니다. Lambda 함수 코드는 로그 객체를 읽고 CloudTrail이 기록한 액세스 레코드를 처리할 수 있습니다. 예를 들어 함수 코드를 작성하여 특정 API 호출이 계정에 생성되는 경우 알릴 수 있습니다.

이 시나리오에서 CloudTrail은 S3 버킷에 액세스 로그를 씁니다. AWS Lambda의 경우 Amazon S3는 이벤트 소스이므로 Amazon S3는 이벤트를 AWS Lambda에 게시하고 Lambda 함수를 호출합니다.

### Example CloudTrail 로그

```
{  
    "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"
    }
}
```

이벤트 소스로 Amazon S3를 구성하는 방법에 대한 자세한 내용은 [Using AWS Lambda with Amazon S3 \(p. 188\)](#) 단원을 참조하십시오.

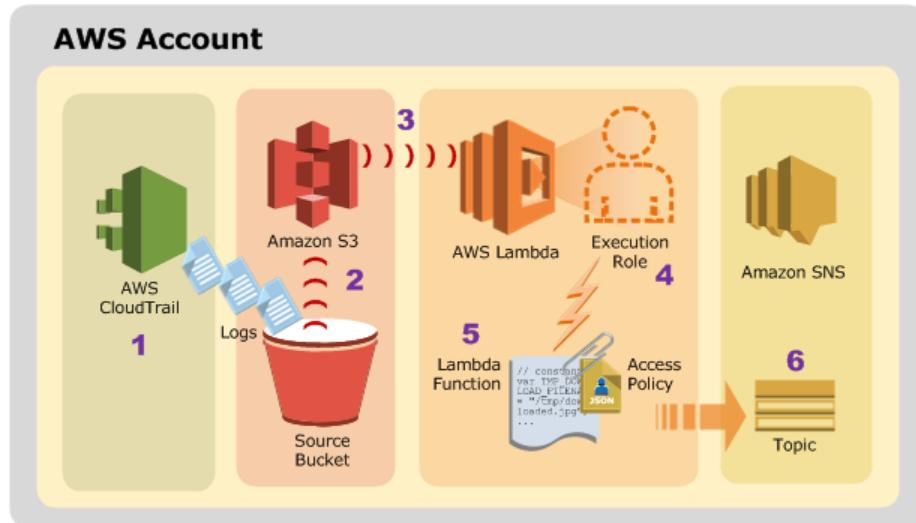
#### 주제

- [자습서: AWS CloudTrail과 함께 AWS Lambda 사용 \(p. 147\)](#)
- [샘플 함수 코드 \(p. 152\)](#)

## 자습서: AWS CloudTrail과 함께 AWS Lambda 사용

이 시나리오에서 AWS CloudTrail은 계정에 대해 실행된 AWS API 호출의 레코드(로그)를 유지하고 SNS 주제를 생성하기 위해 API 호출이 수행되었음을 항상 알립니다. 계정에서 API 호출이 수행되므로 CloudTrail은 사용자가 구성한 Amazon S3 버킷에 로그를 작성합니다. 이 시나리오에서는 Amazon S3가 객체 생성 이벤트를 AWS Lambda에 게시하게 하고, CloudTrail이 로그 객체를 생성하도록 Lambda 함수를 호출하려고 합니다.

다음 다이어그램은 해당 흐름을 보여 줍니다.



1. AWS CloudTrail은 로그를 S3 버킷(객체 생성 이벤트)에 저장합니다.
2. Amazon S3가 객체 생성 이벤트를 감지합니다.
3. Amazon S3는 버킷 알림 구성에 지정된 대로 Lambda 함수를 호출하여 `s3:ObjectCreated:*` 이벤트를 AWS Lambda에 게시합니다. Lambda 함수의 액세스 권한 정책에는 Amazon S3가 함수를 호출할 수 있는 권한이 포함되어 있으므로 Amazon S3는 함수를 호출할 수 있습니다.
4. AWS Lambda는 Lambda 함수를 생성할 때 지정한 실행 역할을 가정하여 Lambda 함수를 실행합니다.
5. Lambda 함수는 파라미터로서 받은 Amazon S3 이벤트를 읽고, CloudTrail 객체가 있는 위치를 확인하고, CloudTrail 객체의 로그 레코드를 처리합니다.

6. 로그에 특정 `eventType` 및 `eventSource` 값이 있는 레코드가 포함되어 있는 경우 해당 함수는 Amazon SNS 주제에 이벤트를 게시합니다. 자습서: [AWS CloudTrail과 함께 AWS Lambda 사용 \(p. 147\)](#)에서 이 메일 프로토콜을 사용하여 SNS 주제를 구독하면 이메일 알림을 받을 수 있습니다.

Amazon S3는 Lambda 함수를 호출할 때 CloudTrail이 생성한 객체의 키 이름과 버킷 이름을 식별하는 S3 이벤트를 전달합니다. 함수 코드는 로그 객체를 읽을 수 있으며 로그에 보고된 API 호출을 알고 있습니다.

CloudTrail이 S3 버킷에 생성하는 각 객체는 하나 이상의 이벤트 레코드가 있는 JSON 객체입니다. 각 레코드는 `eventSource` 및 `eventName`을 제공합니다.

```
{  
    "Records": [  
        {  
            "eventVersion": "1.02",  
            "userIdentity": {  
                ...  
            },  
            "eventTime": "2014-12-16T19:17:43Z",  
            "eventSource": "sns.amazonaws.com",  
            "eventName": "CreateTopic",  
            "awsRegion": "us-east-2",  
            "sourceIPAddress": "72.21.198.64",  
            ...  
        },  
        {  
            ...  
        },  
        ...  
    ]  
}
```

이해를 돋기 위해 Lambda 함수는 Amazon SNS 주제를 생성하기 위한 API 호출이 로그에 보고되면 이메일을 통해 사용자에게 알립니다. 즉, 함수가 로그를 구문 분석할 때 해당 함수는 다음과 같은 레코드를 찾습니다.

- `eventSource = "sns.amazonaws.com"`
- `eventName = "CreateTopic"`

찾고 나면 Amazon SNS 주제에 이벤트를 게시합니다(사용자는 이메일로 알리도록 이 주제를 구성함).

Lambda 함수는 CloudTrail이 생성한 로그 객체의 버킷 이름 및 키 이름을 제공하는 S3 이벤트를 사용합니다. 그런 다음 Lambda 함수 코드는 객체를 읽고 CloudTrail 레코드를 처리합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## CloudTrail 켜기

AWS CloudTrail 콘솔에서 로그를 저장할 CloudTrail의 버킷을 지정하여 계정의 추적을 켭니다. 추적을 구성할 때 SNS 알림을 활성화하지 마십시오.

지침은 AWS CloudTrail User Guide의 [추적 생성 및 업데이트](#)를 참조하십시오.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 역할 이름 – **lambda-cloudtrail-role**.
  - 권한 – 사용자 지정 정책.

```
{  
    "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"  
        }  
    ]  
}
```

이 정책은 함수가 Amazon S3에서 항목을 읽고 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 갖습니다.

## 함수 만들기

다음 예제는 CloudTrail 로그를 처리하고, Amazon SNS 주제가 생성되었을 때 알림을 보냅니다.

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-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
            );
        }
    ],
    callback
}
```

```
], function (err) {
  if (err) {
    console.error('Failed to publish notifications: ', err);
  } else {
    console.log('Successfully published all notifications.');
  }
  callback(null,"message");
});
};
```

### 함수를 만들려면

1. lambda-cloudtrail 폴더의 index.js 파일에 샘플 코드를 복사해 넣습니다.
2. npm을 사용하여 async로 설치합니다.

```
~/lambda-cloudtrail$ npm install async
```

3. 배포 패키지를 만듭니다.

```
~/lambda-cloudtrail$ zip -r function.zip .
```

4. create-function 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name CloudTrailEventProcessing \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 --timeout
10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-cloudtrail-role
```

## 함수 정책에 권한 추가

Lambda 함수의 리소스 정책에 권한을 추가하여 Amazon S3가 함수를 호출하도록 허용합니다.

1. 다음 add-permission 명령을 사용하여 Amazon S3 서비스 보안 주체(s3.amazonaws.com)에 lambda:InvokeFunction 작업을 수행할 수 있는 권한을 부여합니다. 다음 조건이 충족되는 경우에만 에 함수를 호출할 수 있는 권한이 부여됩니다.
  - 객체 생성 이벤트가 특정 버킷에서 감지됩니다.
  - 해당 버킷은 특정 AWS 계정의 소유입니다. 버킷 소유자가 버킷을 삭제하면 다른 AWS 계정이 동일한 이름의 버킷을 생성할 수 있습니다. 이 조건을 통해 특정 AWS 계정만 Lambda 함수를 호출할 수 있습니다.

```
$ 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
```

2. get-policy 명령을 사용하여 함수의 액세스 정책을 확인합니다.

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

## 버킷에 알림 구성

버킷에 알림 구성을 추가하여 Amazon S3에 객체 생성 이벤트를 Lambda에 게시하도록 요청합니다. 구성에 서 다음을 지정합니다.

- 이벤트 유형 – 객체를 생성하는 이벤트 유형입니다.
- Lambda 함수 ARN – Amazon S3가 호출하도록 하려는 Lambda 함수입니다.

```
arn:aws:lambda:us-east-2:123456789012:function:CloudTrailEventProcessing
```

버킷에 알림 구성을 추가하는 방법에 대한 자침은 Amazon Simple Storage Service 콘솔 사용 설명서의 [이벤트 알림 활성화](#)를 참조하십시오.

## 설정 테스트

이제 다음과 같이 설정을 테스트할 수 있습니다.

1. Amazon SNS 주제를 만듭니다.
2. AWS CloudTrail은 버킷에 로그 객체를 생성합니다.
3. Amazon S3는 로그 객체의 위치를 이벤트 데이터로 전달하여 Lambda 함수를 호출합니다.
4. Lambda가 함수를 실행합니다. 이 함수는 로그를 검색하여 CreateTopic SNS 이벤트를 찾아 알림을 보냅니다.

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

주제

- [Node.js \(p. 152\)](#)

## Node.js

다음 예제는 CloudTrail 로그를 처리하고, Amazon SNS 주제가 생성되었을 때 알림을 보냅니다.

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");
});
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Using AWS Lambda with Amazon CloudWatch Events

Amazon CloudWatch Events는 AWS 리소스의 상태 변경에 응답할 수 있게 해줍니다. 리소스 상태가 변경될 경우 이벤트를 이벤트 스트림으로 자동 전송합니다. 스트림에서 선택된 이벤트와 일치할 경우 함수로 라우팅

하여 작업을 실행하는 규칙을 생성할 수 있습니다. 예를 들어, AWS Lambda 함수를 자동으로 호출하여 [EC2 인스턴스](#) 또는 [AutoScaling 그룹](#)의 상태를 기록할 수 있습니다.

규칙 대상 정의를 사용하여 Amazon CloudWatch Events에서 이벤트 소스 매핑을 유지합니다. 자세한 내용은 Amazon CloudWatch Events API 참조의 [PutTargets](#) 작업을 참조하십시오.

Lambda 함수를 생성하고 AWS Lambda에 이를 정기적으로 실행하도록 지시할 수도 있습니다. 고정 비율(예: 1시간 또는 15분마다 함수 실행)을 지정하거나 Cron 식을 지정할 수 있습니다. 표현식 예약에 대한 자세한 내용은 [rate 또는 cron을 사용한 예약 표현식 \(p. 158\)](#)을 참조하십시오.

### Example CloudWatch 이벤트메시지 이벤트

```
{  
    "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"  
    ]  
}
```

AWS Lambda 콘솔이나 AWS CLI를 사용하여 Lambda 함수를 생성할 때 이 기능을 사용할 수 있습니다. AWS CLI를 사용하여 함수를 생성하는 방법은 [AWS CLI를 사용하여 정기적으로 AWS Lambda 함수 실행](#)을 참조하십시오. 콘솔은 이벤트 소스로 CloudWatch 이벤트를 제공합니다. 함수를 생성할 때 이 이벤트 소스를 선택하고 시간 간격을 지정합니다.

함수에 대한 권한을 수동 변경한 경우에는 함수에 예약된 이벤트 액세스를 재적용해야 할 수 있습니다. 이를 위해 다음과 같은 CLI 명령을 사용할 수 있습니다.

```
$ aws lambda add-permission --function-name my-function\  
    --action 'lambda:InvokeFunction' --principal events.amazonaws.com --statement-id  
events-access \  
    --source-arn arn:aws:events:*:123456789012:rule/*
```

각 AWS 계정에는 소스 유형이 CloudWatch 이벤트- 일정인 고유한 이벤트 소스가 최대 100개까지 있을 수 있습니다. 이를 각각이 최대 5개의 함수에 대한 이벤트 소스가 될 수 있습니다. 즉, AWS 계정에서 정기적으로 실행할 수 있는 함수를 최대 500개까지 가질 수 있습니다.

이와 함께 CloudWatch 이벤트 - 일정 유형의 소스를 사용하는 블루프린트(lambda-canary)도 콘솔에 표시됩니다. 이 블루프린트를 사용하여 샘플 함수를 생성하고 이 기능을 테스트할 수 있습니다. 블루프린트가 제공하는 예제 코드는 특정 웹 페이지와 해당 웹 페이지의 특정 텍스트 문자열이 존재하는지 확인합니다. 웹 페이지나 텍스트 문자열이 발견되지 않으면 함수에서 오류가 발생합니다.

## 자습서: 예약 이벤트와 함께 AWS Lambda 사용

이 자습서에서는 다음 작업을 수행합니다.

- lambda-canary 블루프린트를 사용하여 Lambda 함수를 생성합니다. 1분마다 실행되도록 함수를 구성합니다. 함수가 오류 메시지를 반환하면 AWS Lambda가 CloudWatch에 오류 측정치를 로깅합니다.
- AWS Lambda가 CloudWatch에 오류 측정치를 방출할 때 Amazon SNS 주제에 메시지를 게시하도록 Lambda 함수의 Errors 측정치에 대한 CloudWatch 경보를 구성합니다. 이메일 알림을 수신하려면 주제를 구독합니다. 이 자습서에서는 이를 설정하기 위해 다음 작업을 수행했습니다.
  - Amazon SNS 주제를 만듭니다.

- 해당 주제에 새 메시지가 게시될 때 이메일 알림을 수신할 수 있도록 주제를 구독합니다.
- Amazon CloudWatch에서 Lambda 함수의 `Errors` 측정치에 대한 알람을 설정하여 오류 발생 시 SNS 주제에 메시지가 게시되도록 합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

## Lambda 함수 만들기

1. AWS Management 콘솔에 로그인하고 <https://console.aws.amazon.com/lambda/>에서 AWS Lambda 콘솔을 엽니다.
2. 함수 생성을 선택합니다.
3. 블루프린트를 선택합니다.
4. 검색줄에 **canary**를 입력합니다. lambda-canary 블루프린트를 선택한 후 구성을 선택합니다.
5. 다음 설정을 구성합니다.
  - 이름 – **lambda-canary**.
  - 역할 – 1개 이상의 템플릿으로 새 역할 생성
  - 역할 이름 – **lambda-apigateway-role**.
  - 정책 템플릿 – 단순 마이크로서비스 권한
  - 규칙 – 새 규칙 생성.
  - 규칙 이름 – **CheckWebsiteScheduledEvent**.
  - 규칙 설명 – **CheckWebsiteScheduledEvent trigger**.
  - 예약 표현식 – **rate(1 minute)**.
  - 활성화 – True(선택).
  - 환경 변수
    - site – **https://docs.aws.amazon.com/lambda/latest/dg/welcome.html**
    - expected – **What Is AWS Lambda?**
6. 함수 생성을 선택합니다.

CloudWatch 이벤트는 예약 표현식을 기반으로 1분마다 이벤트를 생성합니다. 이벤트는 Lambda 함수를 트리거하여, 지정된 페이지에 해당 문자열이 나타나는지 확인합니다. 표현식 예약에 대한 자세한 내용은 [rate 또는 cron을 사용한 예약 표현식 \(p. 158\)](#)을 참조하십시오.

## Lambda 함수 테스트

Lambda 콘솔에서 제공하는 예제 이벤트를 사용하여 함수를 테스트합니다.

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. lambda-canary를 선택합니다.
3. 페이지 상단의 테스트 버튼 옆의 드롭다운 메뉴에서 테스트 이벤트 구성자를 선택합니다.
4. CloudWatch 이벤트 이벤트 템플릿을 사용하여 새 이벤트를 생성합니다.
5. Create를 선택합니다.
6. [Test]를 선택합니다.

함수 실행의 출력은 페이지 상단에 표시됩니다.

## Amazon SNS 주제를 생성하여 구독

canary 함수가 오류를 반환할 경우 알림을 받을 Amazon Simple Notification Service 주제를 생성합니다.

### 주제 생성

1. [Amazon SNS 콘솔](#)을 엽니다.
2. [Create topic]을 선택합니다.
3. 다음 설정을 사용하여 주제를 생성합니다.
  - 이름 – **lambda-canary-notifications**.
  - 표시 이름 – **Canary**.
4. Create subscription을 선택합니다.
5. 다음 설정을 사용하여 구독을 생성합니다.
  - 프로토콜 – **Email**.
  - 앤드포인트 – 이메일 주소.

Amazon SNS는 주제에 대한 알기 쉬운 이름을 반영하여 Canary <no-reply@sns.amazonaws.com>에서 이메일을 발송합니다. 이메일에 포함된 링크를 사용하여 주소를 확인합니다.

## 경보 구성

Amazon CloudWatch에서 Lambda 함수를 모니터링하여 실패 시 알림을 전송하는 경보를 구성합니다.

### 경보를 만들려면

1. [CloudWatch 콘솔](#)을 엽니다.
  2. 알람을 선택합니다.
  3. 경보 생성을 선택합니다.
  4. 알람을 선택합니다.
  5. 다음 설정을 사용하여 경보를 생성합니다.
    - 측정치 – lambda-canary 오류.
- lambda canary errors**를 검색하여 측정치를 확인합니다.
- 통계 – **Sum**.
- 미리 보기 그래프 위의 드롭다운에서 통계를 선택합니다.
- 이름 – **lambda-canary-alarm**.
  - 설명 – **Lambda canary alarm**.
  - 임계값 – 오류가  $\geq 1$ 일 때마다.
  - Send notification to(알림 보내기) – **lambda-canary-notifications**

## 경보 테스트

함수 구성을 업데이트하여 함수가 오류를 반환하고 경보가 트리거되도록 합니다.

### 경보를 트리거하려면

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. lambda-canary를 선택합니다.

3. 환경 변수에서 `expected`를 **404**로 설정합니다.
4. Save를 선택합니다

잠시 기다린 후 Amazon SNS에서 발송한 이메일을 확인합니다.

## CloudWatch 이벤트 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 \(p. 154\)](#)의 Lambda 애플리케이션을 위한 샘플 AWS SAM 템플릿입니다. 아래 텍스트를 `.yaml` 파일로 복사하고 이전에 만든 ZIP 패키지 옆에 저장합니다. Handler 및 Runtime 파라미터 값은 이전 단원에서 함수를 생성할 때 사용한 것과 일치해야 합니다.

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'
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

## rate 또는 cron을 사용한 예약 표현식

AWS Lambda는 분당 최대 한 번의 빈도로 표준 rate 표현식과 cron 표현식을 지원합니다. CloudWatch 이벤트 rate 표현식은 다음 형식을 갖습니다.

```
rate(Value Unit)
```

**Value**는 양의 정수이고 **Unit**은 분, 시간, 일이 될 수 있습니다. 단수 값의 경우에는 단위가 단수(예: `rate(1 day)`)여야 하고, 그렇지 않은 경우에는 복수(예: `rate(5 days)`)입니다.

Rate 표현식의 예

주파수	표현식
5분마다	<code>rate(5 minutes)</code>
매 시간	<code>rate(1 hour)</code>
7일마다	<code>rate(7 days)</code>

Cron 표현식의 형식은 다음과 같습니다.

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

cron 식의 예

주파수	표현식
매일 10:15 AM(UTC)	<code>cron(15 10 * * ? *)</code>
월요일부터 금요일까지 오후 6:00	<code>cron(0 18 ? * MON-FRI *)</code>
매월 첫날 오전 8:00	<code>cron(0 8 1 * ? *)</code>
평일 10분마다	<code>cron(0/10 * ? * MON-FRI *)</code>
평일 오전 8:00부터 오후 5:55까지 5분마다	<code>cron(0/5 8-17 ? * MON-FRI *)</code>
매월 첫째 월요일 오전 9:00	<code>cron(0 9 ? * 2#1 *)</code>

다음을 참조하십시오.

- Lambda 콘솔을 사용하고 있는 경우 표현식에 cron 접두사를 포함하지 마십시오.
- day-of-month 또는 day-of-week 값 중 하나는 반드시 물음표(?)여야 합니다.

자세한 내용은 CloudWatch 이벤트 사용 설명서의 [규칙에 대한 예약 표현식](#)을 참조하십시오.

## Using AWS Lambda with Amazon CloudWatch Logs

Lambda 함수를 사용하여 Amazon CloudWatch Logs 로그 스트림의 로그를 모니터링하고 분석할 수 있습니다. 하나 이상의 로그 스트림에 대해 [구독](#)을 생성하여 로그가 생성될 때 또는 패턴과 일치할 때(선택 사항) 함

수를 호출할 수 있습니다. 함수를 사용하여 알림을 보내거나 데이터베이스 또는 스토리지에 로그를 영구적으로 유지합니다.

CloudWatch Logs는 인코딩된 로그 데이터를 포함하는 이벤트와 비동기적으로 함수를 호출합니다.

#### Example Amazon CloudWatch Logs 메시지 이벤트

```
{  
  "awslogs": {  
    "data":  
      "ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIIoAiMTIzNDU2Nzg5MDEyIiwKICAgICJsb2dH  
  }  
}
```

디코딩 시 로그 데이터는 다음 구조를 갖춘 JSON 문서입니다.

#### Example Amazon CloudWatch Logs 메시지 데이터(디코딩됨)

```
{  
  "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"  
    }  
  ]  
}
```

CloudWatch Logs를 트리거로 사용하는 샘플 애플리케이션은 [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#) 단원을 참조하십시오.

## AWS Lambda을(를) AWS CloudFormation과 함께 사용

AWS CloudFormation 템플릿에서는 사용자 지정 리소스의 대상으로 Lambda 함수를 지정할 수 있습니다. 사용자 지정 리소스를 사용하여 파라미터를 처리하고, 설정 값을 검색하거나, 스택 수명 주기 이벤트 도중 다른 AWS 서비스를 호출할 수 있습니다.

다음 예시는 템플릿에 정의된 함수를 호출합니다.

#### Example – 사용자 정의 리소스 정의

```
Resources:  
  primerinvoke:  
    Type: AWS::CloudFormation::CustomResource  
    Version: "1.0"
```

```
Properties:  
  ServiceToken: !GetAtt primer.Arn  
  FunctionName: !Ref randomerror
```

서비스 토큰은 스택 생성, 업데이트 또는 삭제 시 AWS CloudFormation이 호출한 함수의 ARN(Amazon Resource Name)입니다. AWS CloudFormation이 원형 그대로 함수로 전달하는, `FunctionName` 같은 추가 속성을 포함할 수도 있습니다.

AWS CloudFormation은 콜백 URL을 포함하는 이벤트와 비동기적으로 Lambda 함수를 호출합니다.

#### Example – AWS CloudFormation 메시지 이벤트

```
{  
  "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"  
  }  
}
```

함수는 성공 또는 실패를 나타내는 콜백 URL에 대한 응답을 반환할 책임을 집니다. 전체 응답 구문은 [사용자 지정 리소스 응답 객체](#)에서 확인할 수 있습니다.

#### Example – AWS CloudFormation 사용자 지정 리소스 응답

```
{  
  "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 CloudFormation은 응답 전송을 처리하는 `cfn-response`라는 라이브러리를 제공합니다. 템플릿 내에서 함수를 정의하려면, 함수를 이름으로 요청하면 됩니다. 그러면 AWS CloudFormation이 함수를 위해 생성된 배포 패키지에 라이브러리를 추가합니다.

다음 예시 함수는 두 번째 함수를 호출합니다. 호출이 성공한다면, 함수는 성공 응답을 AWS CloudFormation에 전송하고 스택 업데이트가 진행됩니다.

#### Example – 사용자 지정 리소스 함수

```
Resources:  
  primer:
```

```
Type: AWS::Serverless::Function
Properties:
  Handler: index.handler
  Runtime: nodejs8.10
  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
```

사용자 지정 리소스가 호출하는 함수가 템플릿에 정의되어 있지 않다면, AWS CloudFormation 사용 설명서의 [cfn-response 모듈](#)에서 cfn-response의 소스 코드를 얻을 수 있습니다.

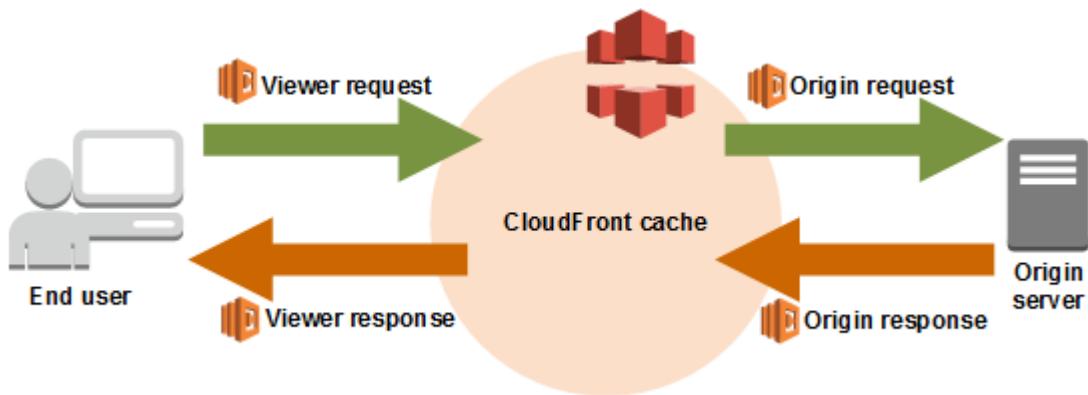
사용자 지정 리소스를 사용하여, 함수의 로그 그룹이 해당 그룹을 의존하는 다른 리소스보다 먼저 생성되도록 하는 방법은 [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#)에서 확인할 수 있습니다.

사용자 지정 리소스에 대한 자세한 정보는 AWS CloudFormation 사용 설명서의 [사용자 지정 리소스](#)를 참조하십시오.

## CloudFront Lambda@Edge와 함께 AWS Lambda

Lambda@Edge를 사용하면 Node.js Lambda 함수를 실행하여 CloudFront가 제공하는 콘텐츠를 사용자 지정하여 AWS 위치의 함수를 최종 사용자와 더 가깝게 실행할 수 있습니다. 이 함수는 서버 프로비저닝 또는 관리 없이 이벤트에 응답하여 실행됩니다. Lambda 함수를 사용하여 CloudFront 요청 및 응답을 다음과 같이 변경할 수 있습니다.

- CloudFront가 최종 사용자로부터 요청을 수신한 후(최종 사용자 요청)
- CloudFront가 오리진에 요청을 전달하기 전(오리진 요청)
- CloudFront가 오리진으로부터 응답을 수신한 후(오리진 응답)
- CloudFront가 최종 사용자에게 응답을 전달하기 전(최종 사용자 응답)



또한 요청을 오리진으로 보내지 않고도 최종 사용자에 대한 응답을 생성할 수 있습니다.

#### Example CloudFront 메시지 이벤트

```
{  
    "Records": [  
        {  
            "cf": {  
                "config": {  
                    "distributionId": "EDFDVBD6EXAMPLE"  
                },  
                "request": {  
                    "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",  
                    "method": "GET",  
                    "uri": "/picture.jpg",  
                    "headers": {  
                        "host": [  
                            {  
                                "key": "Host",  
                                "value": "d111111abcdef8.cloudfront.net"  
                            }  
                        ],  
                        "user-agent": [  
                            {  
                                "key": "User-Agent",  
                                "value": "curl/7.51.0"  
                            }  
                        ]  
                    }  
                }  
            }  
        ]  
    ]  
}
```

Lambda@Edge를 사용하면 다음과 같은 다양한 솔루션을 구축할 수 있습니다.

- 다양한 버전의 사이트에 URL을 다시 쓰기 위해 쿠키를 검사합니다(A/B 테스트용).
- 요청을 제출한 디바이스에 대한 정보가 포함된 User-Agent 헤더를 기반으로 사용자에게 다른 객체를 전송합니다. 예를 들어 디바이스별로 사용자에게 서로 다른 해상도로 이미지를 보낼 수 있습니다.
- 헤더 또는 허가된 토큰을 검사하여 요청을 오리진에 전달하기 전에 해당 헤더를 삽입하고 액세스 제어를 허용합니다.
- 헤더를 추가, 삭제 및 수정하고 캐시의 다양한 객체로 사용자를 이동시키도록 URL 경로를 다시 씁니다.

- 새로운 HTTP 응답을 생성하여 인증되지 않은 사용자를 로그인 페이지로 리디렉션하거나 엣지에서 정적 웹 페이지를 만들고 전달합니다. 자세한 내용은 Amazon CloudFront 개발자 안내서의 [Lambda 함수를 사용하여 최종 사용자 및 오리진 요청에 대한 HTTP 응답 생성](#)을 참조하십시오.

[Lambda@Edge](#)를 사용하는 방법에 대한 자세한 내용은 [Lambda@Edge로 CloudFront 사용하기 단원](#)을 참조하십시오.

## Using AWS Lambda with AWS CodeCommit

AWS CodeCommit 리포지토리에 대한 트리거를 생성하여 리포지토리의 이벤트가 Lambda 함수를 호출하게 할 수 있습니다. 예를 들어, 브랜치 또는 태그가 생성되거나 기존 브랜치를 푸시할 때 함수를 호출할 수 있습니다.

Example AWS CodeCommit메시지 이벤트

```
{  
    "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"  
        }  
    ]  
}
```

자세한 내용은 [AWS CodeCommit 리포지토리 트리거 관리](#)를 참조하십시오.

## Using AWS Lambda with Amazon Cognito

Amazon Cognito 이벤트 기능을 통해 Amazon Cognito에서 중요한 이벤트에 반응하여 Lambda 함수를 실행할 수 있습니다. 예를 들어 데이터 세트가 동기화될 때마다 게시되는 동기화 트리거 이벤트에 대한 함수를 호출할 수 있습니다. 자세한 내용 및 예제를 살펴보려면 모바일 개발 블로그의 [Introducing Amazon Cognito Events: Sync Triggers](#)를 참조하십시오.

Example Amazon Cognito메시지 이벤트

```
{
```

```
"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
}
```

Amazon Cognito 이벤트 구독 구성은 이벤트 소스 매핑을 구성합니다. 이벤트 소스 매핑 및 샘플 이벤트에 대한 자세한 내용은 Amazon Cognito 개발자 안내서의 [Amazon Cognito 이벤트](#)를 참조하십시오.

## Using AWS Lambda with AWS Config

AWS Lambda 함수를 사용하여 AWS 리소스 구성이 사용자 지정 구성 규칙을 준수하는지 여부를 평가할 수 있습니다. 리소스가 생성, 삭제 또는 변경되면 AWS Config는 이러한 변경 사항을 기록하고 Lambda 함수에 정보를 보냅니다. 그런 다음 Lambda 함수는 변경 사항을 평가하고 결과를 AWS Config에 보고합니다. 그런 다음 AWS Config를 사용하여 전체 리소스 규정 준수를 평가할 수 있습니다. 즉, 어떤 리소스가 규칙 미준수이고 어떤 구성 속성이 규칙 미준수의 원인인지 확인할 수 있습니다.

### Example AWS Config 메시지 이벤트

```
{
    "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\": \"2016-02-17T01:34:34.043Z\", \"awsAccountId\": \"000000000000\", \"configurationItemStatus\": \"OK\", \"resourceId\": \"i-00000000\", \"ARN\": \"arn:aws:ec2:us-east-1:000000000000:instance/i-00000000\", \"awsRegion\": \"us-east-1\", \"availabilityZone\": \"us-east-1a\", \"resourceType\": \"AWS::EC2::Instance\", \"tags\": {\"Foo\": \"Bar\"}, \"relationships\": [{\"resourceId\": \"eipalloc-00000000\", \"resourceType\": \"AWS::EC2::EIP\", \"name\": \"Is attached to ElasticIp\"}], \"configuration\": {\"foo\": \"bar\"}, \"messageType\": \"ConfigurationItemChangeNotification\"}, \"ruleParameters\": {\"myParameterKey\": \"myParameterValue\"}, \"resultToken\": \"myResultToken\", \"eventLeftScope\": false, \"executionRoleArn\": \"arn:aws:iam::012345678912:role/config-role\", \"configRuleArn\": \"arn:aws:config:us-east-1:012345678912:config-rule/config-rule-0123456\", \"configRuleName\": \"change-triggered-config-rule\", \"configRuleId\": \"config-rule-0123456\", \"accountId\": \"012345678912\", \"version\": \"1.0\"}
}
```

자세한 내용은 [AWS Config 규칙을 사용하여 리소스 평가](#)를 참조하십시오.

# Using AWS Lambda with Amazon DynamoDB

AWS Lambda 함수를 사용하여 [Amazon DynamoDB 스트림](#)의 레코드를 처리할 수 있습니다. DynamoDB 스트림을 사용하여 DynamoDB가 업데이트될 때마다 추가 작업을 수행하는 Lambda 함수를 트리거할 수 있습니다.

Lambda는 스트림에서 레코드를 읽고 스트림 레코드를 포함하는 이벤트와 [동기적으로 \(p. 80\)](#) 함수를 호출합니다. Lambda는 레코드를 배치(batch)로 읽고 함수를 호출하여 배치의 레코드를 처리합니다.

## Example DynamoDB 스트림 레코드 이벤트

```
{  
  "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"  
          }  
        }  
      }  
    }  
  ]  
}
```

```
        "N": "101"
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES"
},
"awsRegion": "us-west-2",
"eventName": "MODIFY",
"eventSourceARN": sourcearn,
"eventSource": "aws:dynamodb"
}
```

Lambda는 초당 4회의 속도로 DynamoDB 스트림 스트림의 색드에서 레코드를 폴링합니다. 레코드를 사용할 수 있으면 Lambda가 함수를 호출하고 결과를 기다립니다. 처리가 성공하면 Lambda가 레코드를 더 받을 때 까지 폴링을 재개합니다.

함수가 오류를 반환하면 Lambda는 처리가 성공할 때까지 또는 데이터가 만료될 때까지 배치(batch)를 재시도합니다. 문제가 해결될 때까지 색드의 데이터는 처리되지 않습니다. 색드가 중지되어 데이터 손실이 발생하지 않도록 하기 위해 코드의 레코드 처리 오류를 해결합니다.

#### 주제

- [이벤트 소스 매핑 생성 \(p. 166\)](#)
- [실행 역할 권한 \(p. 167\)](#)
- [자습서: Amazon DynamoDB 스트림에 AWS Lambda 사용 \(p. 167\)](#)
- [샘플 함수 코드 \(p. 171\)](#)
- [DynamoDB 애플리케이션을 위한 AWS SAM 템플릿 \(p. 174\)](#)

## 이벤트 소스 매핑 생성

이벤트 소스 매핑을 생성하여 Lambda가 스트림의 레코드를 Lambda 함수로 전송하도록 지시합니다. 여러 이벤트 소스 매핑을 생성하여 여러 Lambda 함수로 동일한 데이터를 처리하거나, 하나의 함수로 여러 스트림의 항목을 처리할 수 있습니다.

함수가 Lambda 콘솔의 DynamoDB 스트림에서 일도록 구성하려면 DynamoDB 트리거를 생성합니다.

트리거를 생성하려면

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. 함수를 선택합니다.
3. 디자이너에서 트리거 유형을 선택하여 함수에 트리거를 추가합니다.
4. 트리거 구성에서 필요한 옵션을 구성한 후 추가를 선택합니다.
5. Save를 선택합니다.

#### 이벤트 소스 옵션

- **DynamoDB 테이블 –** 레코드를 읽을 DynamoDB 테이블.
- **배치 크기 –** 각 배치에서 색드로부터 읽을 레코드 수(최대 1,000개). Lambda는 한 번의 호출로 배치의 모든 레코드를 함수에 전달합니다. 단, 이벤트의 총 크기가 페이로드 한도(6 MB)를 초과하지 않아야 합니다.
- **시작 위치 –** 새 레코드 또는 기존의 모든 레코드만 처리합니다.
  - **최신 –** 스트림에 추가된 새 레코드를 처리합니다.
  - **수평 트리밍 –** 스트림의 모든 레코드를 처리합니다.

기존 레코드 처리 후 함수는 캐시업되고 새 레코드를 계속 처리합니다.

- 활성화 – 이벤트 소스를 비활성화하여 레코드 처리를 중지합니다. Lambda는 마지막으로 처리된 레코드를 추적하여 다시 활성화되면 그 지점부터 처리를 다시 시작합니다.

나중에 이벤트 소스 구성을 관리하기 위해 디자이너에서 트리거를 선택합니다.

## 실행 역할 권한

Lambda는 DynamoDB 스트림 스트림과 관련된 리소스를 관리하기 위해 다음 권한이 필요합니다. 이러한 권한을 함수의 [실행 역할 \(p. 9\)](#)에 추가합니다.

- dynamodb:DescribeStream
- dynamodb:GetRecords
- dynamodb:GetShardIterator
- dynamodb>ListStreams

`AWSLambdaDynamoDBExecutionRole` 관리형 정책에는 이러한 권한이 포함되어 있습니다. 자세한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.

## 자습서: Amazon DynamoDB 스트림에 AWS Lambda 사용

이 자습서에서는 Amazon DynamoDB 스트림의 이벤트를 사용하기 위해 Lambda 함수를 생성합니다.

### 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$)와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

### 실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 개체 – Lambda.
  - 권한 – `AWSLambdaDynamoDBExecutionRole`.

- 역할 이름 – **lambda-dynamodb-role**.

AWSLambdaDynamoDBExecutionRole은 함수가 DynamoDB에서 항목을 읽고 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 함수 만들기

다음은 DynamoDB 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 함수 코드 \(p. 171\)](#) 단원을 참조하십시오.

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");
};
```

함수를 만들려면

1. 샘플 코드를 index.js 파일에 복사합니다.
2. 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

3. create-function 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name ProcessDynamoDBRecords \
--zip-file file:///function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/lambda-dynamodb-role
```

## Lambda 함수 테스트

이 단계에서는 invoke AWS Lambda CLI 명령과 다음 샘플 DynamoDB 이벤트를 사용하여 수동으로 Lambda 함수를 호출합니다.

Example input.txt

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
```

```
"dynamodb":{  
    "Keys":{  
        "Id":{  
            "N":"101"  
        }  
    },  
    "NewImage":{  
        "Message":{  
            "S":"New item!"  
        },  
        "Id":{  
            "N":"101"  
        }  
    },  
    "SequenceNumber":"111",  
    "SizeBytes":26,  
    "StreamViewType":"NEW_AND_OLD_IMAGES"  
},  
"eventSourceARN":"stream-ARN"  
},  
{  
    "eventID":"2",  
    "eventName":"MODIFY",  
    "eventVersion":"1.0",  
    "eventSource":"aws:dynamodb",  
    "awsRegion":"us-east-1",  
    "dynamodb":{  
        "Keys":{  
            "Id":{  
                "N":"101"  
            }  
        },  
        "NewImage":{  
            "Message":{  
                "S":"This item has changed"  
            },  
            "Id":{  
                "N":"101"  
            }  
        },  
        "OldImage":{  
            "Message":{  
                "S":"New item!"  
            },  
            "Id":{  
                "N":"101"  
            }  
        },  
        "SequenceNumber":"222",  
        "SizeBytes":59,  
        "StreamViewType":"NEW_AND_OLD_IMAGES"  
},  
"eventSourceARN":"stream-ARN"  
},  
{  
    "eventID":"3",  
    "eventName":"REMOVE",  
    "eventVersion":"1.0",  
    "eventSource":"aws:dynamodb",  
    "awsRegion":"us-east-1",  
    "dynamodb":{  
        "Keys":{  
            "Id":{  
                "N":"101"  
            }  
        },  
    },  
}
```

```
"OldImage":{  
    "Message":{  
        "S":"This item has changed"  
    },  
    "Id":{  
        "N":"101"  
    }  
},  
"SequenceNumber":"333",  
"SizeBytes":38,  
"StreamViewType":"NEW_AND_OLD_IMAGES"  
},  
"eventSourceARN":"stream-ARN"  
}  
]  
}
```

다음 invoke 명령을 실행합니다.

```
$ aws lambda invoke --function-name ProcessDynamoDBRecords --payload file://input.txt  
outputfile.txt
```

이 함수는 응답 본문에서 문자열 메시지(코드의 `context.succeed()`에 있는 메시지)를 반환합니다.

`outputfile.txt` 파일의 출력을 확인합니다.

## 스트림을 활성화하여 DynamoDB 테이블 생성

활성화된 스트림으로 Amazon DynamoDB 테이블을 생성합니다.

DynamoDB 테이블을 만들려면

1. [DynamoDB 콘솔](#)을 엽니다.
2. [Create table]을 선택합니다.
3. 다음 설정을 사용하여 테이블을 생성합니다.
  - 테이블 이름 – **lambda-dynamodb-stream**
  - 기본 키 – **id**(문자열)
4. Create를 선택합니다.

스트림을 활성화하려면

1. [DynamoDB 콘솔](#)을 엽니다.
2. 테이블을 선택합니다.
3. `lambda-dynamodb-stream` 테이블을 선택합니다.
4. 개요에서 스트림 관리를 선택합니다.
5. [Enable]을 선택합니다.

스트림 ARN을 적어둡니다. 다음 단계에서 스트림을 Lambda 함수와 연결할 때 필요합니다. 스트림 활성화에 대한 자세한 정보는 [DynamoDB 스트림을 사용하여 Table Activity 캡처하기](#)를 참조하십시오.

## AWS Lambda에서 이벤트 소스 추가

AWS Lambda에서 이벤트 소스 매핑을 생성합니다. 이 이벤트 소스 매핑은 DynamoDB 스트림을 함수와 연결합니다. 이 이벤트 소스 매핑을 생성하면 가 스트림 폴링을 시작합니다.

다음 AWS CLI `create-event-source-mapping` 명령을 실행합니다. 명령이 실행된 후 UUID를 적어둡니다. 예를 들어, 이벤트 소스 매핑을 삭제할 때처럼 모든 명령에서 이벤트 소스 매핑을 참조하려면 이 UUID가 필요합니다.

```
$ aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

이는 지정된 DynamoDB 스트림과 Lambda 함수 사이의 매핑을 생성합니다. DynamoDB 스트림을 여러 Lambda 함수와 연결하고 동일한 Lambda 함수를 여러 스트림과 연결할 수 있습니다. 하지만 함수는 공유하는 스트림에 대해 읽기 처리량을 공유합니다.

다음 명령을 실행하여 이벤트 소스 매핑 목록을 가져올 수 있습니다.

```
$ aws lambda list-event-source-mappings
```

이 목록은 사용자가 생성한 모든 이벤트 소스 매핑을 반환하며 각 매핑에 대해 `LastProcessingResult`를 표시합니다. 이 필드는 문제가 있을 경우 유용한 메시지를 제공하는데 사용됩니다. `No records processed`(AWS Lambda가 폴링을 시작하지 않았거나 스트림에 레코드가 없음을 나타냄) 및 `OK`(AWS Lambda가 레코드를 성공적으로 읽고 Lambda 함수를 호출함을 나타냄) 같은 값은 아무 문제가 없다는 점을 나타냅니다. 문제가 있는 경우 오류 메시지를 수신합니다.

이벤트 소스 매핑이 많을 경우 함수 이름 파라미터를 사용하여 결과 범위를 좁히십시오.

```
$ aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

## 설정 테스트

종합적 경험을 테스트합니다. 테이블 업데이트를 수행할 때 DynamoDB는 이벤트 레코드를 스트림에 기록합니다. AWS Lambda가 스트림을 폴링하면 스트림의 새 레코드를 감지하고 이벤트를 함수에 전달하여 사용자의 Lambda 함수를 실행합니다.

1. DynamoDB 콘솔에서 항목을 테이블에 추가, 업데이트 및 삭제합니다. DynamoDB는 이러한 작업에 대한 레코드를 스트림에 기록합니다.
2. AWS Lambda가 스트림을 폴링하고 스트림에 대한 업데이트를 감지하면 스트림에서 찾은 이벤트 데이터를 전달하여 Lambda 함수를 호출합니다.
3. 함수는 Amazon CloudWatch에서 로그를 실행하고 생성합니다. Amazon CloudWatch 콘솔에서 보고된 로그를 확인할 수 있습니다.

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

### 주제

- [Node.js \(p. 171\)](#)
- [Java 8 \(p. 172\)](#)
- [C# \(p. 173\)](#)
- [Python 3 \(p. 173\)](#)
- [Go \(p. 174\)](#)

## Node.js

다음 예에서는 DynamoDB의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

### 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");
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Java 8

다음 예제는 DynamoDB의 메시지를 처리하고, 그 내용을 로깅합니다. `handleRequest`는 AWS Lambda의 이벤트 데이터를 호출하고 제공하는 핸들러입니다. 해당 핸들러는 `aws-lambda-java-events` 라이브러리에서 사전 정의된 `DynamodbEvent` 클래스를 사용합니다.

### 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.";
    }
}
```

핸들러가 예외 없이 정상적으로 반환되면 Lambda는 래코드의 입력 배치가 성공적으로 처리된 것으로 간주하고 스트림의 새 레코드 읽기를 시작합니다. 핸들러에 예외가 발생하면 래코드의 입력 배치를 처리하지 않은 것으로 간주하고 동일한 레코드 배치로 함수를 다시 호출합니다.

### 종속성

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 지침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## C#

다음 예제는 DynamoDB의 메시지를 처리하고, 그 내용을 로깅합니다. `ProcessDynamoEvent`는 AWS Lambda 이벤트 데이터를 호출하고 제공하는 핸들러입니다. 해당 핸들러는 `Amazon.Lambda.DynamoDBEvents` 라이브러리에서 사전 정의된 `DynamoDbEvent` 클래스를 사용합니다.

### 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());
            }
        }
    }
}
```

.NET Core 프로젝트의 `Program.cs`를 위의 샘플로 바꿉니다. 자침은 [.NET Core CLI \(p. 296\)](#) 단원을 참조 하십시오.

## Python 3

다음 예에서는 DynamoDB의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

### 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'])))
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Go

다음 예에서는 DynamoDB의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

### 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())
            }
        }
    }
}
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## DynamoDB 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 애플리케이션 \(p. 167\)](#)을 위한 샘플 AWS SAM 템플릿입니다. 아래 텍스트를 .yaml 파일로 복사하고 이전에 만든 ZIP 패키지 옆에 저장합니다. Handler 및 Runtime 파라미터 값은 이전 단원에서 함수를 생성할 때 사용한 것과 일치해야 합니다.

### 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
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

## Amazon Kinesis에 AWS Lambda 사용

AWS Lambda 함수를 사용하여 [Amazon Kinesis 데이터 스트림](#)의 레코드를 처리할 수 있습니다. Kinesis를 사용하면 많은 소스로부터 데이터를 수집한 후 여러 소비자를 사용하여 그러한 데이터를 처리할 수 있습니다. Lambda는 표준 데이터 스트림 반복자(iterator)와 HTTP/2 스트림 소비자를 지원합니다.

Lambda는 데이터 스트림에서 레코드를 읽고 스트림 레코드를 포함하는 이벤트와 [동기적으로 \(p. 80\)](#) 함수를 호출합니다. Lambda는 레코드를 배치(batch)로 읽고 함수를 호출하여 배치의 레코드를 처리합니다.

### Example Kinesis 레코드 이벤트

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
        "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":  
        "49590338271490256608559692540925702759324208523137515618",  
        "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"  
}  
]  
}
```

동일한 스트림에서 레코드를 읽는 여러 애플리케이션이 있을 경우, 표준 반복자 대신 Kinesis 스트림 소비자를 사용할 수 있습니다. 소비자는 전용 읽기 처리량을 보유하므로 동일한 데이터에 대해 다른 소비자와 경합할 필요가 없습니다. Kinesis는 소비자를 사용하여 HTTP/2 연결을 통해 Lambda에 레코드를 푸시하므로 레코드 추가와 함수 호출 간의 지연 시간을 최소화할 수 있습니다.

함수가 오류를 반환하면 Lambda는 처리가 성공할 때까지 또는 데이터가 만료될 때까지 배치(batch)를 재시도합니다. 문제가 해결될 때까지 색드의 데이터는 처리되지 않습니다. 색드 중지 및 데이터 손실을 방지하기 위해 코드의 프로세싱 오류를 처리하고 기록하십시오.

#### 주제

- [데이터 스트림과 함수 구성 \(p. 176\)](#)
- [이벤트 소스 매핑 생성 \(p. 177\)](#)
- [이벤트 소스 매핑 API \(p. 178\)](#)
- [실행 역할 권한 \(p. 178\)](#)
- [Amazon CloudWatch 측정치 \(p. 179\)](#)
- [자습서: Amazon Kinesis와 함께 AWS Lambda 사용 \(p. 179\)](#)
- [샘플 함수 코드 \(p. 183\)](#)
- [Kinesis 애플리케이션을 위한 AWS SAM 템플릿 \(p. 185\)](#)

## 데이터 스트림과 함수 구성

Lambda 함수는 데이터 스트림의 소비자 애플리케이션입니다. 이 함수는 각 색드에서 한 번에 한 개의 레코드 배치를 처리합니다. Lambda 함수를 데이터 스트림에 매핑하거나(표준 반복자), 스트림의 소비자에 매핑할 수 있습니다([향상된 팬아웃](#)).

표준 반복자의 경우 Lambda는 초당 1회의 속도로 Kinesis 스트림의 각 색드에서 레코드를 폴링합니다. 더 많은 레코드가 사용 가능하면 Lambda는 구성된 최대 배치(batch) 크기보다 작은 배치를 받을 때까지 배치를 계속 처리합니다. 함수는 색드의 다른 소비자와 읽기 처리량을 공유합니다.

지연 시간을 최소화하고 읽기 처리량을 최대화하려면 데이터 스트림 소비자를 생성하십시오. 스트림 소비자는 각 색드에 대해 전용 연결을 설정하므로 스트림에서 읽는 다른 애플리케이션에 영향을 주지 않습니다. 전용 처리량은 많은 애플리케이션에서 동일한 데이터를 읽거나, 큰 레코드를 갖는 스트림을 재처리하는 경우에 유용합니다.

스트림 소비자는 HTTP/2를 사용하여 수명이 긴 연결을 통해 레코드를 Lambda에 푸시하고 요청 헤더를 압축함으로써 지연 시간을 최소화합니다. Kinesis [RegisterStreamConsumer API](#)를 사용하여 스트림 소비자를 생성할 수 있습니다.

```
$ 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
    }
}
```

함수가 레코드를 처리하는 속도를 높이려면 데이터 스트림에 샤드를 추가하십시오. 함수가 오류를 반환할 경우 Lambda는 각 샤드의 레코드를 순서대로 처리하고 샤드의 추가 레코드 처리를 중지합니다. 샤드가 많을수록 한 번에 더 많은 배치가 처리되므로 동시 실행에 대한 오류의 영향이 줄어듭니다.

Note

한 배치의 레코드 총 크기가 [페이로드 한도 \(p. 7\)](#)를 초과하면, Lambda는 처리를 위해 더 작은 배치로 분할합니다.

함수가 샤드당 한 개의 동시 실행을 처리하도록 확장할 수 없을 경우 함수에 대한 [한도 증가를 요청 \(p. 7\)](#)하거나 [동시성을 예약 \(p. 37\)](#)하십시오. 함수에 사용 가능한 동시성은 Kinesis 데이터 스트림의 샤드 수와 일치하거나 그보다 많아야 합니다.

## 이벤트 소스 매핑 생성

이벤트 소스 매핑을 생성하여 Lambda가 데이터 스트림의 레코드를 Lambda 함수로 전송하도록 지시합니다. 여러 이벤트 소스 매핑을 생성하여 여러 Lambda 함수로 동일한 데이터를 처리하거나, 하나의 함수로 여러 데이터 스트림의 항목을 처리할 수 있습니다.

함수가 Lambda 콘솔의 Kinesis에서 읽도록 구성하려면 Kinesis 트리거를 생성합니다.

### 트리거를 생성하려면

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. 함수를 선택합니다.
3. 디자이너에서 트리거 유형을 선택하여 함수에 트리거를 추가합니다.
4. 트리거 구성에서 이벤트 소스 옵션을 구성한 후 추가를 선택합니다.
5. Save를 선택합니다.

Lambda는 Kinesis 이벤트 소스에 대해 다음과 같은 옵션을 지원합니다.

### 이벤트 소스 옵션

- Kinesis 스트림 – 레코드를 읽을 Kinesis 스트림.
- 소비자(선택 사항) – 전용 연결을 통해 스트림에서 읽으려면 스트림 소비자를 사용합니다.
- 배치 크기 – 각 배치에서 샤드로부터 읽을 레코드 수(최대 10,000개). Lambda는 한 번의 호출로 배치의 모든 레코드를 함수에 전달합니다. 단, 이벤트의 총 크기가 페이로드 한도(6 MB)를 초과하지 않아야 합니다.
- 시작 위치 – 새 레코드, 기존의 모든 레코드 또는 특정 날짜 이후에 생성된 레코드만 처리합니다.
  - 최신 – 스트림에 추가된 새 레코드를 처리합니다.
  - 수평 트리밍 – 스트림의 모든 레코드를 처리합니다.
  - 타임스탬프 – 특정 시간에 시작하는 레코드를 시작합니다.

기존 레코드 처리 후 함수는 캐시업되고 새 레코드를 계속 처리합니다.

- 활성화 – 이벤트 소스를 비활성화하여 레코드 처리를 중지합니다. Lambda는 마지막으로 처리된 레코드를 주적하여 다시 활성화되면 그 지점부터 처리를 다시 시작합니다.

나중에 이벤트 소스 구성을 관리하기 위해 디자이너에서 트리거를 선택합니다.

## 이벤트 소스 매핑 API

AWS CLI를 사용하여 이벤트 소스 매핑을 생성하려면 [CreateEventSourceMapping \(p. 343\)](#) API를 사용합니다. 다음 예제는 AWS CLI를 사용하여 my-function 함수를 Kinesis 데이터 스트림에 매핑합니다. 데이터 스트림은 Amazon 리소스 이름(ARN)으로 지정되고, 배치 크기는 500이며 Unix 시간의 타임스탬프부터 시작합니다.

```
$ aws lambda create-event-source-mapping --function-name my-function --no-enabled \
--batch-size 500 --starting-position AT_TIMESTAMP --starting-position-timestamp 1541139109
 \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "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"
}
```

소비자를 사용하려면 스트림의 ARN 대신 소비자의 ARN을 지정합니다. --no-enabled 옵션은 이벤트 소스 매핑을 활성화하지 않고 생성합니다. 활성화하려면 `update-event-source-mapping`을 사용하십시오.

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b1128 --enabled
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b1128",
    "BatchSize": 500,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541190239.996,
    "LastProcessingResult": "No records processed",
    "State": "Enabling",
    "StateTransitionReason": "User action"
}
```

이벤트 소스 매핑을 삭제하려면 `delete-event-source-mapping`을 사용합니다.

```
$ aws lambda delete-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1541190240.0,
    "LastProcessingResult": "No records processed",
    "State": "Deleting",
    "StateTransitionReason": "User action"
}
```

## 실행 역할 권한

Lambda는 Kinesis 데이터 스트림과 관련된 리소스를 관리하기 위해 다음 권한이 필요합니다. 이러한 권한을 함수의 [실행 역할 \(p. 9\)](#)에 추가합니다.

- kinesis:DescribeStream
- kinesis:DescribeStreamSummary
- kinesis:GetRecords
- kinesis:GetShardIterator
- kinesis>ListShards
- kinesis>ListStreams
- kinesis:SubscribeToShard

`AWSLambdaKinesisExecutionRole` 관리형 정책에는 이러한 권한이 포함되어 있습니다. 자세한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.

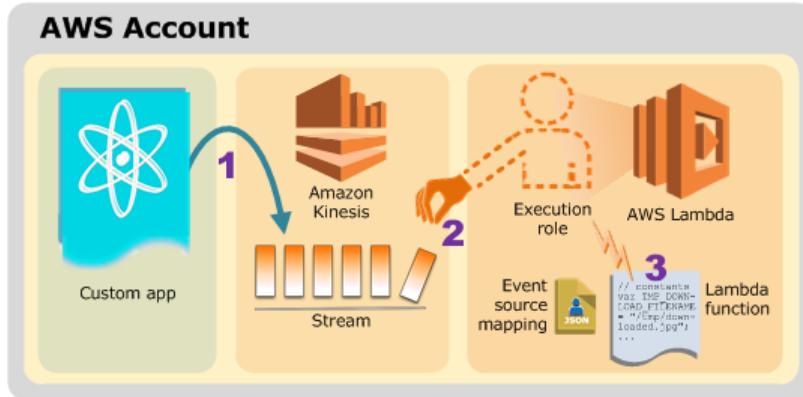
## Amazon CloudWatch 측정치

Lambda는 해당 함수가 한 배치(batch)의 레코드 처리를 완료하면 `IteratorAge` 측정치를 생성합니다. 이 측정치는 처리가 완료되었을 때 배치의 마지막 레코드가 얼마나 오래되었는지를 나타냅니다. 함수가 새 이벤트를 처리하는 경우, 반복기 수명(iterator age)을 사용하여 레코드가 추가된 후 레코드가 처리될 때까지의 지연 시간을 측정할 수 있습니다.

반복기 수명이 증가하는 추세라면 함수에 문제가 있음을 나타낼 수 있습니다. 자세한 내용은 [사용 Amazon CloudWatch \(p. 220\)](#) 단원을 참조하십시오.

## 자습서: Amazon Kinesis과 함께 AWS Lambda 사용

이 자습서에서는 Kinesis 스트림의 이벤트를 사용하기 위해 Lambda 함수를 생성합니다. 다음 디어그램은 애플리케이션 흐름을 보여 줍니다.



1. 사용자 지정 앱은 스트림에 레코드를 기록합니다.
2. AWS Lambda는 스트림에서 새로운 레코드가 감지될 때마다 스트림을 폴링하고 Lambda 함수를 호출합니다.
3. AWS Lambda는 Lambda 함수를 생성할 때 지정한 실행 역할을 가정하여 Lambda 함수를 실행합니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$)와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaKinesisExecutionRole.
  - 역할 이름 – **lambda-kinesis-role**.

AWSLambdaKinesisExecutionRole 정책은 함수가 Kinesis에서 항목을 읽고 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 함수 만들기

다음은 Kinesis 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 함수 코드 \(p. 183\)](#) 단원을 참조하십시오.

Example index.js

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

exports.handler = function(event, context) {
    //console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
};
```

함수를 만들려면

1. 샘플 코드를 index.js 파일에 복사합니다.
2. 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

3. `create-function` 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/lambda-kinesis-role
```

## Lambda 함수 테스트

invoke AWS Lambda CLI 명령 및 샘플 Kinesis 이벤트를 사용하여 Lambda 함수를 수동으로 호출합니다.

Lambda 함수를 테스트하려면

1. 다음 JSON을 파일에 복사하고 input.txt로 저장합니다.

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
                "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-kinesis-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
        }
    ]
}
```

2. invoke 명령을 사용하여 함수로 이벤트를 전송합니다.

```
$ aws lambda invoke --function-name ProcessKinesisRecords --payload file://input.txt
out.txt
```

응답이 out.txt로 저장됩니다.

## Kinesis 스트림 만들기

create-stream 명령을 사용하여 스트림을 만듭니다.

```
$ aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

다음 describe-stream 명령을 실행하여 스트림 ARN을 가져옵니다.

```
$ 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  
}  
}
```

다음 단계에서 스트림 ARN을 사용하여 스트림을 Lambda 함수와 연결합니다.

## AWS Lambda에서 이벤트 소스 추가

다음 AWS CLI add-event-source 명령을 실행합니다.

```
$ 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
```

나중에 사용할 수 있도록 매팅 ID를 기록해둡니다. list-event-source-mappings 명령을 실행하여 이벤트 소스 매팅 목록을 가져올 수 있습니다.

```
$ aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \  
--event-source arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream
```

응답에서 상태 값이 enabled인지 확인할 수 있습니다. 레코드가 일시적으로 손실되는 폴링을 일시 중지하기 위해 이벤트 소스 매팅은 비활성화할 수 있습니다.

## 설정 테스트

이벤트 소스 매팅을 테스트하려면 Kinesis 스트림에 이벤트 레코드를 추가합니다. --data 값은 Kinesis로 전송하기 전에 CLI가 base64로 인코딩하는 문자열입니다. 동일한 명령을 두 번 이상 실행하여 여러 레코드를 스트림에 추가할 수 있습니다.

```
$ aws kinesis put-record --stream-name lambda-stream --partition-key 1 \  
--data "Hello, this is a test."
```

Lambda에서는 실행 역할을 사용하여 스트림에서 레코드를 읽습니다. 그런 다음 Lambda 함수를 호출해 레코드 배치를 전달합니다. 함수는 각 레코드에서 데이터를 디코딩하고 로깅해 CloudWatch Logs에 출력력을 전송합니다. 로그는 [CloudWatch 콘솔](#)에서 확인합니다.

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

### 주제

- [Node.js 8 \(p. 183\)](#)
- [Java 8 \(p. 183\)](#)
- [C# \(p. 184\)](#)
- [Python 3 \(p. 185\)](#)
- [Go \(p. 185\)](#)

## Node.js 8

다음은 Kinesis 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Example index.js

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

exports.handler = function(event, context) {
    //console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Java 8

다음은 입력으로 Kinesis 이벤트 레코드 데이터를 수신하고 처리하는 Java 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

코드에서 recordHandler는 핸들러입니다. 해당 핸들러는 aws-lambda-java-events 라이브러리에서 사전 정의된 KinesisEvent 클래스를 사용합니다.

### 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;
    }
}
```

```
}
```

핸들러가 예외 없이 정상적으로 반환되면 Lambda는 레코드의 입력 배치가 성공적으로 처리된 것으로 간주하고 스트림의 새 레코드 읽기를 시작합니다. 핸들러에 예외가 발생하면 레코드의 입력 배치를 처리하지 않은 것으로 간주하고 동일한 레코드 배치로 함수를 다시 호출합니다.

## 종속성

- aws-lambda-java-core
- aws-lambda-java-events
- aws-java-sdk-s3

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## C#

다음은 입력으로 Kinesis 이벤트 레코드 데이터를 수신하고 처리하는 C# 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

코드에서 `HandleKinesisRecord`는 핸들러입니다. 해당 핸들러는 `Amazon.Lambda.KinesisEvents` 라이브러리에서 사전 정의된 `KinesisEvent` 클래스를 사용합니다.

### 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();
            }
        }
    }
}
```

```
        }
    }
}
```

.NET Core 프로젝트의 `Program.cs`를 위의 샘플로 바꿉니다. 지침은 [.NET Core CLI \(p. 296\)](#) 단원을 참조하십시오.

## Python 3

다음은 입력으로 Kinesis 이벤트 레코드 데이터를 수신하고 처리하는 Python 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 에 기록합니다.

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))
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Go

다음은 입력으로 Kinesis 이벤트 레코드 데이터를 수신하고 처리하는 Go 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 에 기록합니다.

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)
    }
}
```

`go build`를 사용하여 실행 파일을 빌드하고 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지 \(Go\) \(p. 283\)](#) 단원을 참조하십시오.

## Kinesis 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 \(p. 179\)](#)의 Lambda 애플리케이션을 위한 샘플 AWS SAM 템플릿입니다. 이 템플릿의 함수와 핸들러는 Node.js 코드용입니다. 다른 코드 샘플을 사용하는 경우 값을 적절히 업데이트하십시오.

### Example template.yaml - Kinesis 스트림

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  LambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      Timeout: 10
      Tracing: Active
      Events:
        Stream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            BatchSize: 100
            StartingPosition: LATEST
  KinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
Outputs:
  FunctionName:
    Description: "Function name"
    Value: !Ref LambdaFunction
  StreamARN:
    Description: "Stream ARN"
    Value: !GetAtt KinesisStream.Arn
```

템플릿에서 Lambda 함수, Kinesis 스트림, 이벤트 소스 매핑을 생성합니다. 이벤트 소스 매핑이 스트림에서 값을 읽고 함수를 호출합니다.

[HTTP/2 스트림 소비자 \(p. 176\)](#)를 사용하려면 템플릿에서 소비자를 만들고, 스트림 대신 소비자에서 값을 읽도록 이벤트 소스 매핑을 구성합니다.

### Example template.yaml - Kinesis 스트림 소비자

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A function that processes data from a Kinesis stream.
Resources:
  kinesisprocessrecordpython:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      Timeout: 10
      Tracing: Active
      Events:
        Stream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt StreamConsumer.ConsumerARN
            StartingPosition: LATEST
            BatchSize: 100
  KinesisStream:
    Type: "AWS::Kinesis::Stream"
    Properties:
      ShardCount: 1
  StreamConsumer:
    Type: "AWS::Kinesis::StreamConsumer"
    Properties:
```

```
StreamARN: !GetAtt KinesisStream.Arn
ConsumerName: "TestConsumer"
Outputs:
  FunctionName:
    Description: "Function name"
    Value: !Ref LambdaFunction
  StreamARN:
    Description: "Stream ARN"
    Value: !GetAtt KinesisStream.Arn
  ConsumerARN:
    Description: "Stream consumer ARN"
    Value: !GetAtt StreamConsumer.ConsumerARN
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

## Using AWS Lambda with Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose는 스트리밍 데이터를 캡처하고 변환하여 Kinesis Data Analytics 또는 Amazon S3 등과 같은 다운스트림 서비스에 로드합니다. 다운스트림으로 보내기 전에 함수를 작성하여 추가로 사용자 지정된 데이터 처리를 요청할 수 있습니다.

Example Amazon Kinesis Data Firehose 메시지 이벤트

```
{
  "invocationId": "invoked123",
  "deliveryStreamArn": "aws:lambda:events",
  "region": "us-west-2",
  "records": [
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record1",
      "approximateArrivalTimestamp": 1510772160000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000000",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c317a",
        "approximateArrivalTimestamp": "2012-04-23T18:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785154",
        "subsequenceNumber": ""
      }
    },
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record2",
      "approximateArrivalTimestamp": 1510772160000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000001",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",
        "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",
        "subsequenceNumber": ""
      }
    }
  ]
}
```

자세한 내용은 Kinesis Data Firehose 개발자 안내서의 [Amazon Kinesis Data Firehose 데이터 변환](#)을 참조하십시오.

## Using AWS Lambda with Amazon Lex

Amazon Lex는 음성과 텍스트를 사용하는 애플리케이션에 대화형 인터페이스를 구축하는 AWS 서비스입니다. Amazon Lex는 AWS Lambda와의 사전 빌드 통합을 제공하여 Amazon Lex 봇과 코드 후크로 사용할 Lambda 함수를 생성할 수 있습니다. 사용자 의도 구성에서 초기화/검증, 이행 또는 두 가지 모두를 수행할 함수를 식별할 수 있습니다.

### Example Amazon Lex메시지 이벤트

```
{  
    "messageVersion": "1.0",  
    "invocationSource": "FulfillmentCodeHook",  
    "userId": "ABCD1234",  
    "sessionAttributes": {  
        "key1": "value1",  
        "key2": "value2",  
    },  
    "bot": {  
        "name": "my-bot",  
        "alias": "prod",  
        "version": "1"  
    },  
    "outputDialogMode": "Text",  
    "currentIntent": {  
        "name": "my-intent",  
        "slots": {  
            "slot-name": "value",  
            "slot-name": "value",  
            "slot-name": "value"  
        },  
        "confirmationStatus": "Confirmed"  
    }  
}
```

자세한 내용은 [Lambda 함수 사용을 참조하십시오](#). 사용 사례는 [실습 1: 블루프린트를 사용하여 Amazon Lex 봇 생성](#)을 참조하십시오.

## Using AWS Lambda with Amazon S3

Amazon S3는 이벤트를 AWS Lambda에 게시하고(예: 객체가 버킷에 생성될 때) 이벤트 데이터를 파라미터로 전달하여 Lambda 함수를 호출할 수 있습니다. 이 통합을 사용하면 Amazon S3 이벤트를 처리하는 Lambda 함수를 작성할 수 있습니다. Amazon S3에서, Amazon S3가 게시할 이벤트 유형 및 호출할 Lambda 함수를 식별하는 버킷 알림 구성을 추가합니다.

### Important

Lambda 함수가 함수를 트리거하는 동일한 버킷을 사용하는 경우 함수가 루프에서 실행되게 만들 수 있습니다. 예를 들어 객체가 업로드될 때마다 버킷이 함수를 트리거하고 그 함수가 객체를 버킷에 업로드하는 경우, 함수는 간접적으로 자신을 트리거합니다. 이렇게 되지 않도록 하려면 두 개의 버킷을 사용하거나, 수신 객체에 사용되는 접두사에만 적용되도록 트리거를 구성합니다.

### Example Amazon S3메시지 이벤트

```
{  
    "Records": [  
        {  
            "eventVersion": "2.0",  
            "eventSource": "aws:s3",  
            "eventSourceARN": "arn:aws:s3:::my-bucket",  
            "awsRegion": "us-east-1",  
            "s3": {  
                "object": {  
                    "size": 1024,  
                    "eTag": "01234567890123456789012345678901",  
                    "sequencer": "0A1B2C3D4E5F6G7H8I9J0K1L2M3N4O5P6Q7R8S9T0U1V2W3X4Y5Z6",  
                    "key": "image.jpg",  
                    "type": "ObjectCreated:Put"  
                }  
            }  
        }  
    ]  
}
```

```
"eventVersion": "2.0",
"eventSource": "aws:s3",
"awsRegion": "us-west-2",
"eventTime": "1970-01-01T00:00:00.000Z",
"eventName": "ObjectCreated:Put",
"userIdentity": {
    "principalId": "AIDAJDPLRKLG7UEXAMPLE"
},
"requestParameters": {
    "sourceIPAddress": "127.0.0.1"
},
"responseElements": {
    "x-amz-request-id": "C3D13FE58DE4C810",
    "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
},
"s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "testConfigRule",
    "bucket": {
        "name": "sourcebucket",
        "ownerIdentity": {
            "principalId": "A3NL1KOZZKExample"
        },
        "arn": "arn:aws:s3:::sourcebucket"
    },
    "object": {
        "key": "HappyFace.jpg",
        "size": 1024,
        "eTag": "d41d8cd98f00b204e9800998ecf8427e",
        "versionId": "096fKKXTRTt13on89fVO.nfljtsv6qko"
    }
}
]
```

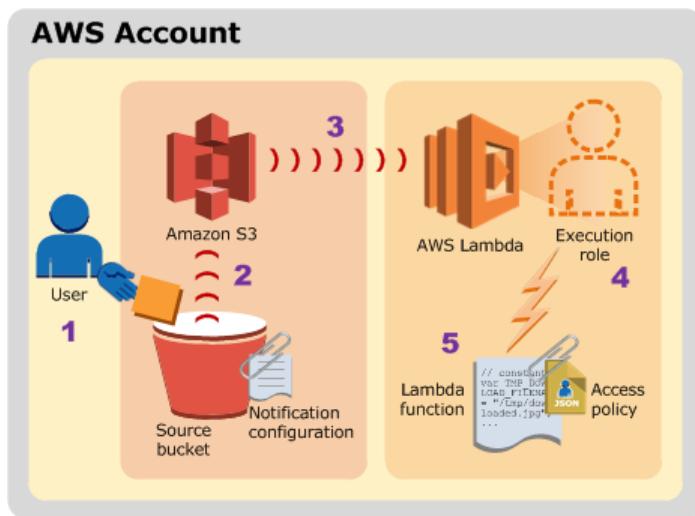
Amazon S3 및 AWS Lambda 통합이 작동하는 방법에 대해 다음 사항을 참고합니다.

- **비 스트림 기반(비동기식) 모델** – Amazon S3가 버킷을 모니터링하고 이벤트 데이터를 파라미터로 전달하여 Lambda 함수를 호출하는 모델입니다([AWS Lambda 이벤트 소스 매핑 \(p. 80\)](#) 참조). 푸시 모델에서는 버킷 알림 구성을 사용하여 내에서 이벤트 소스 매핑을 유지합니다. 구성에서 Amazon S3가 모니터링 할 이벤트 유형과 Amazon S3가 호출할 AWS Lambda 함수를 지정합니다. 자세한 내용은 [Amazon Simple Storage Service 개발자 가이드의 Amazon S3 이벤트 알림 구성](#)을 참조하십시오.
- **비동기식 호출** – AWS Lambda는 Event 호출 유형(비동기식 호출)을 사용하여 Lambda 함수를 호출합니다. 호출 유형에 대한 자세한 내용은 [호출 유형 \(p. 80\)](#) 단원을 참조하십시오.
- **이벤트 구조** – Lambda 함수가 수신하는 이벤트는 단일 객체에 대한 이벤트이며 버킷 이름 및 객체 키 이름과 같은 정보를 제공합니다.

종합적인 경험을 설정하는 데 사용하는 권한 정책은 두 가지 유형이 있습니다.

- **Lambda 함수에 대한 권한** – Lambda 함수를 호출하는 것과 관계없이 AWS Lambda는 Lambda 함수를 만들 때 지정하는 IAM 역할(실행 역할)을 가정하여 함수를 실행합니다. 이 역할과 연결된 권한 정책을 사용하여 함수에 필요한 권한을 부여합니다. 예를 들어, Lambda 함수가 객체를 읽어야 하는 경우, 권한 정책에서 관련 Amazon S3 작업에 대한 권한을 부여합니다. 자세한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.
- **Amazon S3가 Lambda 함수를 호출할 수 있는 권한** – Amazon S3는 사용자의 허가 없이 Lambda 함수를 호출할 수 없습니다. 사용자는 함수와 연결된 권한 정책을 통해 이 권한을 부여합니다.

다음 다이어그램은 해당 흐름을 보여 줍니다.



1. 사용자가 S3 버킷(객체 생성 이벤트)에 객체를 업로드합니다.
2. Amazon S3가 객체 생성 이벤트를 감지합니다.
3. Amazon S3는 버킷 알림 구성에 지정된 Lambda 함수를 호출합니다.
4. AWS Lambda는 Lambda 함수를 생성할 때 지정한 실행 역할을 가정하여 Lambda 함수를 실행합니다.
5. Lambda 함수가 실행됩니다.

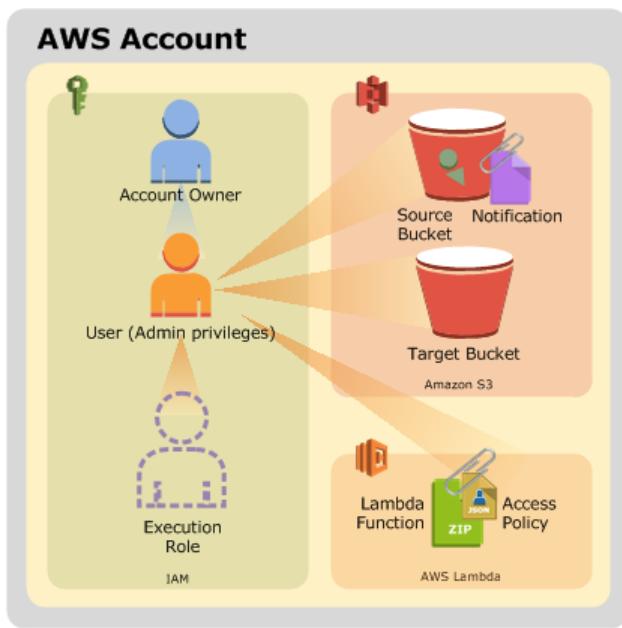
#### 주제

- [자습서: Amazon S3과 함께 AWS Lambda 사용 \(p. 190\)](#)
- [샘플 Amazon Simple Storage Service 함수 코드 \(p. 197\)](#)
- [Amazon S3 애플리케이션을 위한 AWS SAM 템플릿 \(p. 203\)](#)

## 자습서: Amazon S3과 함께 AWS Lambda 사용

버킷에 업로드된 각 이미지 파일에 대해 썸네일을 만들어 보겠습니다. 객체 생성 시 Amazon S3가 호출할 수 있는 Lambda 함수(`CreateThumbnail`)를 생성할 수 있습니다. 그런 다음 Lambda 함수 코드는 소스 버킷에서 이미지 객체를 읽고 썸네일 이미지 대상 버킷을 생성할 수 있습니다.

이 자습서를 완료하면 계정에 다음과 같은 Amazon S3, Lambda 및 IAM 리소스가 생성됩니다.



### Lambda 리소스

- Lambda 함수.
- Lambda 함수에 연결된 액세스 정책은 Lambda 함수를 호출하는 Amazon S3 권한을 부여합니다.

### IAM 리소스

- 이 역할과 연결된 권한 정책을 통해 Lambda 함수에 필요한 권한을 부여하는 실행 역할입니다.

### Amazon S3 리소스

- Lambda 함수를 호출하는 알림 구성을 포함하는 소스 버킷입니다.
- 함수가 크기 변경된 이미지를 저장하는 대상 버킷입니다.

## 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$)와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

NPM 설치하여 함수의 종속 항목을 관리합니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaExecute.
  - 역할 이름 – **lambda-s3-role**.

AWSLambdaExecute 정책은 함수가 Amazon S3의 객체를 관리하고 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 버킷 생성 및 샘플 객체 업로드

다음 단계에 따라 버킷을 만들고 객체를 업로드합니다.

1. [Amazon S3 콘솔](#)을 엽니다.
2. 두 개의 버킷을 만듭니다. 대상 버킷 이름은 **source** 다음에 **resized**가 와야 하고, 여기에서 **source**는 소스에 사용할 버킷의 이름입니다. 예: mybucket, mybucketresized.
3. 원본 버킷에서 .jpg 객체 HappyFace.jpg를 업로드합니다.

Amazon S3에 연결하기 전에 Lambda 함수를 수동으로 호출하면, 원본 버킷을 지정하는 함수에 샘플 이벤트 데이터를 전달하고 HappyFace.jpg를 새로 생성한 객체로 전달하므로 이 샘플 객체를 먼저 생성해야 합니다.

## 함수 만들기

다음은 Amazon S3 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이 코드는 소스 버킷의 이미지 크기를 변경하고 출력을 대상 버킷에 저장합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 Amazon Simple Storage Service 함수 코드 \(p. 197\)](#) 단원을 참조하십시오.

Example index.js

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();

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

```
// Read options from the event.
console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
var srcBucket = event.Records[0].s3.bucket.name;
// Object key may have spaces or unicode non-ASCII characters.
var srcKey    =
decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
var dstBucket = srcBucket + "resized";
var dstKey    = "resized-" + srcKey;

// Sanity check: validate that source and destination are different buckets.
if (srcBucket == dstBucket) {
    callback("Source and destination buckets are the same.");
    return;
}

// Infer the image type.
var typeMatch = srcKey.match(/^(.*\.(?:jpg|png))$/);
if (!typeMatch) {
    callback("Could not determine the image type.");
    return;
}
var imageType = typeMatch[1];
if (imageType != "jpg" && imageType != "png") {
    callback('Unsupported image type: ${imageType}');
    return;
}

// Download the image from S3, transform, and upload to a different S3 bucket.
async.waterfall([
    function download(next) {
        // Download the image from S3 into a buffer.
        s3.getObject({
            Bucket: srcBucket,
            Key: srcKey
        },
        next);
    },
    function transform(response, next) {
        gm(response.Body).size(function(err, size) {
            // Infer the scaling factor to avoid stretching the image unnaturally.
            var scalingFactor = Math.min(
                MAX_WIDTH / size.width,
                MAX_HEIGHT / size.height
            );
            var width  = scalingFactor * size.width;
            var height = scalingFactor * size.height;

            // Transform the image buffer in memory.
            this.resize(width, height)
                .toBuffer(imageType, function(err, buffer) {
                    if (err) {
                        next(err);
                    } else {
                        next(null, response.ContentType, buffer);
                    }
                });
        });
    },
    function upload(contentType, data, next) {
        // Stream the transformed image to a different S3 bucket.
        s3.putObject({
            Bucket: dstBucket,
            Key: dstKey,
            Body: data,
            ContentType: contentType
        },
        next);
    }
],
```

```
        next);
    }
], function (err) {
  if (err) {
    console.error(
      'Unable to resize ' + srcBucket + '/' + srcKey +
      ' and upload to ' + dstBucket + '/' + dstKey +
      ' due to an error: ' + err
    );
  } else {
    console.log(
      'Successfully resized ' + srcBucket + '/' + srcKey +
      ' and uploaded to ' + dstBucket + '/' + dstKey
    );
  }
}

callback(null, "message");
);
);
};
```

앞의 코드를 검토하고 다음 사항을 확인합니다.

- 이 함수는 파라미터로 받은 이벤트 데이터에서 원본 버킷 이름과 객체 키 이름을 알고 있습니다. 객체가 jpg인 경우, 코드는 썸네일을 만들고 대상 버킷에 저장합니다.
- 이 코드는 대상 버킷이 존재하고 해당 이름이 원본 버킷 이름과 `resized` 문자열이 이어진 것이라고 가정합니다. 예를 들어 이벤트 데이터에서 식별된 원본 버킷이 `examplebucket`인 경우 코드는 사용자에게 `examplebucketresized` 대상 버킷이 있다고 가정합니다.
- 생성하는 썸네일의 경우 코드는 `resized-` 문자열과 소스 객체 키 이름이 연결된 키 이름을 가져옵니다. 예를 들어 소스 객체 키가 `sample.jpg`인 경우 코드는 키가 `resized-sample.jpg`인 썸네일 객체를 만듭니다.

배포 패키지는 Lambda 함수 코드 및 종속성이 포함되어 있는 `.zip` 파일입니다.

배포 패키지를 만들려면

1. 함수 코드를 `lambda-s3` 폴더에 `index.js`로 저장합니다.
2. NPM을 사용하여 GraphicsMagick과 Async 라이브러리를 설치합니다.

```
lambda-s3$ npm install async gm
```

이 단계를 완료한 후에는 다음과 같은 폴더 구조를 갖게 됩니다.

```
lambda-s3
|- index.js
|- /node_modules/gm
# /node_modules/async
```

3. 함수 코드와 종속 항목을 포함하는 배포 패키지를 생성합니다.

```
lambda-s3$ zip -r function.zip .
```

함수를 만들려면

- `create-function` 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name CreateThumbnail \
```

```
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

위의 명령은 함수 구성으로 10초의 제한 시간 값을 지정합니다. 업로드하는 객체의 크기에 따라 다음 AWS CLI 명령을 사용하여 제한 시간 값을 늘려야 할 수도 있습니다.

```
$ aws lambda update-function-configuration --function-name CreateThumbnail --timeout 30
```

## Lambda 함수 테스트

이 단계에서는 샘플 Amazon S3 이벤트 데이터를 사용하여 Lambda 함수를 수동으로 호출합니다.

### Lambda 함수를 테스트하려면

1. 다음 Amazon S3 샘플 이벤트 데이터를 파일에 저장하고 이 파일을 `inputFile.txt`로 저장합니다. `sourcebucket` 이름 및 `.jpg` 객체 키를 제공하여 JSON을 업데이트해야 합니다.

```
{
    "Records": [
        {
            "eventVersion": "2.0",
            "eventSource": "aws:s3",
            "awsRegion": "us-west-2",
            "eventTime": "1970-01-01T00:00:00.000Z",
            "eventName": "ObjectCreated:Put",
            "userIdentity": {
                "principalId": "AIDAJDPLRKLG7UEXAMPLE"
            },
            "requestParameters": {
                "sourceIPAddress": "127.0.0.1"
            },
            "responseElements": {
                "x-amz-request-id": "C3D13FE58DE4C810",
                "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
            },
            "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "testConfigRule",
                "bucket": {
                    "name": "sourcebucket",
                    "ownerIdentity": {
                        "principalId": "A3NL1KOZZKExample"
                    },
                    "arn": "arn:aws:s3:::sourcebucket"
                },
                "object": {
                    "key": "HappyFace.jpg",
                    "size": 1024,
                    "eTag": "d41d8cd98f00b204e9800998ecf8427e",
                    "versionId": "096fKKXTRTl3on89fvO.nfljtsv6qko"
                }
            }
        }
    ]
}
```

2. 다음과 같은 Lambda invoke 명령을 실행하여 함수를 호출합니다. 이 명령은 비동기 실행을 요청합니다. 선택에 따라 RequestResponse를 invocation-type 파라미터 값으로 지정하여 동기식으로 함수를 호출할 수 있습니다.

```
$ aws lambda invoke --function-name CreateThumbnail --invocation-type Event \
--payload file://inputfile.txt outputfile.txt
```

3. 대상 버킷에 썸네일이 생성되었는지 확인합니다.

## 이벤트를 게시하도록 Amazon S3 구성

이 단계에서는 Amazon S3가 객체 생성 이벤트를 AWS Lambda에 게시하고 Lambda 함수를 호출하게 하도록 나머지 구성을 추가합니다. 이 단계에서는 다음 작업을 수행합니다.

- Lambda 함수의 액세스 정책에 권한을 추가하여 Amazon S3가 함수를 호출하도록 허용합니다.
- 원본 버킷에 알림 구성은 추가합니다. 알림 구성에서 다음을 제공합니다.
  - Amazon S3가 게시하게 하려는 이벤트 유형. 이 자습서에서는 객체가 생성될 때 Amazon S3가 이벤트를 게시할 수 있도록 s3:ObjectCreated:\* 이벤트 유형을 지정합니다.
  - 호출할 Lambda 함수.

함수 정책에 권한을 추가하려면

1. 다음 Lambda CLI add-permission 명령을 사용하여 Amazon S3 서비스 보안 주체 (`s3.amazonaws.com`)에게 `lambda:InvokeFunction` 작업을 수행할 수 있는 권한을 부여합니다. 다음 조건이 충족되는 경우에만 Amazon S3에 함수를 호출할 수 있는 권한이 부여됩니다.
  - 객체 생성 이벤트가 특정 버킷에서 감지됩니다.
  - 해당 버킷은 특정 AWS 계정의 소유입니다. 버킷 소유자가 버킷을 삭제하면 다른 AWS 계정이 동일한 이름의 버킷을 생성할 수 있습니다. 이 조건을 통해 특정 AWS 계정만 Lambda 함수를 호출할 수 있습니다.

```
$ aws lambda add-permission --function-name CreateThumbnail --principal
s3.amazonaws.com \
--statement-id some-unique-id --action "lambda:InvokeFunction" \
--source-arn arn:aws:s3:::sourcebucket \
--source-account bucket-owner-account-id
```

2. AWS CLI get-policy 명령을 실행하여 함수의 액세스 정책을 확인합니다.

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

원본 버킷에 알림 구성은 추가하여 Amazon S3에 객체 생성 이벤트를 Lambda에 게시하도록 요청합니다.

알림을 구성하려면

1. [Amazon S3 콘솔](#)을 엽니다.
2. 소스 버킷을 선택합니다.
3. 속성을 선택합니다.
4. 이벤트에서 다음과 같은 설정으로 알림을 구성합니다.
  - 이름 – **lambda-trigger**.
  - 이벤트 – **ObjectCreate (All)**.
  - 전송 대상 – **Lambda function**.
  - Lambda – **CreateThumbnail**.

이벤트 구성에 대한 자세한 내용은 Amazon Simple Storage Service 콘솔 사용 설명서의 [이벤트 알림 활성화](#)를 참조하십시오.

## 설정 테스트

이제 다음과 같이 설정을 테스트할 수 있습니다.

1. Amazon S3 콘솔을 사용하여 .jpg 또는 .png 객체를 원본 버킷에 업로드합니다.
2. CreateThumbnail 함수를 사용하여 대상 버킷에 썸네일이 생성되었는지 확인합니다.
3. CloudWatch 콘솔에서 로그를 봅니다.

## 샘플 Amazon Simple Storage Service 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

주제

- [Node.js 8 \(p. 197\)](#)
- [Java 8 \(p. 199\)](#)
- [Python 3 \(p. 202\)](#)

### Node.js 8

다음은 Amazon S3 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이 코드는 소스 버킷의 이미지 크기를 변경하고 출력을 대상 버킷에 저장합니다.

Example index.js

```
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

var s3 = new AWS.S3();

exports.handler = function(event, context, callback) {
    // Read options from the event.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    var srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    var srcKey    =
        decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    var dstBucket = srcBucket + "resized";
    var dstKey    = "resized-" + srcKey;

    // Sanity check: validate that source and destination are different buckets.
    if (srcBucket == dstBucket) {
        callback("Source and destination buckets are the same.");
        return;
    }

    // Infer the image type.
    var typeMatch = srcKey.match(/\.([^.]*$)/);
    if (!typeMatch) {
        callback("Could not determine the image type.");
    }
}
```

```
        return;
    }
    var imageType = typeMatch[1];
    if (imageType != "jpg" && imageType != "png") {
        callback('Unsupported image type: ${imageType}');
        return;
    }

    // Download the image from S3, transform, and upload to a different S3 bucket.
    async.waterfall([
        function download(next) {
            // Download the image from S3 into a buffer.
            s3.getObject({
                Bucket: srcBucket,
                Key: srcKey
            },
            next);
        },
        function transform(response, next) {
            gm(response.Body).size(function(err, size) {
                // Infer the scaling factor to avoid stretching the image unnaturally.
                var scalingFactor = Math.min(
                    MAX_WIDTH / size.width,
                    MAX_HEIGHT / size.height
                );
                var width = scalingFactor * size.width;
                var height = scalingFactor * size.height;

                // Transform the image buffer in memory.
                this.resize(width, height)
                    .toBuffer(imageType, function(err, buffer) {
                        if (err) {
                            next(err);
                        } else {
                            next(null, response.ContentType, buffer);
                        }
                    });
            });
        },
        function upload(contentType, data, next) {
            // Stream the transformed image to a different S3 bucket.
            s3.putObject({
                Bucket: dstBucket,
                Key: dstKey,
                Body: data,
                ContentType: contentType
            },
            next);
        }
    ], function (err) {
        if (err) {
            console.error(
                'Unable to resize ' + srcBucket + '/' + srcKey +
                ' and upload to ' + dstBucket + '/' + dstKey +
                ' due to an error: ' + err
            );
        } else {
            console.log(
                'Successfully resized ' + srcBucket + '/' + srcKey +
                ' and uploaded to ' + dstBucket + '/' + dstKey
            );
        }
        callback(null, "message");
    });
});
```

```
};
```

배포 패키지는 Lambda 함수 코드 및 종속성이 포함되어 있는 .zip 파일입니다.

배포 패키지를 만들려면

1. 폴더(examplefolder)를 생성한 다음 하위 폴더(node\_modules)를 생성합니다.
2. Node.js 플랫폼을 설치합니다. 자세한 내용은 [Node.js 웹 사이트](#)를 참조하십시오.
3. 종속 항목을 설치합니다. 이 코드 예제는 다음 라이브러리를 사용합니다.
  - Node.js로 작성된 JavaScript용 AWS SDK
  - gm, GraphicsMagick(node.js용)
  - 비동기 유틸리티 모듈

AWS Lambda 런타임에는 Node.js에 이미 JavaScript용 AWS SDK가 있으므로 다른 라이브러리만 설치하면 됩니다. 명령 프롬프트를 열고 examplefolder로 이동한 후 Node.js의 일부인 npm 명령을 사용하여 라이브러리를 설치합니다.

```
$ npm install async gm
```

4. 샘플 코드를 index.js 파일에 저장합니다.
5. 앞의 코드를 검토하고 다음 사항을 확인합니다.
  - 이 함수는 파라미터로 받은 이벤트 데이터에서 원본 버킷 이름과 객체 키 이름을 알고 있습니다. 객체가 .jpg인 경우, 코드는 썸네일을 만들고 대상 버킷에 저장합니다.
  - 이 코드는 대상 버킷이 존재하고 해당 이름이 원본 버킷 이름과 resized 문자열이 이어진 것이라고 가정합니다. 예를 들어 이벤트 데이터에서 식별된 원본 버킷이 examplebucket인 경우 코드는 사용자에게 examplebucketresized 대상 버킷이 있다고 가정합니다.
  - 생성하는 썸네일의 경우 코드는 resized- 문자열과 소스 객체 키 이름이 연결된 키 이름을 가져옵니다. 예를 들어 소스 객체 키가 sample.jpg인 경우 코드는 키가 resized-sample.jpg인 썸네일 객체를 만듭니다.
6. 파일을 index.js에 examplefolder로 저장합니다. 이 단계를 완료한 후에는 다음과 같은 폴더 구조를 갖게 됩니다.

```
index.js
/node_modules/gm
/node_modules/async
```

7. index.js 파일과 node\_modules 폴더를 CreateThumbnail.zip으로 압축합니다.

## Java 8

다음은 수신하는 Amazon S3 이벤트를 읽고 썸네일을 생성하는 Java 코드의 예입니다. 이는 aws-lambda-java-core 라이브러리에 제공된 RequestHandler 인터페이스를 구현합니다. 따라서 Lambda 함수를 만들 때 클래스를 핸들러(즉, example.S3EventProcessorCreateThumbnail)로 지정합니다. 인터페이스를 사용하여 핸들러를 제공하는 방법에 대한 자세한 내용은 [핸들러를 생성하기 위해 사전 정의된 인터페이스 활용\(Java\) \(p. 271\)](#) 단원을 참조하십시오.

핸들러가 입력 유형으로 사용하는 S3Event 유형은 수신하는 Amazon S3 이벤트에서 정보를 쉽게 읽을 수 있는 방법을 제공하는 aws-lambda-java-events 라이브러리의 사전 정의된 클래스 중 하나입니다. 핸들러는 문자열을 출력으로 반환합니다.

### Example S3EventProcessorCreateThumbnail.java

```
package example;
```

```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLDecoder;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.event.S3EventNotification.S3EventNotificationRecord;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3Object;

public class S3EventProcessorCreateThumbnail implements
    RequestHandler<S3Event, String> {
    private static final float MAX_WIDTH = 100;
    private static final float MAX_HEIGHT = 100;
    private final String JPG_TYPE = (String) "jpg";
    private final String JPG_MIME = (String) "image/jpeg";
    private final String PNG_TYPE = (String) "png";
    private final String PNG_MIME = (String) "image/png";

    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);

            String srcBucket = record.getS3().getBucket().getName();
            // Object key may have spaces or unicode non-ASCII characters.
            String srcKey = record.getS3().getObject().getKey()
                .replace('+', ' ');
            srcKey = URLDecoder.decode(srcKey, "UTF-8");

            String dstBucket = srcBucket + "resized";
            String dstKey = "resized-" + srcKey;

            // Sanity check: validate that source and destination are different
            // buckets.
            if (srcBucket.equals(dstBucket)) {
                System.out
                    .println("Destination bucket must not match source bucket.");
                return "";
            }

            // Infer the image type.
            Matcher matcher = Pattern.compile(".*\\.(\\[^\\.\\.]*)").matcher(srcKey);
            if (!matcher.matches()) {
                System.out.println("Unable to infer image type for key "
                    + srcKey);
                return "";
            }
            String imageType = matcher.group(1);
            if (!(JPG_TYPE.equals(imageType)) && !(PNG_TYPE.equals(imageType))) {
                System.out.println("Skipping non-image " + srcKey);
                return "";
            }
        }
    }
}
```

```

        // Download the image from S3 into a stream
        AmazonS3 s3Client = new AmazonS3Client();
        S3Object s3Object = s3Client.getObject(new GetObjectRequest(
                srcBucket, srcKey));
        InputStream objectData = s3Object.getObjectContent();

        // Read the source image
        BufferedImage srcImage = ImageIO.read(objectData);
        int srcHeight = srcImage.getHeight();
        int srcWidth = srcImage.getWidth();
        // Infer the scaling factor to avoid stretching the image
        // unnaturally
        float scalingFactor = Math.min(MAX_WIDTH / srcWidth, MAX_HEIGHT
                / srcHeight);
        int width = (int) (scalingFactor * srcWidth);
        int height = (int) (scalingFactor * srcHeight);

        BufferedImage resizedImage = new BufferedImage(width, height,
                BufferedImage.TYPE_INT_RGB);
        Graphics2D g = resizedImage.createGraphics();
        // Fill with white before applying semi-transparent (alpha) images
        g.setPaint(Color.white);
        g.fillRect(0, 0, width, height);
        // Simple bilinear resize
        g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        g.drawImage(srcImage, 0, 0, width, height, null);
        g.dispose();

        // Re-encode image to target format
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        ImageIO.write(resizedImage, imageType, os);
        InputStream is = new ByteArrayInputStream(os.toByteArray());
        // Set Content-Length and Content-Type
        ObjectMetadata meta = new ObjectMetadata();
        meta.setContentLength(os.size());
        if (JPG_TYPE.equals(imageType)) {
            meta.setContentType(JPG_MIME);
        }
        if (PNG_TYPE.equals(imageType)) {
            meta.setContentType(PNG_MIME);
        }

        // Uploading to S3 destination bucket
        System.out.println("Writing to: " + dstBucket + "/" + dstKey);
        s3Client.putObject(dstBucket, dstKey, is, meta);
        System.out.println("Successfully resized " + srcBucket + "/"
                + srcKey + " and uploaded to " + dstBucket + "/" + dstKey);
        return "Ok";
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

```

Amazon S3는 Event 호출 유형을 사용해 Lambda 함수를 호출하며, 여기에서 AWS Lambda는 코드를 비동기식으로 실행합니다. 반환하는 항목은 중요하지 않습니다. 하지만 이 경우 반환 유형을 지정해야 하는 인터페이스를 구현하므로 이 예제에서는 핸들러가 `String`을 반환 유형으로 사용합니다.

종속성

- `aws-lambda-java-core`

- aws-lambda-java-events
- aws-java-sdk-s3

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## Python 3

다음은 Amazon S3 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이 코드는 소스 버킷의 이미지 크기를 변경하고 출력을 대상 버킷에 저장합니다.

Example CreateThumbnail.py

```
import boto3
import os
import sys
import uuid
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
        upload_path = '/tmp/resized-{}'.format(key)

        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}resized'.format(bucket), key)
```

배포 패키지를 만들려면

1. 샘플 코드를 CreateThumbnail.py 파일에 복사합니다.
2. 가상 환경을 생성합니다.

```
$ virtualenv ~/shrink_venv
$ source ~/shrink_venv/bin/activate
3. 가상 환경에 라이브러리를 설치합니다.

$ pip install Pillow
$ pip install boto3
4. lib 및 lib64 사이트 패키지 컨텐츠를 .zip 파일에 추가합니다.

$ cd $VIRTUAL_ENV/lib/python3.7/site-packages
$ zip -r ~/CreateThumbnail.zip .
5. python 코드를 .zip 파일에 추가합니다.

$ cd ~
```

```
$ zip -g CreateThumbnail.zip CreateThumbnail.py
```

## Amazon S3 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 \(p. 190\)](#)의 Lambda 애플리케이션을 위한 샘플 AWS SAM 템플릿입니다. 아래 텍스트를 .yaml 파일로 복사하고 이전에 만든 ZIP 패키지 옆에 저장합니다. Handler 및 Runtime 파라미터 값은 이전 단원에서 함수를 생성할 때 사용한 것과 일치해야 합니다.

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
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

## Using AWS Lambda with Amazon SES

Amazon SES를 사용하여 메시지를 받으면 메시지가 도착할 때 Lambda 함수를 호출하도록 Amazon SES를 구성할 수 있습니다. 그런 다음 서비스는 수신되는 이메일 이벤트를 전달하여 Lambda 함수를 호출할 수 있습니다. 여기에서 실제 수신되는 이메일 이벤트는 Amazon SNS 이벤트의 Amazon SES 메시지인 파라미터로 전달됩니다.

Example Amazon SES메시지 이벤트

```
{
  "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=jX3F0bCAI7s1bkHyy3mLYO28ieDQz2R0P8HwQkklFj4=; 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"
]
}
```

자세한 내용은 Amazon SES 개발자 가이드의 [Lambda 작업](#)을 참조하십시오.

## Using AWS Lambda with Amazon SNS

Amazon Simple Notification Service 알림을 처리하도록 Lambda 함수를 작성할 수 있습니다. Amazon SNS 주제에 메시지가 게시되면 해당 서비스는 메시지 페이로드를 파라미터로 전달하여 Lambda 함수를 호출할 수 있습니다. 그런 다음 Lambda 함수 코드는 이벤트를 처리할 수 있습니다(예: 다른 Amazon SNS 주제에 메시지 게시 또는 다른 AWS 서비스로 메시지 전송).

Lambda 함수가 구독되는 주제에 대해 사용자가 SNS Publish API를 호출하면 Amazon SNS가 Lambda를 호출하여 함수를 비동기식으로 호출합니다. 그러면 Lambda에서 전송 상태를 반환합니다. Lambda를 호출하는 중 오류가 발생하면 Amazon SNS는 Lambda 함수를 최대 세 번 호출하려고 시도합니다. 세 번 시도한 후 Amazon SNS가 Lambda 함수를 성공적으로 호출할 수 없으면 Amazon SNS는 CloudWatch로 전송 상태 실패 메시지를 보냅니다.

Lambda에 교차 계정 Amazon SNS 전송을 수행하려면 Amazon SNS에서 호출할 Lambda 함수를 승인해야 합니다. 이에 대해 Amazon SNS는 Lambda 계정이 Amazon SNS 주제에 등록할 수 있게 허용해야 합니다. 예를 들어, Amazon SNS 주제가 계정 A에 있고 Lambda 함수가 계정 B에 있으면 두 계정 모두 다른 리소스에 대한 액세스 권한을 다른 계정에 부여해야 합니다. 교차 계정 권한 설정을 위한 모든 옵션을 AWS 콘솔에서 사용할 수 있는 것은 아니기 때문에, AWS CLI를 사용하여 전체 프로세스를 설정합니다.

### Example Amazon SNS메시지 이벤트

```
{
    "Records": [
        {
            "EventVersion": "1.0",
            "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:test-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
            "EventSource": "aws:sns",
            "Sns": {
                "SignatureVersion": "1",
                "Timestamp": "1970-01-01T00:00:00.000Z",
                "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEka16RibDsvpi+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": topicarn,
"Subject": "TestInvoke"
}
]
```

주제 구독 구성은 Amazon SNS에서 이벤트 소스 매핑을 구성합니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 [Amazon SNS 알림을 사용하여 Lambda 함수 호출](#)을 참조하십시오.

#### 주제

- [자습서: Amazon Simple Notification Service과 함께 AWS Lambda 사용 \(p. 206\)](#)
- [샘플 함수 코드 \(p. 209\)](#)

## 자습서: Amazon Simple Notification Service과 함께 AWS Lambda 사용

하나의 AWS 계정에서 Lambda 함수를 사용하여 별도의 AWS 계정에 Amazon SNS 주제를 구독합니다. 이 자습서에서는 AWS Command Line Interface를 사용하여 Lambda 함수 생성, Amazon SNS 주제 생성 및 이 두 리소스가 서로 액세스할 수 있는 권한 부여와 같은 AWS Lambda 작업을 수행합니다.

### 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

이 자습서에서는 계정 2개를 사용합니다. AWS CLI 명령은 각각 다른 계정에서 사용하도록 구성된 [명명된 프로필](#) 2개를 사용하여 이를 설명합니다. 이름이 다른 프로필 즉, 기본 프로필 하나 또는 명명된 프로필 하나를 사용하는 경우 필요에 따라 명령을 수정합니다.

## Amazon SNS 주제 생성

계정 A에서 Amazon SNS 주제를 생성합니다.

```
$ aws sns create-topic --name lambda-x-account --profile accountA
```

명령을 통해 반환된 주제 ARN을 메모합니다. 이는 주제를 구독하기 위해 Lambda 함수에 권한을 추가할 때 필요합니다.

## 실행 역할 만들기

계정 B에서 함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaBasicExecutionRole.
  - 역할 이름 – **lambda-sns-role**.

AWSLambdaBasicExecutionRole 정책은 함수가 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## Lambda 함수 만들기

계정 B에서 Amazon SNS의 이벤트를 처리하는 함수를 만듭니다. 다음은 Amazon SNS 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 함수 코드 \(p. 209\)](#) 단원을 참조하십시오.

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");
};
```

함수를 만들려면

1. 샘플 코드를 `index.js` 파일에 복사합니다.
2. 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

3. `create-function` 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name SNS-X-Account \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::012345678901B:role/service-role/lambda-sns-execution-role \
--timeout 60 --profile accountB
```

명령을 통해 반환된 함수 ARN에 주목합니다. 이는 Amazon SNS에 함수를 호출할 권한을 추가할 때 필요합니다.

## 교차 계정 권한 설정

계정 A에서 계정 B에 주제를 구독할 수 있는 권한을 부여합니다.

```
$ 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
```

계정 B에서 Lambda 권한을 추가하여 Amazon SNS로부터 호출을 허용합니다.

```
$ 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\"}"
}
```

정책을 추가할 때 --source-account 파라미터를 사용하여 Lambda 정책에 소스 계정을 추가하지 마십시오. 소스 계정은 이벤트 소스에 대해 지원되지 않으므로 액세스가 거부됩니다.

## 구독 생성

계정 B에서 해당 주제에 대한 Lambda 함수를 구독합니다. 메시지가 계정 A의 lambda-x-account 주제로 전송되면, Amazon SNS는 계정 B에서 SNS-X-Account 함수를 호출합니다.

```
$ 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"
}
```

출력에는 주제 구독의 ARN이 포함되어 있습니다.

## 구독 테스트

계정 A에서 구독을 테스트합니다. 텍스트 파일에 Hello World를 입력하고 message.txt로 저장합니다. 그런 다음, 다음 명령을 실행합니다.

```
$ aws sns publish --message file://message.txt --subject Test \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
```

```
--profile accountA
```

이는 Amazon SNS 서비스가 메시지를 승인했다는 것을 나타내는 고유한 식별자가 있는 메시지 ID를 반환합니다. 그러면 Amazon SNS는 주제의 구독자에게 이를 전달하려고 시도합니다. 또는 JSON 문자열을 message 파라미터에 직접 제공할 수 있지만, 텍스트 파일을 사용하면 메시지에서 줄 바꿈을 사용할 수 있습니다.

Amazon SNS에 대한 자세한 내용은 [Amazon Simple Notification Service](#)란을 참조하십시오.

## 샘플 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

주제

- [Node.js 8 \(p. 209\)](#)
- [Java 8 \(p. 209\)](#)
- [Go \(p. 210\)](#)
- [Python 3 \(p. 210\)](#)

## Node.js 8

다음 예에서는 Amazon SNS의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

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");
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Java 8

다음 예에서는 Amazon SNS의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

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;
    }
}
```

## 종속성

- aws-lambda-java-core
- aws-lambda-java-events

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## Go

다음 예에서는 Amazon SNS의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

### 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)
}
```

go build를 사용하여 실행 파일을 빌드하고 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지 \(Go\) \(p. 283\)](#) 단원을 참조하십시오.

## Python 3

다음 예에서는 Amazon SNS의 메시지를 처리하고 해당 메시지의 내용을 로깅합니다.

### 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
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Using AWS Lambda with Amazon SQS

AWS Lambda 함수를 사용하여 [표준 Amazon Simple Queue Service\(Amazon SQS\) 대기열](#)에서 메시지를 처리할 수 있습니다. Amazon SQS를 사용하면 작업을 대기열로 전송하고 비동기식으로 처리하여 애플리케이션의 구성 요소 중 하나에서 작업을 오프로드할 수 있습니다.

Lambda에서는 대기열을 폴링하고 대기열 메시지가 포함된 이벤트를 사용해 함수를 [동기식으로 \(p. 80\)](#) 호출합니다. Lambda는 메시지를 배치로 읽고 각 배치에 대해 한 번씩 함수를 호출합니다. 함수가 배치를 성공적으로 처리하면 Lambda는 대기열에서 메시지를 삭제합니다.

Example Amazon SQS메시지 이벤트

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
      "body": "test",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIEQZJOL023YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
      "messageAttributes": {},
      "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
      "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",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082650636",
        "SenderId": "AIDAIEQZJOL023YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082650649"
      },
      "messageAttributes": {},
      "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
      "awsRegion": "us-east-2"
    }
  ]
}
```

Lambda에서는 [긴 폴링](#)을 사용하여 활성 상태가 될 때까지 대기열을 폴링합니다. 메시지를 사용할 수 있는 경우 Lambda는 배치를 읽는 속도를 높여 동시성 한도에 도달할 때까지 함수를 호출합니다. Lambda가 규모

조정을 통해 Amazon SQS 대기열에서 메시지를 처리하는 자세한 내용은 [조정 동작 이해 \(p. 84\)](#) 단원을 참조 하십시오.

Lambda가 대기열에서 메시지를 읽을 때 해당 메시지는 대기열에 그대로 남아 있지만 Lambda가 삭제할 때 까지 습김 상태가 됩니다. 함수가 오류를 반환하거나 대기열의 [제한 시간 초과](#) 전에 처리를 마치지 못하면 다시 표시됩니다. 그런 다음 Lambda는 Lambda 함수에 다시 메시지를 전송합니다. 실패한 배치의 모든 메시지는 대기열로 반환되므로 함수 코드가 부작용 없이 동일한 메시지를 여러 번 처리할 수 있어야 합니다.

#### 섹션

- [Lambda에서 사용할 수 있도록 대기열 구성 \(p. 212\)](#)
- [대기열을 이벤트 소스로 구성 \(p. 212\)](#)
- [실행 역할 권한 \(p. 213\)](#)
- [자습서: Amazon Simple Queue Service과 함께 AWS Lambda 사용 \(p. 213\)](#)
- [샘플 Amazon SQS 함수 코드 \(p. 216\)](#)
- [Amazon SQS 애플리케이션을 위한 AWS SAM 템플릿 \(p. 219\)](#)

## Lambda에서 사용할 수 있도록 대기열 구성

Lambda 함수의 이벤트 소스로 제공할 [표준 Amazon SQS 대기열을 만듭니다](#). 그런 다음 Lambda 함수가 각 이벤트 배치를 처리하고 Lambda가 확장 시 병목 현상 오류에 응답해 다시 시도할 수 있는 시간을 허용하도록 대기열을 구성합니다.

각 레코드 배치를 처리할 함수 시간을 허용하려면 소스 대기열의 제한 시간 초과를 함수에 대해 구성한 [제한 시간 \(p. 35\)](#)의 6배 이상으로 설정합니다. 추가 시간은 함수가 이전 배치를 처리하는 중 함수 실행에서 병목 현상이 발생한 경우 Lambda가 재시도하도록 허용합니다.

메시지 처리에 여러 번 실패하면 Amazon SQS에서는 해당 메시지를 [배달 못한 편지 대기열](#)로 보냅니다. 문제 해결을 위해 처리하지 못한 메시지를 보관하도록 소스 대기열에 대해 배달 못한 편지 대기열을 구성합니다. 제한(throttling)으로 인해 메시지를 배달 못한 편지 대기열로 보내지 않으려면 대기열의 리드라이브 정책의 `maxReceiveCount`를 5 이상으로 설정합니다.

## 대기열을 이벤트 소스로 구성

이벤트 소스 매핑을 생성하여 Lambda가 대기열의 항목을 Lambda 함수로 전송하게 할 수 있습니다. 여러 이벤트 소스 매핑을 생성하여 하나의 함수로 여러 대기열의 항목을 처리할 수 있습니다. Lambda가 대상 함수를 호출하면 이벤트에는 구성 가능한 최대 배치 크기까지 항목이 여러 개 포함될 수 있습니다.

Amazon SQS 대기열에 대한 이벤트 소스 매핑을 추가하려면

1. Lambda 콘솔 [함수 페이지](#)를 엽니다.
2. 함수를 선택합니다.
3. 트리거 추가에서 SQS를 선택합니다.
4. 트리거 구성에서 이벤트 소스를 구성합니다.
  - SQS 대기열 – 소스 대기열을 지정합니다.
  - 배치 크기 – 한 번 호출 시 대기열에서 읽고 함수로 보내는 최대 항목 수를 지정합니다.
  - 활성화 – 이벤트 소스를 비활성화하려면 확인란 선택을 취소합니다.
5. [추가]를 선택합니다.
6. Save를 선택합니다.

함수 제한 시간은 항목의 전체 배치를 처리할 시간이 충분하도록 구성합니다. 항목을 처리하는 데 걸리는 시간이 길면 더 작은 배치 크기를 선택합니다. 배치 크기가 크면 매우 빠르거나 오버헤드가 큰 워크로드에 대한

효율성에 영향을 미칠 수 있습니다. 하지만 함수에서 오류를 반환하면 배치 내 모든 항목이 대기열로 반환됩니다. 함수에 [예약된 동시성 \(p. 37\)](#)을 구성할 경우, 최소 동시성 실행 수를 5로 설정하여 Lambda가 함수를 호출할 때 발생할 수 있는 조절 오류를 최소화하십시오.

Lambda API 또는 AWS SDK를 사용하여 이벤트 소스를 구성하려면 [CreateEventSourceMapping \(p. 343\)](#) 및 [UpdateEventSourceMapping \(p. 444\)](#) 작업을 사용하십시오.

## 실행 역할 권한

Amazon SQS 대기열의 메시지를 관리하려면 Lambda에 다음 권한이 필요합니다. 이러한 권한을 함수의 [실행 역할 \(p. 9\)](#)에 추가합니다.

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

자세한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.

## 자습서: Amazon Simple Queue Service과 함께 AWS Lambda 사용

이 자습서에서는 [Amazon SQS](#) 대기열에서 메시지를 사용하기 위해 Lambda 함수를 생성합니다.

### 사전 조건

이 자습서는 사용자가 Lambda 작업과 Lambda 콘솔에 대한 기본 지식을 알고 있다고 가정합니다. 그렇지 않은 경우 [AWS Lambda 시작하기 \(p. 3\)](#)의 지침에 따라 첫 Lambda 함수를 생성합니다.

이 설명서의 절차에 따르려면 명령을 실행할 셀 또는 명령줄 터미널이 필요합니다. 명령은 프롬프트 기호(\$) 와 해당하는 경우 현재 디렉터리의 이름이 앞에 붙은 상태로 목록에 표시됩니다.

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

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

Linux 및 macOS는 선호 셸과 패키지 관리자를 사용합니다. Windows 10에서 [Linux용 Windows Subsystem을 설치](#)하여 Ubuntu와 Bash의 Windows 통합 버전을 가져옵니다.

## 실행 역할 만들기

함수에 AWS 리소스에 액세스할 수 있는 권한을 제공하는 [실행 역할 \(p. 9\)](#)을 만듭니다.

실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – AWS Lambda.
  - 권한 – AWSLambdaSQSQueueExecutionRole.
  - 역할 이름 – `lambda-sqs-role`.

AWSLambdaSQSQueueExecutionRole 정책은 함수가 Amazon SQS에서 항목을 읽고 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

## 함수 만들기

다음은 Amazon SQS 이벤트 입력을 수신하고 이벤트에 포함된 메시지를 처리하는 예제 코드입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

### Note

다른 언어로 작성된 샘플 코드는 [샘플 Amazon SQS 함수 코드 \(p. 216\)](#) 단원을 참조하십시오.

#### Example index.js

```
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

#### 함수를 만들려면

- 샘플 코드를 index.js 파일에 복사합니다.
- 배포 패키지를 만듭니다.

```
$ zip function.zip index.js
```

- create-function 명령을 사용해 Lambda 함수를 만듭니다.

```
$ aws lambda create-function --function-name ProcessSQSRecord \
--zip-file file://function.zip --handler index.handler --runtime nodejs8.10 \
--role arn:aws:iam::123456789012:role/lambda-sqs-role
```

## 함수 테스트

invoke AWS Lambda CLI 명령 및 샘플 Amazon Simple Queue Service 이벤트를 사용하여 Lambda 함수를 수동으로 호출합니다.

핸들러가 예외 없이 정상적으로 반환되면 Lambda는 메시지가 성공적으로 처리된 것으로 간주하고 대기열의 새 메시지 읽기를 시작합니다. 메시지가 성공적으로 처리되면 대기열에서 자동으로 삭제됩니다. 핸들러에 예외가 발생하면 Lambda는 메시지의 입력을 처리하지 않은 것으로 간주하고 동일한 메시지 배치로 함수를 호출합니다.

- 다음 JSON을 파일에 복사하고 input.txt로 저장합니다.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
      "body": "test",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIEQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649183"
      }
    }
  ]
}
```

```
        "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
}
]
```

2. 다음 invoke 명령을 실행합니다.

```
$ aws lambda invoke --invocation-type RequestResponse --function-name ProcessSQSRecord \
--payload file://input.txt outputfile.txt
```

invoke 명령은 동기식 실행을 요청하는 RequestResponse를 호출 유형으로 지정합니다. 자세한 내용은 [호출 유형 \(p. 80\)](#) 단원을 참조하십시오.

3. outputfile.txt 파일의 출력을 확인합니다.

## Amazon SQS 대기열을 생성합니다.

Lambda 함수가 이벤트 소스로 사용할 수 있는 Amazon SQS 대기열을 생성합니다.

대기열을 생성하려면

1. AWS Management 콘솔에 로그인한 후 다음에서 Amazon SQS 콘솔을 엽니다. <https://console.aws.amazon.com/sqs/>
2. Amazon SQS 콘솔에서 대기열을 만듭니다.
3. 식별한 대기열 ARN(Amazon 리소스 이름)을 적어 두거나 기록합니다. 다음 단계에서 대기열을 Lambda 함수와 연결할 때 필요합니다.

AWS Lambda에서 이벤트 소스 매핑을 생성합니다. 이 이벤트 소스 매핑은 Amazon SQS 대기열을 Lambda 함수와 연결합니다. 이 이벤트 소스 매핑을 생성하면 AWS Lambda가 대기열 폴링을 시작합니다.

종합적 경험을 테스트합니다. 대기열 업데이트를 수행할 때 Amazon Simple Queue Service은 메시지를 대기열에 기록합니다. AWS Lambda가 대기열을 폴링하면 새 레코드를 감지하고 이벤트(이 경우 Amazon SQS 메시지)를 함수에 전달하여 사용자의 Lambda 함수를 실행합니다.

## 이벤트 소스 구성

지정된 Amazon SQS 대기열과 Lambda 함수 간 매핑을 생성하려면 다음 AWS CLI `create-event-source-mapping` 명령을 실행합니다. 명령이 실행된 후 UUID를 적어 두거나 기록합니다. 예를 들어, 이벤트 소스 매핑을 삭제하기로 선택할 때처럼 다른 모든 명령에서 이벤트 소스 매핑을 참조하려면 이 UUID가 필요합니다.

```
$ aws lambda create-event-source-mapping --function-name ProcessSQSRecord --batch-size 10 \
--event-source SQS-queue-arn
```

다음 명령을 실행하여 이벤트 소스 매핑 목록을 가져올 수 있습니다.

```
$ aws lambda list-event-source-mappings --function-name ProcessSQSRecord \
--event-source SQS-queue-arn
```

이 목록은 사용자가 생성한 모든 이벤트 소스 매팅을 반환하며 각 매팅에 대해 `LastProcessingResult`를 표시합니다. 이 필드는 문제가 있을 경우 유용한 메시지를 제공하는데 사용됩니다. `No records processed`(AWS Lambda가 폴링을 시작하지 않았거나 대기열에 레코드가 없음을 나타냄) 및 `OK`(AWS Lambda가 대기열에서 레코드를 성공적으로 읽고 Lambda 함수를 호출함을 나타냄) 같은 값은 아무 문제가 없다는 점을 나타냅니다. 문제가 있는 경우 오류 메시지를 수신합니다.

## 설정 테스트

아래 다음과 같이 설정을 테스트할 수 있습니다.

1. Amazon SQS 콘솔에서 대기열로 메시지를 전송합니다. Amazon SQS는 이러한 작업에 대한 레코드를 대기열에 기록합니다.
2. AWS Lambda가 대기열을 폴링하고 대기열에 대한 업데이트를 감지하면 대기열에서 찾은 이벤트 데이터를 전달하여 Lambda 함수를 호출합니다.
3. 함수는 Amazon CloudWatch에서 로그를 실행하고 생성합니다. Amazon CloudWatch 콘솔에서 보고된 로그를 확인할 수 있습니다.

## 샘플 Amazon SQS 함수 코드

샘플 코드는 다음 언어로 제공됩니다.

### 주제

- [Node.js \(p. 216\)](#)
- [Java \(p. 217\)](#)
- [C# \(p. 217\)](#)
- [Go \(p. 218\)](#)
- [Python \(p. 218\)](#)

## Node.js

다음은 입력으로 Amazon SQS 이벤트 메시지를 수신하고 처리하는 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

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");
};
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 자침은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

## Java

다음은 입력으로 Amazon SQS 이벤트 메시지를 수신하고 처리하는 Java 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

코드에서 `handleRequest`는 핸들러입니다. 해당 핸들러는 `aws-lambda-java-events` 라이브러리에서 사전 정의된 `SQSEvent` 클래스를 사용합니다.

### Example ProcessSQSRecord.java

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class ProcessSQSEvents implements RequestHandler<SQSEvent, Void>{
    @Override
    public Void handleRequest(SQSEvent event, Context context)
    {
        for(SQSMessage msg : event.getRecords()){
            System.out.println(new String(msg.getSQS().getBody()));
        }
        return null;
    }
}
```

### 종속성

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## C#

다음은 입력으로 Amazon SQS 이벤트 메시지를 수신하고 처리하는 C# 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 콘솔에 기록합니다.

코드에서 `handleRequest`는 핸들러입니다. 해당 핸들러는 `AWS.Lambda.SQSEvents` 라이브러리에서 사전 정의된 `SQSEvent` 클래스를 사용합니다.

### 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.";
}
}
```

.NET Core 프로젝트의 `Program.cs`를 위의 샘플로 바꿉니다. 지침은 [.NET Core CLI \(p. 296\)](#) 단원을 참조하십시오.

## Go

다음은 입력으로 Amazon SQS 이벤트 메시지를 수신하고 처리하는 Go 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 CloudWatch Logs에 기록합니다.

코드에서 `handler`는 핸들러입니다. 해당 핸들러는 `aws-lambda-go-events` 라이브러리에서 사전 정의된 `SQSEvent` 클래스를 사용합니다.

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)
}
```

`go build`를 사용하여 실행 파일을 빌드하고 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지 \(Go\) \(p. 283\)](#) 단원을 참조하십시오.

## Python

다음은 입력으로 Amazon SQS 레코드를 수락하고 처리하는 Python 코드의 예입니다. 이해를 돋기 위해, 코드는 수신 이벤트 데이터의 일부를 에 기록합니다.

지침에 따라 AWS Lambda 함수 배포 패키지를 생성합니다.

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))
```

샘플 코드를 압축하여 배포 패키지를 생성합니다. 지침은 [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#) 단원을 참조하십시오.

## Amazon SQS 애플리케이션을 위한 AWS SAM 템플릿

AWS SAM을 사용하여 이 애플리케이션을 빌드할 수 있습니다. AWS SAM 템플릿을 만드는 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [AWS SAM 템플릿 기본 사항](#)을 참조하십시오.

다음은 [자습서 \(p. 213\)](#)의 Lambda 애플리케이션을 위한 샘플 AWS SAM 템플릿입니다. 아래 텍스트를 .yaml 파일로 복사하고 이전에 만든 ZIP 패키지 옆에 저장합니다. Handler 및 Runtime 파라미터 값은 이전 단원에서 함수를 생성할 때 사용한 것과 일치해야 합니다.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Example of processing messages on an SQS queue with Lambda
Resources:
  MySQSQueueFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Events:
        MySQSEvent:
          Type: SQS
          Properties:
            Queue: !GetAtt MySqsQueue.Arn
            BatchSize: 10
  MySqsQueue:
    Type: AWS::SQS::Queue
```

패키지 및 배포 명령을 사용하여 서비스 애플리케이션을 패키징하고 배포하는 방법은 AWS Serverless Application Model 개발자 안내서의 [서비스 애플리케이션 배포](#) 단원을 참조하십시오.

# Lambda 애플리케이션 모니터링 및 문제 해결

AWS Lambda는 Lambda 함수 호출의 동작을 자동으로 추적하여 사용자가 모니터링할 수 있는 피드백을 제공합니다. 또한 이벤트 소스 통합 및 다운스트림 리소스가 예상대로 작동하는지 여부를 비롯하여 전반적인 함수 호출을 분석하는데 사용할 수 있는 측정치를 제공합니다. 다음 단원에서는 Lambda 함수 호출 동작을 분석하는데 사용할 수 있는 도구에 대한 지침을 제공합니다.

## 주제

- [사용 Amazon CloudWatch \(p. 220\)](#)
- [AWS X-Ray 사용 \(p. 226\)](#)
- [AWS CloudTrail를 사용하여 AWS Lambda API 호출 로깅 \(p. 231\)](#)

## 사용 Amazon CloudWatch

AWS Lambda에서는 사용자 대신 Lambda 함수를 자동으로 모니터링하여 Amazon CloudWatch를 통해 지표를 보고합니다. 실행되는 동안 코드를 모니터링할 수 있도록, Lambda는 요청 수, 요청당 실행 시간 및 오류를 유발하는 요청 수를 자동으로 추적하고 연결된 CloudWatch 지표를 게시합니다. 이러한 지표를 활용하여 사용자 지정 경보를 설정할 수 있습니다. CloudWatch에 대한 자세한 내용은 [Amazon CloudWatch 사용 설명서](#)를 참조하십시오.

AWS Lambda 콘솔, CloudWatch 콘솔 및 기타 Amazon Web Services(AWS) 리소스를 사용하여 각 Lambda 함수에 대한 요청 빈도 및 오류 비율을 확인할 수 있습니다. 다음 주제에서는 Lambda CloudWatch 지표 및 액세스 방법에 대해 설명합니다.

- [AWS Lambda에 대한 Amazon CloudWatch 지표 액세스 \(p. 222\)](#)
- [AWS Lambda 지표 \(p. 223\)](#)

코드에 로깅 문을 삽입하여 코드가 예상대로 작동하는지 확인할 수 있습니다. Lambda는 Amazon CloudWatch Logs와 자동으로 통합되며 코드의 모든 로그를 Lambda 함수(/aws/lambda/<function name>)와 연결된 CloudWatch Logs 그룹으로 푸시합니다. 로그 그룹 및 CloudWatch 콘솔을 통해 해당 그룹에 액세스하는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [시스템, 애플리케이션 및 사용자 지정 로그 파일 모니터링](#)을 참조하십시오. CloudWatch 로그 항목 액세스 방법에 대한 자세한 내용은 AWS Lambda에 대한 [Amazon CloudWatch 로그 액세스 \(p. 222\)](#) 단원을 참조하십시오.

### Note

Lambda 함수 코드가 실행 중이지만 몇 분 후에 로그 데이터가 생성되지 않는 경우, Lambda 함수의 실행 역할이 로그 데이터를 CloudWatch Logs에 쓸 수 있는 권한을 부여하지 않았다는 의미일 수 있습니다. 이러한 권한을 부여하기 위해 실행 역할을 올바르게 설정했는지 확인하는 방법에 대한 내용은 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.

## AWS Lambda 문제 해결 시나리오

이 단원에서는 CloudWatch의 로깅 및 모니터링 기능을 사용하여 Lambda 함수를 모니터링하고 문제를 해결하는 방법에 대한 예를 설명합니다.

### 문제 해결 시나리오 1: Lambda 함수가 정상적으로 작동하지 않음

이 시나리오에서 사용자는 [자습서: Amazon S3과 함께 AWS Lambda 사용 \(p. 190\)](#) 단원을 방금 완료했습니다. 하지만 썸네일 이미지를 Amazon S3에 업로드하기 위해 만든 Lambda 함수가 S3 객체를 생성할 때 정상

적으로 작동하지 않습니다. 예 객체를 업로드하면 썸네일 이미지가 업로드되지 않는 것을 알 수 있습니다. 다음 방법을 통해 이 문제를 해결할 수 있습니다.

Lambda 함수가 정상적으로 작동하지 않는 이유를 확인하려면

1. 코드를 확인하고 제대로 작동하는지 확인합니다. 오류 비율의 증가는 코드가 제대로 작동하지 않는다는 점을 나타냅니다.

다른 Node.js 함수처럼 코드를 로컬에서 테스트하거나, 콘솔의 테스트 호출 기능을 사용하여 Lambda 콘솔에서 코드를 테스트하거나, AWS CLI `Invoke` 명령을 사용할 수 있습니다. 코드는 이벤트에 응답해서 실행될 때마다 `/aws/lambda/<function name>`인 Lambda 함수와 연결된 로그 그룹에 로그 항목을 쓩니다.

다음은 로그에 나타날 수 있는 몇 가지 오류의 예입니다.

- 로그에 스택 추적이 표시되면 코드에 오류가 있는 것일 수 있습니다. 코드를 검토하고 스택 추적에서 참조하는 오류를 디버깅합니다.
- 로그에 `permissions denied` 오류가 표시되는 경우 실행 역할로 제공한 IAM 역할에 필요한 권한이 없을 수 있습니다. 역할을 확인하고 코드에서 참조하는 AWS 리소스에 액세스하는 데 필요한 모든 권한이 있는지 확인합니다. 실행 역할을 올바르게 설정했는지 확인하려면 [AWS Lambda 실행 역할 \(p. 9\)](#) 단원을 참조하십시오.
- 로그에 `timeout exceeded` 오류가 있으면 구성된 제한 시간 내에 반환하지 못해서 함수가 종료된 것입니다. 제한 시간이 너무 짧거나 코드를 실행하는 데 시간이 너무 오래 걸리기 때문일 수 있습니다.
- 로그에 `memory exceeded` 오류가 표시되는 경우 메모리 설정이 너무 낮은 것입니다. 이를 더 높은 값으로 설정하십시오. 메모리 크기 한도에 대한 정보는 [CreateFunction \(p. 347\)](#) 단원을 참조하십시오.

2. Lambda 함수를 확인하고 요청을 수신하는지 확인합니다.

함수 코드가 제대로 작동하고 테스트 호출에 올바르게 응답하는 경우에도 이 함수는 Amazon S3에서 요청을 수신하지 못할 수 있습니다. Amazon S3가 함수를 호출할 수 있는 경우 CloudWatch 요청 지표가 증가해야 합니다. 요청이 증가하지 않는 경우 함수와 연결된 액세스 권한 정책을 확인하십시오.

## 문제 해결 시나리오 2: Lambda 함수를 실행 시 소요 시간이 증가 함

이 시나리오에서 사용자는 [자습서: Amazon S3과 함께 AWS Lambda 사용 \(p. 190\)](#) 단원을 방금 완료했습니다. 하지만 썸네일 이미지를 Amazon S3에 업로드하기 위해 만든 Lambda 함수가 S3 객체를 생성할 때 정상적으로 작동하지 않습니다. 예 객체를 업로드하면 썸네일 이미지가 업로드되지 않지만 코드가 예상보다 오래 실행되는 것을 알 수 있습니다. 이 문제는 여러 방법으로 해결할 수 있습니다. 예를 들어, Lambda 함수에 대한 소요 시간 지표를 모니터링하여 실행 시간이 증가하는지 확인할 수 있습니다. 또는 Lambda 함수에 대한 CloudWatch 오류 지표가 증가한다면 이는 제한 시간 오류 때문일 수 있습니다.

Lambda 함수를 실행할 때 소요 시간이 증가하는 이유를 확인하려면

1. 다른 메모리 설정으로 코드를 테스트합니다.

코드 실행에 너무 오래 걸리는 경우 로직을 실행하는 데 필요한 컴퓨팅 리소스가 충분하지 않은 것일 수 있습니다. Lambda 콘솔의 테스트 호출 기능을 사용하여 함수에 할당된 메모리를 늘리고 코드를 다시 테스트합니다. 사용된 메모리, 코드 실행 시간 및 함수 로그 항목에 할당된 메모리를 확인할 수 있습니다. 메모리 설정을 변경하면 실행 시간에 대한 청구 방법도 변경될 수 있습니다. 요금에 대한 자세한 내용은 [AWS Lambda](#) 단원을 참조하십시오.

2. 로그를 사용하여 실행 병목 현상의 원인을 조사합니다.

다른 Node.js 함수와 마찬가지로 코드를 로컬에서 테스트하거나, Lambda 콘솔의 테스트 호출 기능을 사용하여 Lambda 내에서 테스트하거나, AWS CLI를 사용하여 `asyncInvoke` 명령을 사용할 수 있습니다. 코드는 이벤트에 응답해서 실행될 때마다 `/aws/lambda/<function name>`라는 이름의 Lambda 함

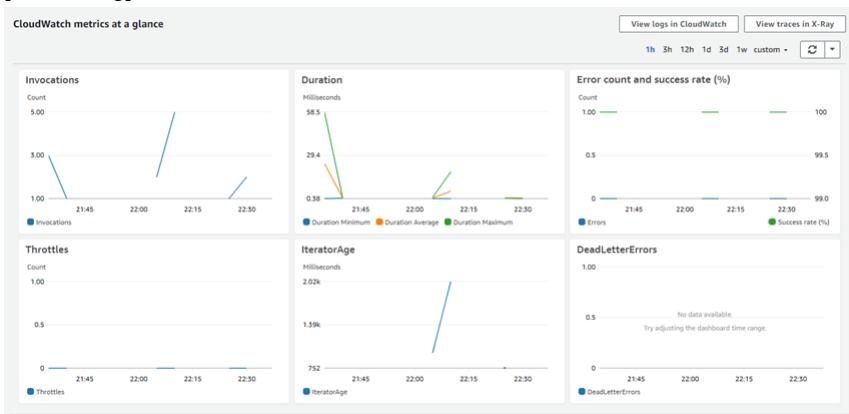
수와 연결된 로그 그룹에 로그 항목을 씁니다. 코드의 다른 부분을 실행하는 데 걸리는 시간을 확인하려면 다른 서비스에 대한 요청 같은 코드의 여러 부분에 로깅 문을 추가합니다.

## AWS Lambda에 대한 Amazon CloudWatch 지표 액세스

AWS Lambda에서는 사용자 대신 함수를 자동으로 모니터링하여 Amazon CloudWatch를 통해 지표를 보고합니다. 이 지표에는 총 요청 수, 소요 시간, 오류 발생률이 포함됩니다.

Lambda 콘솔을 사용하여 지표에 액세스하려면

1. [Lambda 콘솔](#)을 엽니다.
2. [Lambda 콘솔 함수 페이지](#)를 엽니다.
3. [Monitoring]을 선택합니다.



콘솔에서 다음 그래프를 볼 수 있습니다.

### Lambda 모니터링 그래프

- Invocations - 5분 기간 동안 함수가 호출된 횟수입니다.
- Duration - 평균, 최소, 최대 실행 시간입니다.
- 오류 수 및 성공률(%) - 오류 수 및 오류 없이 완료된 실행의 비율입니다.
- Throttles - 동시 사용자 한도로 인해 실행에 실패한 횟수입니다.
- IteratorAge - 스트림 이벤트 소스에서 Lambda가 배치의 마지막 항목을 받아 함수를 호출했을 때 해당 항목의 수명입니다.
- DeadLetterErrors - Lambda가 배달 못한 편지 대기열에 쓰려고 시도했으나 실패한 이벤트 수입니다.

CloudWatch의 그래프 정의를 보려면 그래프 오른쪽 상단의 메뉴에서 View in metrics(지표에서 보기)를 선택하십시오. Lambda에서 기록하는 지표에 대한 자세한 내용은 [AWS Lambda 지표 \(p. 223\)](#)을 참조하십시오.

## AWS Lambda에 대한 Amazon CloudWatch 로그 액세스

AWS Lambda에서는 사용자 대신 Lambda 함수를 자동으로 모니터링하여 Amazon CloudWatch를 통해 지표를 보고합니다. 함수의 실패 문제를 해결하는 데 도움을 제공하기 위해, Lambda는 함수에 의해 처리되는 모든 요청을 기록하고 코드로 생성된 로그를 Amazon CloudWatch Logs를 통해 자동으로 저장합니다.

코드에 로깅 문을 삽입하여 코드가 예상대로 작동하는지 확인할 수 있습니다. Lambda는 CloudWatch Logs와 자동으로 통합되며 코드의 모든 로그를 Lambda 함수(/aws/lambda/<function name>)와 연결된 CloudWatch Logs 그룹으로 푸시합니다. 로그 그룹 및 CloudWatch 콘솔을 통해 해당 그룹에 액세스하는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [시스템, 애플리케이션 및 사용자 지정 로그 파일 모니터링](#)을 참조하십시오.

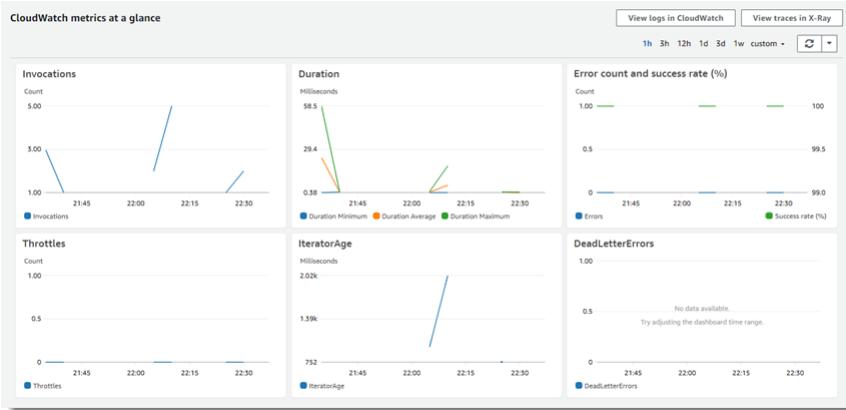
Lambda 콘솔, CloudWatch 콘솔, AWS CLI 또는 CloudWatch API를 사용하여 Lambda에 대한 로그를 확인할 수 있습니다. 다음 절차에서는 콘솔을 사용하여 로그를 확인하는 방법을 소개합니다.

#### Note

Lambda 로그 사용에 대한 추가 비용은 없지만 표준 CloudWatch Logs 비용이 적용됩니다. 자세한 내용은 [CloudWatch 요금](#)을 참조하십시오.

Lambda 콘솔을 사용하여 로그를 보려면

1. [Lambda 콘솔](#)을 엽니다.
2. Lambda 콘솔 [함수 페이지](#)를 엽니다.
3. [Monitoring]을 선택합니다.



Lambda 함수에 대한 지표를 그래픽으로 보여 줍니다.

4. CloudWatch에서 로그 보기 선택합니다.

## AWS Lambda 지표

이 주제에서는 AWS Lambda 네임스페이스, 지표 및 차원에 대해 설명합니다. AWS Lambda에서는 사용자 대신 함수를 자동으로 모니터링하여 Amazon CloudWatch를 통해 지표를 보고합니다. 이러한 지표에는 전체 호출, 오류, 기간, 제한, DLQ 오류 및 스트림 기반 호출의 반복기 수명이 포함됩니다.

CloudWatch는 기본적으로 지표 리포지토리입니다. 지표는 기본 개념으로 시간별로 정렬된 데이터 요소 세트를 나타냅니다. 사용자(또는 AWS 서비스)는 지표 데이터 포인트를 예제시하고, 사용자는 나중에 해당 데이터 포인트에 대한 통계를 시계열 데이터 세트로 순서대로 가져옵니다.

지표는 이름, 네임스페이스 및 하나 이상의 차원으로 고유하게 정의됩니다. 각 데이터 요소에는 타임스탬프와 측정 단위(선택 사항)가 있습니다. 통계를 요청하면 네임스페이스, 지표 이름 및 차원으로 반환된 데이터 스트림이 식별됩니다. CloudWatch에 대한 자세한 내용은 [Amazon CloudWatch 사용 설명서](#) 단원을 참조하십시오.

## AWS Lambda CloudWatch 지표

AWS/Lambda 네임스페이스에는 다음 지표가 포함되어 있습니다.

지표	설명
<b>Invocations</b>	<p>이벤트 또는 호출 API 호출에 대한 응답으로 함수가 호출된 횟수를 측정합니다. 이 지표가 지금은 사용이 중단된 RequestCount 지표를 대체합니다. 여기에는 성공한 호출과 실패한 호출이 모두 포함되지만, 병목 현상이 발생한 시도 횟수는 포함되지 않습니다. 이는 요금이 청구된 함수 요청과 동일합니다. AWS Lambda는 값이 0이 아닌 경우에만 이러한 지표를 CloudWatch에 전송합니다.</p> <p>단위: 수</p>
<b>Errors</b>	<p>함수 오류(응답 코드 4XX)로 인해 실패한 호출 수를 측정합니다. 이 지표가 지금은 사용이 중단된 ErrorCount 지표를 대체합니다. 호출이 실패하면 재시도가 트리거되어 성공에 이를 수도 있습니다. 여기에는 다음이 포함됩니다.</p> <ul style="list-style-type: none"> <li>처리된 예외(예: context.fail(오류))</li> <li>처리되지 않은 예외로 인한 코드 종료</li> <li>메모리 부족 예외</li> <li>시간 초과</li> <li>권한 오류</li> </ul> <p>여기에서 기본 동시 제한을 초과하는 호출 비율(오류 코드 429)로 인해 실패하거나, 혹은 내부 서비스 오류(오류 코드 500)로 인해 실패한 호출은 포함되지 않습니다.</p> <p>단위: 수</p>
<b>DeadLetterErrors</b>	<p>Lambda가 실패한 이벤트 페이로드를 구성된 배달 못한 편지 대기열에 작성할 수 없을 때 증가합니다. 이러한 원인은 다음과 같을 수 있습니다.</p> <ul style="list-style-type: none"> <li>권한 오류</li> <li>다운스트림 서비스의 병목 현상</li> <li>잘못 구성된 리소스</li> <li>시간 초과</li> </ul> <p>단위: 수</p>
<b>Duration</b>	<p>함수 코드가 호출에 따라 실행을 시작할 때부터 실행을 중단할 때까지 걸린 실제 실행 시간(wall clock time)을 측정합니다. 가능한 최대 데이터 포인트 값은 함수 제한 시간 구성과 동일합니다. 요금이 청구되는 지속 시간은 가장 가까운 100밀리초까지 반올림됩니다. AWS Lambda는 값이 0이 아닌 경우에만 이러한 지표를 CloudWatch에 전송합니다.</p> <p>단위: 밀리초</p>
<b>Throttles</b>	<p>Lambda 함수를 호출할 때 고객의 동시 제한을 초과하는 호출 비율(오류 코드 429)로 인해 병목 현상이 발생한 시도 횟수를 측정합니다. 호출이 실패하면 재시도가 트리거되어 성공에 이를 수도 있습니다.</p> <p>단위: 수</p>
<b>IteratorAge</b>	<p>스트림 기반 호출인 경우에만 내보냅니다(Lambda 스트림 또는 Kinesis 스트림에서 트리거된 함수). 레코드 배치(batch)를 처리할 때마다 마지막 레코드의 경과 시간을 측정합니다. 여기에서 경과 시간이란 가 배치를 수신한 시간과 배치에서 마지막 레코드가 스트림에 작성된 시간의 차이를 말합니다.</p>

지표	설명
	단위: 밀리초
ConcurrentExecutions	계정의 모든 함수와 사용자 지정 동시성 제한이 지정된 함수에 대한 집계 지표로 내보냅니다. 버전이나 별칭에는 해당되지 않습니다. 해당 시점에 지정된 함수에 대해 동시 실행 수 합계를 측정합니다. 일정 기간 동안 집계된 경우 평균 지표로 표시되어야 합니다.
	단위: 수
UnreservedConcurrentExecutions	계정의 모든 함수의 집계 지표로만 내보냅니다. 함수, 버전 또는 별칭에는 해당되지 않습니다. 사용자 지정 동시성 제한이 지정되지 않은 함수의 동시성 합계를 나타냅니다. 일정 기간 동안 집계된 경우 평균 지표로 표시되어야 합니다.
	단위: 개수

CloudWatch 콘솔을 사용하여 지표에 액세스하려면

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 [Metrics]를 선택합니다.
3. CloudWatch Metrics by Category(범주별 CloudWatch 지표) 창에서 Lambda 지표를 선택합니다.

#### 오류/호출 비율

Lambda 함수 호출에 대한 오류 비율을 계산할 때는 호출 요청과 실제 호출을 구분하는 것이 중요합니다. 오류 비율이 요금이 청구되는 Lambda 함수 호출 수를 초과할 수 있습니다. Lambda는 Lambda 함수 코드가 실행된 경우에만 호출 지표를 보고합니다. 호출 요청으로 인해 Lambda 함수 코드의 호출을 방해하는 조절 또는 기타 초기화 오류가 발생하는 경우 Lambda는 오류를 보고하지만 호출 지표를 로그에 기록하지 않습니다.

- 함수가 실행되면 Lambda가 Invocations=1을 내보냅니다. 하지만 함수가 실행되지 않으면 아무것도 내보내지 않습니다.
- Lambda는 각 호출 요청에 대해 Errors 데이터 포인트를 내보냅니다. Errors=0은 함수 실행 오류가 없음을 의미합니다. Errors=1은 함수 실행 오류가 있음을 의미합니다.
- Lambda는 각 호출 요청에 대해 Throttles 데이터 포인트를 내보냅니다. Throttles=0은 호출 조절이 없음을 의미합니다. Throttles=1은 호출 조절이 있음을 의미합니다.

## AWS Lambda CloudWatch 차원

다음 표의 차원을 사용하여 Lambda 함수에 대해 반환되는 지표를 구체화할 수 있습니다.

차원	설명
FunctionName	Lambda 함수를 기준으로 지표 데이터를 필터링합니다.
Resource	함수 버전 또는 별칭과 같은 Lambda 함수 리소스를 기준으로 지표 데이터를 필터링합니다.
ExecutedVersion	Lambda 함수 버전을 기준으로 지표 데이터를 필터링합니다. 별칭 호출에만 적용됩니다.

## AWS X-Ray 사용

일반적으로 Lambda 기반 애플리케이션은 Amazon S3, Amazon SNS 알림, API 작업으로의 객체 업로드 같은 이벤트에 의해 트리거되는 하나 이상의 함수로 이루어져 있습니다. 트리거되면 이러한 함수는 보통 DynamoDB 테이블이나 Amazon S3 버킷 같은 다운스트림 리소스를 호출하거나 다른 API를 호출합니다. AWS Lambda는 Amazon CloudWatch를 활용하여 모든 함수 호출에 대한 측정치와 로그를 자동으로 방출합니다. 그러나 이러한 메커니즘은 함수를 호출한 이벤트 리소스를 트레이스하거나 함수를 구성하고 있는 다른 스트림 호출을 트레이스하기에 편리하지 않을 수 있습니다. 트레이스가 어떻게 이루어지는가에 대한 전체 개요는 [AWS X-Ray](#)를 참조하십시오.

## AWS X-Ray를 통한 Lambda 기반 애플리케이션 추적

AWS X-Ray는 AWS Lambda 애플리케이션에서 성능 문제를 감지, 분석 및 최적화할 수 있도록 해주는 AWS 서비스입니다. X-Ray는 Lambda 서비스와 애플리케이션을 구성하는 업스트림 또는 다운스트림 서비스로부터 메타데이터를 수집합니다. X-Ray는 이러한 메타데이터를 이용하여 성능 병목 현상, 지연 시간 스파이크 및 Lambda 애플리케이션의 성능에 영향을 미치는 기타 문제들을 보여주는 상세한 서비스 그래프를 생성합니다.

[AWS X-Ray 서비스 맵의 Lambda \(p. 226\)](#)를 사용하여 문제가 있는 리소스나 구성 요소를 식별한 후에는 이를 확대해서 보거나 요청의 시각적 표현을 볼 수 있습니다. 이러한 시각적 표현은 이벤트 소스가 Lambda 함수를 트리거하는 시점부터 함수 실행이 완료되는 시점까지 지원됩니다. X-Ray는 Lambda 함수가 다른 서비스에 대해 수행한 다운스트림 호출과 관련된 정보를 포함하여 함수의 작업을 상세히 보여줍니다. 뿐만 아니라, X-Ray는 Lambda에 통합되어 AWS Lambda 서비스 오버헤드에 대한 가시성을 제공합니다. 이를 위해 요청의 유지 시간, 호출 수 같은 세부 사항을 표시합니다.

### Note

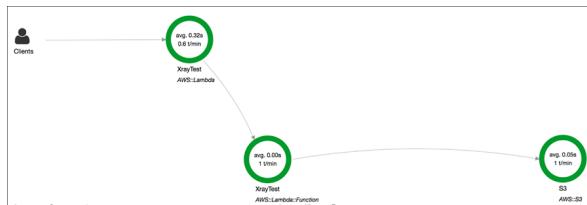
현재 X-Ray에 통합된 서비스만이 Lambda 트레이스에 포함되지 않은 독립형 트레이스로서 표시됩니다. 현재 X-Ray를 지원하는 서비스 목록은 [다른 AWS 서비스에 AWS X-Ray 통합](#)을 참조하십시오.

## AWS X-Ray 서비스 맵의 Lambda

X-Ray는 Lambda가 처리한 요청에 대한 서비스 맵에 세 가지 유형의 노드를 표시합니다.

- Lambda 서비스(AWS::Lambda) – 이 유형의 노드는 요청이 Lambda 서비스에서 소요한 시간을 나타냅니다. Lambda가 처음 요청을 수신할 때부터 요청이 Lambda 서비스를 떠날 때까지의 시간입니다.
- Lambda 함수(AWS::Lambda::Function) – 이 유형의 노드는 Lambda 함수의 실행 시간을 나타냅니다.
- 다운스트림 서비스 호출 – 이 유형에서는 Lambda 함수 내의 각 다운스트림 서비스 호출이 별도의 노드로 표현됩니다.

다음 그림에 나와 있는 노드는 Lambda 서비스, 사용자 함수 및 Amazon S3에 대한 다운스트림 호출입니다 (왼쪽에서 오른쪽으로).



자세한 정보는 [서비스 맵 보기](#)를 참조하십시오.

## AWS X-Ray 트레이스로서 Lambda

서비스 맵에서 Lambda 함수의 트레이스 보기 를 확대하여 볼 수 있습니다. 트레이스는 세그먼트 및 하위 세그먼트로 표현된 함수 호출과 관련된 세부 정보를 표시합니다.

- Lambda 서비스 세그먼트 – 이 세그먼트는 함수 호출에 사용된 이벤트 소스에 따라 서로 다른 정보를 표시합니다.
  - 동기식 및 스트림 이벤트 소스 – 서비스 세그먼트는 Lambda 서비스가 요청/이벤트를 수신할 때 시간을 측정하고 Lambda 서비스를 떠날 때(요청에 대한 최종 호출이 완료된 이후) 종료됩니다.
  - 비동기식 – 서비스 세그먼트가 응답 시간, 즉 Lambda 서비스가 클라이언트에 202 응답을 반환할 때까지 소요된 시간을 표시합니다.

Lambda 서비스 세그먼트에는 두 가지 유형의 하위 세그먼트가 포함될 수 있습니다.

- 유지 시간(비동기식 호출에만 해당) – 호출에 앞서 함수가 Lambda 서비스에서 소요한 시간을 나타냅니다. 이 하위 세그먼트는 Lambda 서비스가 요청/이벤트를 수신할 때 시작했다가 Lambda 함수가 처음 호출될 때 종료됩니다.
- 시도 – Lambda 서비스에서 발생한 오버헤드를 포함하여 단일 호출 시도를 나타냅니다. 오버헤드로는 함수 코드를 초기화하는 데 소요되는 시간과 함수 실행 시간이 있습니다.
- Lambda 함수 세그먼트 – 특정 호출 시도에서 해당 함수의 실행 시간을 나타냅니다. 함수 핸들러가 실행을 시작할 때부터 함수가 종료될 때까지의 시간입니다. 이 세그먼트에는 세 가지 유형의 하위 세그먼트가 포함될 수 있습니다.
  - 초기화 – Lambda 함수 핸들러나 정적 이니셜라이저 밖에서 코드로 정의된 함수의 `initialization` 코드를 실행하는 데 소요된 시간입니다.
  - 다운스트림 호출 – Lambda 함수의 코드에서 다른 AWS 서비스에 대해 이루어진 호출입니다.
  - 사용자 지정 하위 세그먼트 – X-Ray SDK를 사용하여 Lambda 함수 세그먼트에 추가할 수 있는 사용자 지정 하위 세그먼트 또는 사용자 주석입니다.

### Note

트레이스된 각 호출에 대해 Lambda는 Lambda 서비스 세그먼트와 모든 하위 세그먼트를 방출합니다. 이 세그먼트는 런타임과 관계없이 방출되며 이 세그먼트에서는 X-Ray SDK를 AWS API 호출에 사용해야 합니다.

## Lambda를 통한 AWS X-Ray 설정

아래에는 Lambda로 X-Ray를 설정하는 방법에 대한 자세한 정보가 나와 있습니다.

### 시작하기 전

Lambda CLI를 사용하여 Lambda 함수에 대한 트레이스를 활성화하려면 먼저 함수의 실행 역할에 트레이스 권한을 추가해야 합니다. 이 작업을 수행하려면 다음 단계를 수행하십시오.

- AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- Lambda 함수의 실행 역할을 찾습니다.
- `AWSXrayWriteOnlyAccess`라는 관리형 정책을 연결합니다.

이러한 정책에 대한 자세한 정보는 [AWS X-Ray](#)를 참조하십시오.

Lambda 콘솔을 사용하여 트레이스 모드를 활성으로 변경하는 경우에는, 다음 단원에 설명되어 있듯이 트레이스 권한이 자동으로 추가됩니다.

## 추적

애플리케이션을 경유하는 요청 경로는 트레이스 ID로 트레이스됩니다. trace는 단일 요청(보통 HTTP GET 또는 POST 요청)에 의해 생성된 모든 세그먼트를 수집합니다.

Lambda 함수의 트레이스 모드는 두 가지가 있습니다.

- Pass Through: 함수의 실행 역할에 트레이스 권한을 추가한 경우, 모든 Lambda 함수에 적용되는 기본 설정입니다. 이 모드에서는 X-Ray가 AWS Elastic Beanstalk 같은 업스트림 서비스에 대해 활성화한 경우에만 Lambda 함수가 트레이스됩니다.
- Active: Lambda 함수에 이 설정이 지정되어 있으면 Lambda는 X-Ray에 의해 지정된 샘플링 알고리즘에 따라 호출 요청을 자동으로 샘플링합니다.

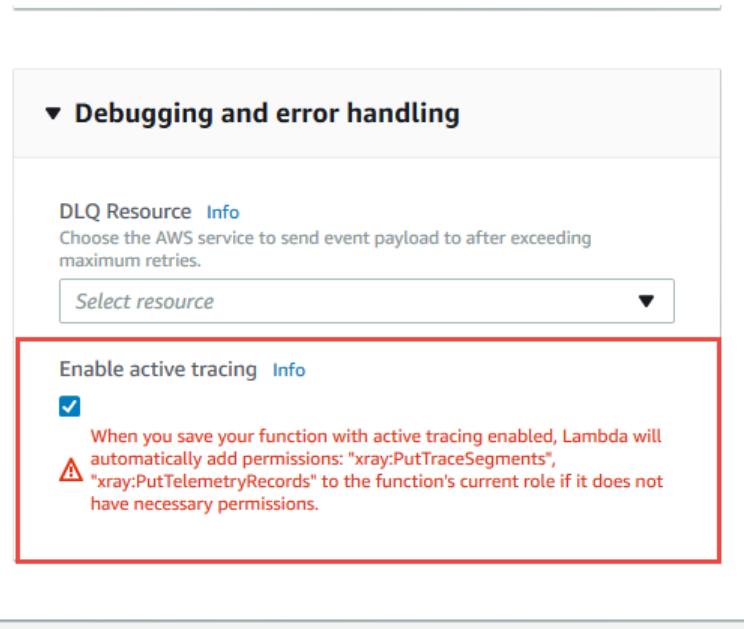
### Note

X-Ray는 여전히 애플리케이션이 처리하는 요청에 대한 대표 샘플을 제공하면서 트레이스가 효율적일 수 있도록 샘플링 알고리즘을 적용합니다. 기본 샘플링 알고리즘은 분당 1회 요청인데, 요청의 5%가 이 제한을 넘어서 샘플링됩니다. 한편, 함수에 대한 트래픽 볼륨이 적으면 샘플링 속도가 증가할 수 있습니다.

Lambda Management Console, Lambda [CreateFunction \(p. 347\)](#) 또는 [UpdateFunctionConfiguration \(p. 455\)](#) API 작업을 사용하여 Lambda 함수의 트레이스 모드를 변경할 수 있습니다.

Lambda 콘솔을 사용하는 경우에는 다음이 적용됩니다.

- 함수의 트레이스 모드를 활성으로 변경하면 트레이스 권한이 함수의 실행 역할에 자동으로 연결됩니다. Lambda가 함수의 실행 역할에 AWSXrayWriteOnlyAccess 정책을 추가할 수 없었다는 오류가 표시되면 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔에 로그인하여 정책을 수동으로 추가합니다.
- 트레이스 기능을 활성화하려면 함수의 [Configuration] 탭으로 이동해서 [Enable active tracing] 상자를 선택합니다.



Lambda [CreateFunction \(p. 347\)](#) 또는 [UpdateFunctionConfiguration \(p. 455\)](#) API 작업을 사용하고 있는 경우에는 다음과 같이 하십시오.

- 트레이스 모드를 활성으로 변경하고 싶으면 `TracingConfig` 파라미터의 `Mode` 속성을 `Active`.로 설정합니다. 다시 말하지만, 새 함수의 트레이스 모드는 기본적으로 `PassThrough`로 설정되어 있습니다.
- 신규 또는 업데이트된 Lambda 함수는 `$LATEST` 버전이 지정한 값으로 설정되어 있습니다.

Note

함수의 실행 역할에 트레이스 권한을 추가하지 않은 경우에는 오류가 표시됩니다. 자세한 내용은 [시작하기 전 \(p. 227\)](#) 단원을 참조하십시오.

## Lambda 함수에서 트레이스 세그먼트 방출

트레이스된 각 호출에 대해 Lambda는 Lambda 서비스 세그먼트와 모든 하위 세그먼트를 방출합니다. 뿐만 아니라,는 Lambda 함수 세그먼트와 init 하위 세그먼트를 방출합니다. 이들 세그먼트는 함수의 런타임에 관계 없이 방출되며, 코드 변경이나 추가적인 라이브러리가 필요하지 않습니다. Lambda 함수의 X-Ray 트레이스에 다운스트림 호출에 대한 사용자 지정 세그먼트, 주석 또는 하위 세그먼트를 포함시키고 싶으면 추가 라이브러리를 포함시키고 코드에 주석을 추가해야 합니다.

모든 계측 코드는 초기화 코드의 일부가 아니라 Lambda 함수 핸들러 내에서 실행되어야 합니다.

다음 예제들은 지원되는 런타임 내에서 이를 수행하는 방법을 설명합니다.

- [AWS Lambda에서 Python 코드 계측 \(p. 254\)](#)
- [AWS Lambda에서 Node.js 코드 계측 \(p. 243\)](#)
- [AWS Lambda에서 Java 코드 계측 \(p. 280\)](#)
- [??? \(p. 293\)](#)

## Lambda 환경의 AWS X-Ray 데몬

[AWS X-Ray 데몬](#)은 원시 세그먼트 데이터를 수집하여 AWS X-Ray 서비스에 중계하는 소프트웨어 애플리케이션입니다. 데몬은 AWS X-Ray SDK와 함께 작동하기 때문에 SDK가 전송하는 데이터가 X-Ray 서비스에 도달할 수 있습니다.

Lambda 함수를 트레이스하면 X-Ray 데몬이 Lambda 환경에서 자동으로 실행되어 트레이스 데이터를 수집하여 X-Ray로 전송합니다. 트레이스 시 데몬은 최대 16MB(함수에 할당된 메모리의 3%)를 사용합니다. 예를 들어 메모리의 128MB를 Lambda 함수에 할당한 경우에는 X-Ray 데몬은 이 중 16MB를 트레이스에 사용하게 됩니다. Lambda 함수에 1024MB를 할당하면 데몬은 31MB(3%)를 트레이스에 사용합니다. 자세한 내용은 [AWS X-Ray 데몬](#)을 참조하십시오.

Note

Lambda는 함수의 메모리 한도를 초과하지 않도록 X-Ray 데몬을 종료하려고 할 것입니다. 예를 들어 Lambda 함수에 128MB가 할당되어 X-Ray 데몬이 16MB를 트레이스에 사용할 수 있다고 가정해 보겠습니다. 함수에는 112MB의 메모리가 남게 됩니다. 그러나 함수가 112MB를 초과하면 메모리 부족 오류가 발생하지 않도록 데몬이 종료됩니다.

## 환경 변수를 사용하여 AWS X-Ray와 통신

AWS Lambda는 환경 변수를 사용하여 X-Ray 데몬과의 통신을 촉진하고 X-Ray SDK를 구성합니다.

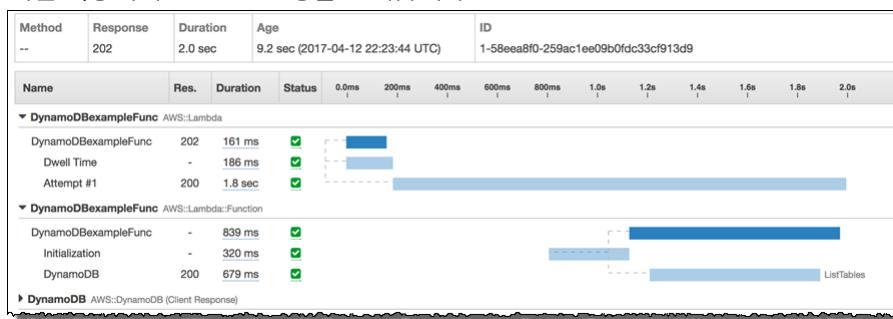
- `_X_AMZN_TRACE_ID`: 샘플링 결정, 트레이스 ID 및 상위 세그먼트 ID가 포함된 트레이스 헤더를 포함합니다 (이들 속성에 대한 자세한 정보는 [트레이스 헤더](#)를 참조). Lambda가 함수 호출 시 트레이스 헤더를 수신할 경우, 이 헤더는 `_X_AMZN_TRACE_ID` 환경 변수를 채우는데 사용됩니다. 트레이스 헤더가 수신되지 않은 경우에는 Lambda가 사용자를 위해 하나 생성하게 됩니다.

- AWS\_XRAY\_CONTEXT\_MISSING: X-Ray SDK는 이 변수를 사용하여 함수가 X-Ray 데이터를 기록하려고 하는 경우에 수행할 동작을 결정하지만, 트레이스 헤더를 사용할 수는 없습니다. Lambda는 기본적으로 이 값을 LOG\_ERROR로 설정합니다.
- AWS\_XRAY\_DAEMON\_ADDRESS: 이 환경 변수는 X-Ray 데몬의 주소를 [IP\\_ADDRESS:PORT](#)라는 형식으로 노출합니다. X-Ray 데몬의 주소를 사용하면 X-Ray SDK를 사용하지 않고도 X-Ray 데몬에 트레이스 데이터를 직접 전송할 수 있습니다.

## AWS X-Ray 콘솔의 Lambda 트레이스: 예제

다음 단원에서는 두 가지 다른 Lambda 함수에 대한 Lambda 트레이스를 보여줍니다. 각 트레이스는 호출 유형(비동기식 및 동기식)에 따른 트레이스 구조를 보여줍니다.

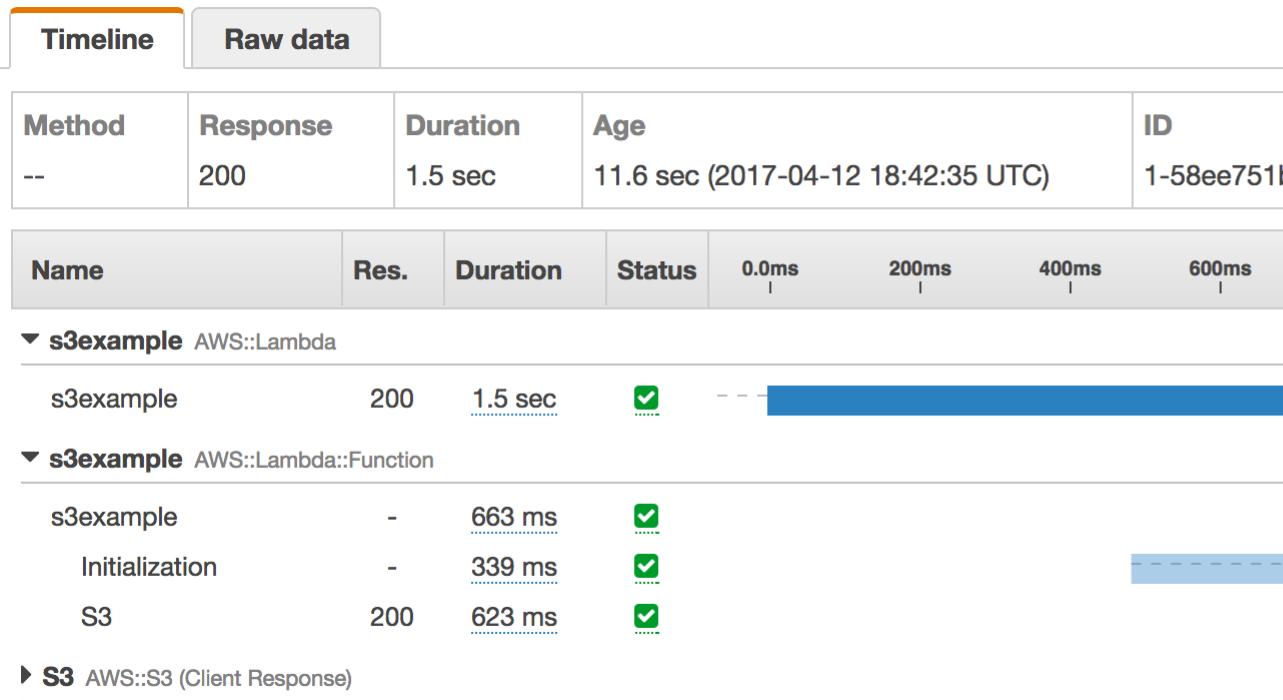
- Async – 다음 예제는 호출이 1회 성공적으로 이루어지고 DynamoDB에 대한 다운스트림 호출이 1회 이루어진 비동기식 Lambda 요청을 보여줍니다.



Lambda 서비스 세그먼트는 응답 시간, 즉 클라이언트에 응답(예를 들어 202)을 반환하는 데 걸리는 시간을 캡슐화합니다. 여기에는 Lambda 서비스 대기열(유지 시간)과 각 호출 시도에서 소요된 시간에 대한 하위 세그먼트가 포함되어 있습니다(위의 예제에서는 오직 하나의 호출 시도만 표시). 서비스 세그먼트의 각 연결 하위 세그먼트는 해당되는 사용자 함수 세그먼트를 갖게 됩니다. 이 예제에서는 사용자 함수 세그먼트에 두 개의 세그먼트가 포함되어 있습니다. 하나는 핸들러 이전에 실행된 함수의 초기화 코드를 표시하는 초기화 하위 세그먼트이고, 다른 하나는 DynamoDB에 대한 ListTables 호출을 표시하는 다운스트림 호출 하위 세그먼트입니다.

각 호출 하위 세그먼트와 각 다운스트림 호출에 대해 상태 코드와 오류 메시지가 표시됩니다.

- 동기식 – 다음 예제는 Amazon S3에 대해 다운스트림 호출이 1회 이루어진 동기식 호출을 보여줍니다.



Lambda 서비스 세그먼트는 Lambda 서비스에서 요청 수행에 소요된 전체 시간을 캡처합니다. 서비스 세그먼트는 해당되는 사용자 함수 세그먼트를 갖게 됩니다. 이 예제에서는 사용자 함수 세그먼트에는 함수의 초기화 코드(핸들러 이전에 실행된 코드)를 표시하는 하위 세그먼트와 Amazon S3에 대한 `PutObject` 호출을 표시하는 하위 세그먼트가 포함되어 있습니다.

#### Note

HTTP 호출을 트레이스하고 싶으면 HTTP 클라이언트를 사용해야 합니다. 자세한 내용은 [Java용 X-Ray SDK](#)를 사용하여 다운스트림 HTTP 웹 서비스에 대한 [호출 트레이스](#) 또는 [Node.js용 X-Ray SDK](#)를 사용하여 다운스트림 HTTP 웹 서비스에 대한 [호출 트레이스](#)를 참조하십시오.

## AWS CloudTrail를 사용하여 AWS Lambda API 호출 로깅

AWS Lambda는 AWS Lambda에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 AWS Lambda에 대한 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 AWS Lambda 콘솔로부터의 호출과 AWS Lambda API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 AWS Lambda 이벤트를 비롯하여 CloudTrail 이벤트를 Amazon S3 버킷으로 지속적으로 배포할 수 있습니다. 추적을 구성하지 않은 경우 Event history(이벤트 기록)에서 CloudTrail 콘솔의 최신 이벤트를 볼 수도 있습니다. CloudTrail에서 수집하는 정보를 사용하여 AWS Lambda에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

그 구성 및 활성화 방법을 포함하여 CloudTrail에 대한 자세한 내용은 [AWS CloudTrail User Guide](#)를 참조하십시오.

## CloudTrail의 AWS Lambda 정보

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 지원되는 이벤트 활동이 AWS Lambda에서 이루어지면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록에서 이벤트 보기](#)를 참조하십시오.

AWS Lambda 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려는 경우 추적을 생성합니다. 추적은 CloudTrail이 Amazon S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 정보는 다음을 참조하십시오.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기](#)

AWS Lambda는 CloudTrail 로그 파일의 이벤트로 다음 작업의 로깅을 지원합니다.

- [AddPermission \(p. 335\)](#)
- [CreateEventSourceMapping \(p. 343\)](#)
- [CreateFunction \(p. 347\)](#)  
  
(`ZipFile` 파라미터가 `CreateFunction`의 CloudTrail 로그에서 생략)
- [DeleteEventSourceMapping \(p. 357\)](#)
- [DeleteFunction \(p. 360\)](#)
- [GetEventSourceMapping \(p. 371\)](#)
- [GetFunction \(p. 374\)](#)
- [GetFunctionConfiguration \(p. 377\)](#)
- [GetPolicy \(p. 390\)](#)
- [ListEventSourceMappings \(p. 402\)](#)
- [ListFunctions \(p. 405\)](#)
- [RemovePermission \(p. 433\)](#)
- [UpdateEventSourceMapping \(p. 444\)](#)
- [UpdateFunctionCode \(p. 448\)](#)  
  
(`ZipFile` 파라미터가 `UpdateFunctionCode`의 CloudTrail 로그에서 생략)
- [UpdateFunctionConfiguration \(p. 455\)](#)

모든 로그 항목은 누가 요청을 생성했는가에 대한 정보가 들어 있습니다. 로그의 사용자 신원 정보를 참조하여 루트 또는 IAM 사용자 자격 증명 혹은 어떤 역할이나 폐더레이션 사용자를 위한 임시 보안 자격 증명을 사용하여 또는 다른 AWS 서비스에 의해 요청이 이루어졌는지 확인할 수 있습니다. 자세한 내용은 [CloudTrail Event Reference](#)의 `userIdentity` 필드를 참조하십시오.

원하는 기간만큼 버킷에 로그 파일을 저장할 수 있습니다. 그러나 Amazon S3 수명 주기 규칙을 정의하여 자동으로 로그 파일을 보관하거나 삭제할 수도 있습니다. 기본적으로 로그 파일은 Amazon S3 서버 쪽 암호화 (SSE)를 사용하여 암호화합니다.

로그 파일 전송 시 신속한 조치를 취해야 하는 경우 새 로그 파일이 전송될 때 CloudTrail에서 Amazon SNS 알림을 게시하게 할 수 있습니다. 자세한 정보는 [CloudTrail용 Amazon SNS 알림 구성](#)을 참조하십시오.

또한 여러 AWS 리전 및 여러 AWS 계정의 AWS Lambda 로그 파일을 하나의 S3 버킷으로 집계할 수도 있습니다. 자세한 정보는 [CloudTrail 로그 파일 작업](#)을 참조하십시오.

## AWS Lambda 로그 파일 항목 이해

CloudTrail 로그 파일은 하나 이상의 로그 항목이 있으며, 각 항목은 여러 개의 JSON 형식 이벤트로 구성됩니다. 로그 항목은 어떤 소스로부터의 요청 하나를 나타내며 요청된 작업, 모든 파라미터, 작업 날짜와 시간 등에 대한 정보가 들어 있습니다. 로그 항목의 순서가 정해져 있는 것은 아닙니다. 즉 순서가 지정된 퍼블릭 API 호출의 스택 추적이 아닙니다.

다음은 `GetFunction` 및 `DeleteFunction` 작업의 CloudTrail 로그 항목을 보여주는 예제입니다.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::999999999999:user/myUserName",
        "accountId": "999999999999",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:03:36Z",
      "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",
      "recipientAccountId": "999999999999"
    }
  ]
}
```

```
        "eventType": "AwsApiCall",
        "recipientAccountId": "999999999999"
    }
}
```

#### Note

`eventName`은 "GetFunction20150331"과 같은 날짜 및 버전 정보를 포함할 수 있지만, 여전히 동일한 퍼블릭 API를 참조합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 [CloudTrail 이벤트 기록이 지원하는 서비스](#)를 참조하십시오.

## CloudTrail을 사용하여 함수 호출 추적

CloudTrail은 데이터 이벤트도 기록합니다. 데이터 이벤트 로깅 기능을 켜서 Lambda 함수가 호출될 때마다 이벤트를 기록할 수 있습니다. 이렇게 하면 함수를 호출 중인 ID가 무엇인지 및 호출 빈도를 파악할 수 있습니다. 이 작업은 AWS CloudTrail 콘솔 또는 [Invoke \(p. 392\)](#) CLI 작업을 사용하여 수행할 수 있습니다. 이 옵션에 대한 자세한 내용은 [추적을 위해 데이터 및 관리 이벤트 기록](#)을 참조하십시오.

# Node.js를 사용하여 Lambda 함수 빌드

AWS Lambda는 다음과 같은 Node.js 런타임을 지원합니다.

## Node.js 런타임

이름	식별자	Node.js 버전	JavaScript용 AWS SDK	운영 체제
Node.js 10	nodejs10.x	10.15	2.437.0	Amazon Linux 2
Node.js 8.10	nodejs8.10	8.10	2.290.0	Amazon Linux

Lambda 함수를 생성할 때 사용하고 싶은 런타임을 지정합니다. 자세한 내용은 [CreateFunction \(p. 347\)](#)의 `runtime` 파라미터를 참조하십시오.

다음 단원에서는 [일반적인 프로그래밍 패턴 및 핵심 개념](#)이 Node.js에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다. 별도로 명시된 경우를 제외하고, 다음 단원에서 설명하는 프로그래밍 모델은 지원되는 모든 런타임 버전에 적용됩니다.

## 주제

- [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#)
- [AWS Lambda 함수 핸들러\(Node.js\) \(p. 236\)](#)
- [AWS Lambda 콘텍스트 객체\(Node.js\) \(p. 238\)](#)
- [AWS Lambda 함수 로깅\(Node.js\) \(p. 239\)](#)
- [AWS Lambda 함수 오류\(Node.js\) \(p. 240\)](#)
- [AWS Lambda에서 Node.js 코드 계측 \(p. 243\)](#)

## AWS Lambda 배포 패키지(Node.js)

Lambda 함수를 생성하려면 먼저, 코드와 종속 프로그램으로 구성된 zip 파일인 Lambda 함수 배포 패키지를 만듭니다.

배포 패키지를 자체적으로 생성하거나 Lambda 콘솔에서 직접 코드를 작성할 수 있습니다. 후자의 경우, 콘솔이 사용자를 대신하여 배포 패키지를 생성하고 이를 업로드하여 Lambda 함수를 생성합니다. 함수를 생성하기 위해 콘솔을 사용할 것인지 여부를 결정하려면 다음을 참조하십시오.

- 간단한 시나리오 – 사용자 지정 코드에 AWS SDK 라이브러리만 필요할 경우에는 AWS Lambda 콘솔에서 인라인 에디터를 사용할 수 있습니다. 콘솔을 사용하여 코드를 편집하고 이를 업로드할 수 있습니다. 콘솔은 서비스가 실행할 수 있는 배포 패키지로 해당 구성 정보가 포함된 코드를 압축합니다.

또한 샘플 이벤트 데이터를 사용하여 수동으로 호출함으로써 콘솔에서 코드를 테스트할 수도 있습니다.

### Note

Lambda 서비스는 Node.js용 AWS SDK를 사전 설치했습니다.

- 고급 시나리오 – 이미지 처리를 위한 그래픽 라이브러리 같이 다른 리소스를 사용하는 코드를 작성하고 있는 경우나 콘솔 대신 AWS CLI를 사용하고 싶은 경우에는 먼저 Lambda 함수 배포 패키지를 생성한 다음, 콘솔이나 CLI를 사용하여 패키지를 업로드해야 합니다.

### Note

배포 패키지를 생성한 후에는 이를 직접 업로드하거나 Lambda 함수를 생성하고 싶은 동일한 AWS 리전의 Amazon S3 버킷에 .zip 파일을 먼저 업로드한 다음, 콘솔이나 AWS CLI를 사용하여 Lambda 함수를 생성할 때 버킷 이름과 객체 키 이름을 지정할 수 있습니다.

다음은 배포 패키지를 생성하기 위한 절차의 예제입니다(콘솔을 사용하지 않는 경우). `filename.js` 코드 파일을 포함하는 배포 패키지를 생성하기를 원하고 코드가 `async` 라이브러리를 사용한다고 가정해 보십시오.

1. 텍스트 에디터를 열고 코드를 기록합니다. 파일(예: `filename.js`)을 저장합니다.

Lambda 함수를 생성할 때 이 파일 이름을 사용하여 핸들러를 지정합니다.

2. 동일한 디렉터리에서 npm을 사용하여 코드의 기반이 되는 라이브러리를 설치합니다. 예를 들어 코드가 `async` 라이브러리를 사용하는 경우, 다음의 npm 명령을 사용합니다.

```
npm install async
```

3. 디렉터리는 다음의 구조를 갖게 됩니다.

```
filename.js
node_modules/async
node_modules/async/lib
node_modules/async/lib/async.js
node_modules/async/package.json
```

4. 폴더의 내용을 압축하여 배포 패키지를 만듭니다(예: `sample.zip`).

그런 다음, Lambda 함수를 생성할 때 배포 패키지로 .zip 파일 이름을 지정합니다.

기본 바이너리를 포함하여 자체 바이너리를 포함시키고 싶은 경우에는 업로드한 Zip 파일에 이들을 패키징한 다음, 이전에 시작한 Node.js 또는 다른 프로세스에서 호출할 때 이들을 참조하기만 하면 됩니다(생성한 Zip 파일 내의 상대 경로 포함). 함수 코드의 시작 부분에 `process.env['PATH'] = process.env['PATH'] + ':' + process.env['LAMBDA_TASK_ROOT']`을 포함시키십시오.

Lambda 함수 패키지에 네이티브 바이너리를 포함시키는 방법에 대한 자세한 내용은 [AWS Lambda에서 실행 파일 실행](#)을 참조하십시오. 이와 함께 Zip 파일의 내용에 대한 필수 권한도 제공해야 합니다. 자세한 내용은 [Lambda 배포 패키지에 대한 권한 정책 \(p. 32\)](#) 단원을 참조하십시오.

## AWS Lambda 함수 핸들러(Node.js)

AWS Lambda에서 `handler` 객체를 통해 Lambda 함수를 호출합니다. `handler`는 Lambda 함수의 이름을 나타내며, AWS Lambda가 함수 코드를 실행할 때 사용하는 진입점 역할을 합니다. 예:

```
exports.myHandler = function(event, context, callback) {
  ... function code
  callback(null, "some success message");
  // or
  // callback("some error type");
}
```

- `myHandler` – AWS Lambda가 호출하는 함수의 이름입니다. 이 코드를 `helloworld.js`로 저장한다고 생각해 보십시오. 이때 `myHandler`는 Lambda 함수 코드가 들어 있는 함수이고, `helloworld`는 배포 패키지를 대표하는 파일의 이름입니다. 자세한 내용은 [AWS Lambda 배포 패키지\(Node.js\) \(p. 235\)](#) 단원을 참조하십시오.

- context – AWS Lambda는 이 파라미터를 사용하여 Lambda 함수의 실행에 관한 세부 정보를 제공합니다. 자세한 내용은 [AWS Lambda 콘텍스트 객체\(Node.js\) \(p. 238\)](#) 단원을 참조하십시오.
- callback(선택 사항) – [콜백 파라미터 사용 \(p. 237\)](#) 단원을 참조하십시오.

## 콜백 파라미터 사용

Node.js 런타임은 callback 파라미터를 옵션으로 지원합니다. 콜백 옵션을 사용하여 호출자에게 정보를 명시적으로 반환할 수 있습니다.

```
callback(Error error, Object result);
```

두 파라미터는 모두 옵션입니다. `error`는 실패한 Lambda 함수 실행의 결과를 제공하는 데 사용할 수 있는 옵션 파라미터입니다. 함수 실행에 성공하면 첫 번째 파라미터로 `null`을 전달할 수 있습니다.

`result`는 성공적인 함수 실행의 결과를 제공하는 데 사용할 수 있는 파라미터 옵션입니다. 제공된 결과는 `JSON.stringify`와 호환이 가능해야 합니다. 오류 메시지가 나타나면 이 파라미터가 무시됩니다.

코드에서 `callback`을 사용하지 않으면 AWS Lambda가 둑시적으로 이를 호출하고 반환 값은 `null`이 됩니다. 콜백이 호출되면 AWS Lambda는 이벤트 루프가 빈 상태가 될 때까지 Lambda 함수를 계속 호출합니다.

다음은 콜백 예제입니다.

```
callback();      // Indicates success but no information returned to the caller.  
callback(null); // Indicates success but no information returned to the caller.  
callback(null, "success"); // Indicates success with information returned to the caller.  
callback(error); // Indicates error with error information returned to the caller.
```

AWS Lambda는 `error` 파라미터에 대해 `null`이 아닌 모든 값을 처리된 예외로 취급합니다.

콜백 메서드는 `null`이 아닌 `error` 값의 문자열을 Lambda 함수에 연결된 Amazon CloudWatch Logs 스트림에 자동으로 로깅합니다.

Lambda 함수가 동기식으로 호출되면 콜백은 다음과 같이 응답 본문을 반환합니다. `error`가 `null`이면 응답 본문이 `result`의 문자열 표현으로 설정됩니다. `error`가 `null`이 아니면 `error` 값이 응답 본문에 채워집니다.

`callback(error, null)`([및 `callback\(error\)`](#))이 호출되면 Lambda는 오류 객체의 첫 번째 256KB를 로깅합니다. 더 큰 오류 객체의 경우, AWS Lambda가 로그를 자르고 오류 객체 옆에 `Truncated by Lambda 텍스트`를 표시합니다.

런타임 버전 8.10을 사용하는 경우에는 다음 `async` 키워드를 넣을 수 있습니다.

```
exports.myHandler = async function(event, context) {  
    ...  
  
    // return information to the caller.  
}
```

## 예

다음 예제 코드를 검토해 보십시오.

```
exports.myHandler = function(event, context, callback) {  
    console.log("value1 = " + event.key1);  
    console.log("value2 = " + event.key2);  
    callback(null, "some success message");
```

```
// or
// callback("some error type");
}
```

이 예제에는 `myHandler`라는 함수가 한 개 있습니다.

해당 함수에서 `console.log()` 문은 수신되는 이벤트 데이터의 일부를 CloudWatch Logs에 로깅합니다. `callback` 파라미터가 호출되면 Lambda 함수는 전달된 이벤트 루프가 비워져야만 종료됩니다.

v8.10 런타임의 `async` 기능을 사용하려면 다음과 같은 코드 샘플을 고려해 보십시오.

```
exports.myHandler = async function(event, context) {
  console.log("value1 = " + event.key1);
  console.log("value2 = " + event.key2);
  return "some success message";
// or
// throw new Error("some error type");
}
```

## AWS Lambda 콘텍스트 객체(Node.js)

Lambda은 함수를 실행할 때 컨텍스트 객체를 [핸들러 \(p. 236\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 제공하는 메서드 및 속성들을 제공합니다.

### 컨텍스트 메서드

- `getRemainingTimeInMillis()` – 실행 시간이 초과되기 전에 남은 시간(밀리초)을 반환합니다.

### 컨텍스트 속성

- `functionName` – Lambda 함수의 이름
- `functionVersion` – 함수의 버전 ([p. 45](#))
- `invokedFunctionArn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `memoryLimitInMB` – 함수에 구성된 메모리 양
- `awsRequestId` – 호출 요청의 식별자
- `logGroupName` – 함수에 대한 로그 그룹
- `logStreamName` – 함수 인스턴스에 대한 로그 스트림
- `identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
  - `cognitoIdentityId` – 인증된 Amazon Cognito 자격 증명
  - `cognitoIdentityPoolId` – 호출에 대한 권한을 부여한 Amazon Cognito ID 풀
- `clientContext` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트
  - `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 – 모바일 애플리케이션에서 설정된 사용자 지정 값입니다.
- `callbackWaitsForEmptyEventLoop` – Node.js 이벤트 루프가 빌 때까지 대기하는 대신, 콜백 ([p. 237](#))이 실행될 때 즉시 응답을 보내려면 `false`로 설정합니다. `false`인 경우, 대기 중인 이벤트는 다음 번 호출 중에 계속 실행됩니다.

다음 예제는 컨텍스트 정보를 기록하는 핸들러 함수를 보여줍니다.

#### Example index.js

```
exports.handler = function(event, context, callback) {
    console.log('remaining time =', context.getRemainingTimeInMillis());
    console.log('functionName =', context.functionName);
    console.log('AWSrequestID =', context.awsRequestId);
    callback(null, context.functionName);
};
```

## AWS Lambda 함수 로깅(Node.js)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드의 로그를 출력하려면, [콘솔 객체](#)에서 메서드를 사용하거나, `stdout` 또는 `stderr`에 쓰는 로깅 라이브러리를 사용합니다. 다음 예제는 환경 변수의 값과 이벤트 객체를 로깅합니다.

#### Example index.js

```
exports.handler = async (event) => {
    console.log("## ENVIRONMENT VARIABLES");
    console.log(JSON.stringify(process.env, null, 2));
    console.log("## EVENT");
    console.log(JSON.stringify(event, null, 2));
};
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

#### Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스](#) ([p. 101](#))에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션](#) ([p. 116](#)) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 `--log-type` 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 `LogResult` 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBUlQgUmVxdWVzdElkOjA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms      Billed
Duration: 100 ms           Memory Size: 128 MB     Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(Node.js)

Lambda 함수가 AWS Lambda에 실행에 실패했음을 알리면 Lambda가 오류 객체의 문자열 변환을 시도합니다. 다음 예제를 고려하십시오.

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

exports.handler = function(event, context, callback) {
    // This example code only throws error.
    var error = new Error("something is wrong");
    callback(error);
};
```

이 Lambda 함수를 호출하면 함수 실행이 완료되었으나 오류가 발생하여 AWS Lambda에 오류 정보를 전달한다고 이 함수가 AWS Lambda에 알립니다. AWS Lambda는 클라이언트에게 다시 오류 정보를 반환합니다.

```
{
    "errorMessage": "something is wrong",
    "errorType": "Error",
    "stackTrace": [
        "exports.handler (/var/task/index.js:10:17)"
    ]
}
```

Node.js 런타임 버전 8.10의 비동기 기능으로 함수를 작성해도 같은 결과를 얻게 됩니다. 예:

```
exports.handler = async function(event, context) {
    function AccountAlreadyExistsError(message) {
        this.name = "AccountAlreadyExistsError";
        this.message = message;
    }
}
```

```

}
AccountAlreadyExistsError.prototype = new Error();

const error = new AccountAlreadyExistsError("Account is in use!");
throw error
};

```

다시 이 Lambda 함수를 호출하면 함수 실행이 완료되었으나 오류가 발생하여 AWS Lambda에 오류 정보를 전달한다고 이 함수가 AWS Lambda에 알립니다. AWS Lambda는 클라이언트에게 다시 오류 정보를 반환합니다.

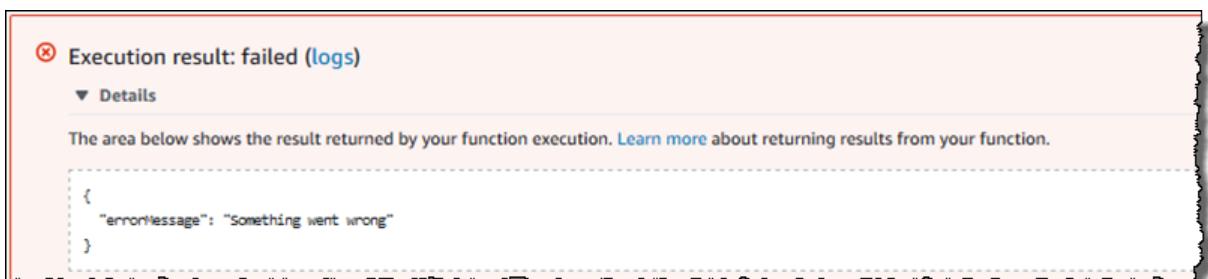
```
{
  "errorMessage": "Account is in use!",
  "errorType": "Error",
  "stackTrace": [
    "exports.handler (/var/task/index.js:10:17)"
  ]
}
```

이때 오류 정보는 스택 추적 요소의 stackTrace JSON 배열로 반환됩니다.

오류 정보를 다시 가져오는 방법은 클라이언트가 함수를 호출할 때 지정하는 호출 유형에 따라 다릅니다.

- 클라이언트가 호출 유형을 RequestResponse로 지정한 경우(동기식 실행), 호출을 한 클라이언트로 결과가 반환됩니다.

예를 들어 콘솔은 항상 RequestResponse 호출 유형을 사용하기 때문에 콘솔은 [Execution result] 섹션에 다음과 같이 오류 메시지를 표시합니다.



같은 정보가 CloudWatch에도 전송되며 [Log output] 섹션에 같은 로그가 표시됩니다.

Summary	Log output
Code SHA-256 U4b2T1lAJ6JHw7VXNt W0zs3RXCtox6Ph3Jw3 7xQfO6g=	The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. <a href="#">Click here</a> to view the CloudWatch log group.  START RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Version: \$LATEST 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value3 = undefined 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value4 = undefined 2017-07-19T19:39:53.488Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value5 = undefined 2017-07-19T19:39:53.489Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 {"errorMessage": "Something went wrong"} END RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 REPORT RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Duration: 41.24 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 17 MB
Request ID 093bf2a1-6cba-11e7-b9ad-850729ae85a8	
Duration 41.24 ms	
Billed duration 100 ms	

- 클라이언트가 호출 유형을 Event(즉, 비동기식 실행)로 지정한 경우에는 AWS Lambda가 어떤 결과도 반환하지 않습니다. 그 대신 CloudWatch Logs에 오류 정보를 기록합니다. CloudWatch 측정치에서도 오류 측정치를 볼 수 있습니다.

이벤트 소스에 따라 AWS Lambda는 실패한 Lambda 함수를 다시 시도할 수 있습니다. 예를 들어 Kinesis가 이벤트 소스인 경우, AWS Lambda는 Lambda 함수가 성공하거나 스트림의 레코드가 만료될 때까지 실패한 호출을 재시도합니다. 재시도에 대한 자세한 내용은 [AWS Lambda 재시도 동작 \(p. 83\)](#)을 참조하십시오.

콘솔에서 위의 Node.js 코드를 테스트하려면

- 콘솔에서 hello-world 블루프린트를 사용하여 Lambda 함수를 생성합니다. [runtime]에서 [Node.js]를 선택하고 [Role]에서 [Basic execution role]을 선택합니다. 작업 방법에 대한 자침은 [콘솔로 Lambda 함수 만들기 \(p. 3\)](#) 단원을 참조하십시오.
- 템플릿 코드를 이 단원에서 제공되는 코드로 대체합니다.
- Lambda 콘솔에서 제공되는 Hello World라는 샘플 이벤트 템플릿을 사용하여 Lambda 함수를 테스트합니다.

## 함수 오류 처리

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 AWS Step Functions 상태 시스템 내에서 직접 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

CreateAccount 상태가 Lambda 함수를 사용하여 고객의 세부 정보를 데이터베이스에 기록하는 작업인 경우를 생각해 보십시오.

- 작업이 성공하면 계정이 만들어지고 환영 이메일이 전송됩니다.
- 사용자가 이미 존재하는 사용자 이름에 대해 계정을 만들려고 시도하면 Lambda 함수는 오류를 발생시켜 상태 시스템이 다른 사용자 이름을 제안하고 계정 생성 프로세스를 재시도하도록 만듭니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다. Node.js의 사용자 지정 오류는 오류 프로토 타입을 확장해야 합니다.

```
exports.handler = function(event, context, callback) {
    function AccountAlreadyExistsError(message) {
        this.name = "AccountAlreadyExistsError";
        this.message = message;
    }
    AccountAlreadyExistsError.prototype = new Error();

    const error = new AccountAlreadyExistsError("Account is in use!");
    callback(error);
};
```

Catch 규칙을 사용하여 오류를 파악하도록 Step Functions를 구성할 수 있습니다.

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": ["AccountAlreadyExistsError"],
```

```
        "Next": "SuggestAccountName"
    }
],
},
...
}
```

런타임에 AWS Step Functions는 오류를 파악하여 Next 전환에 지정된 대로 SuggestAccountName 상태로 전환을 합니다.

#### Note

Error 객체의 이름 속성은 ErrorEquals 값과 일치해야 합니다.

사용자 지정 오류 처리는 [서버리스 애플리케이션](#)을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda [프로그래밍 모델](#) (p. 31)에서 지원되는 모든 언어와 통합이 되기 때문에 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서버리스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

## AWS Lambda에서 Node.js 코드 계측

Node.js의 경우, Lambda가 X-Ray에 하위 세그먼트를 방출하여 함수에서 이루어진 다른 AWS 서비스에 대한 다운스트림 호출 관련 정보를 표시합니다. 이를 위해서는 먼저 배포 패키지에 [Node.js용 AWS X-Ray SDK](#)를 포함시켜야 합니다. 뿐만 아니라 다음과 같이 AWS SDK require 문을 래핑합니다.

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

그런 다음, 이전 예제에서 정의한 AWS 변수를 사용하여 예를 들면 X-Ray에서 트레이스를 원하는 서비스 클라이언트를 초기화합니다.

```
s3Client = AWS.S3();
```

이러한 단계들을 거친 후에는 s3Client를 사용하여 함수에서 이루어진 모든 호출에서 X-Ray 하위 세그먼트가 해당 호출을 표시합니다. 예를 들면 Node.js 함수를 다음과 같이 실행하여에서 트레이스가 어떻게 보이는지 확인할 수 있습니다.

#### Example index.js

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));

var s3 = new AWS.S3();

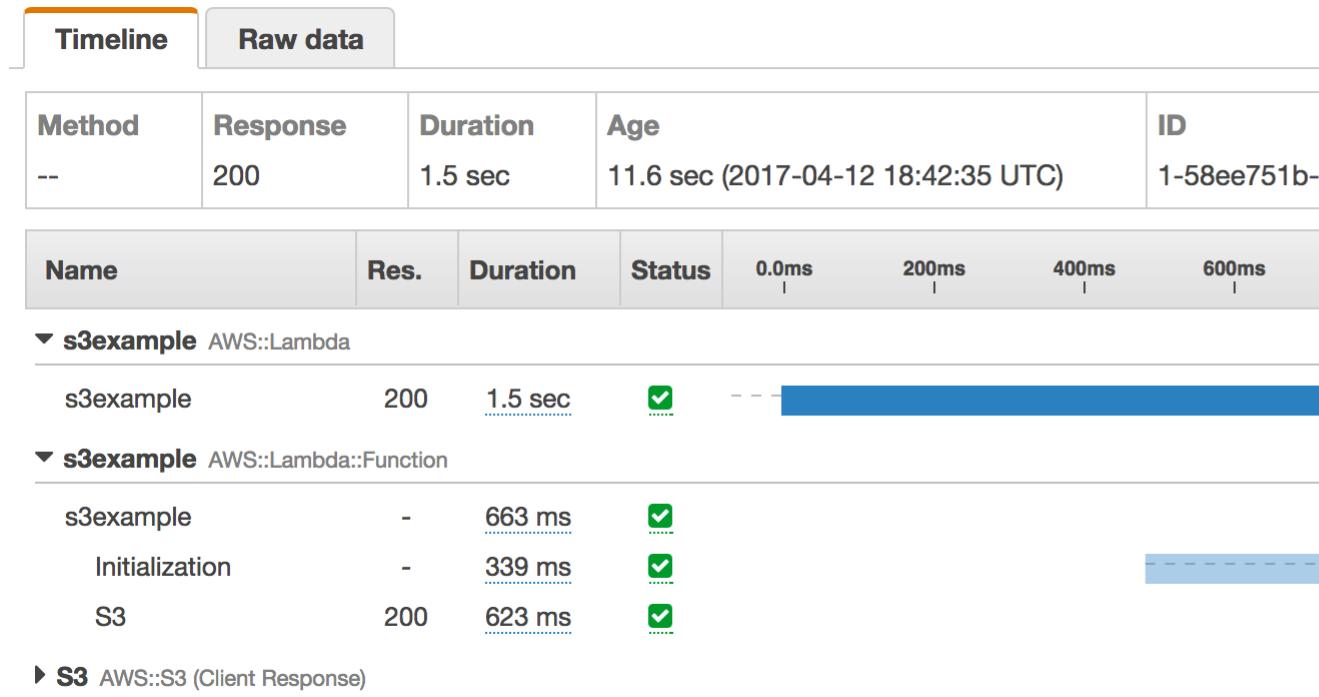
exports.handler = (event, context, callback) => {

    var params = {Bucket: process.env.BUCKET_NAME, Key: process.env.BUCKET_KEY, Body: process.env.BODY};

    s3.putObject(params, function(err, data) {
        if (err)
            { console.log(err) }
        else {
            console.log('success!')
    })
}
```

```
    });
};
```

위의 코드에서 내보낸 추적은 다음과 같습니다(비동기식 호출).



# Python을 사용하여 Lambda 함수 빌드

다음 단원에서는 [일반적인 프로그래밍 패턴 및 핵심 개념](#)이 Python에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다.

## Python 런타임

이름	식별자	Python용 AWS SDK	운영 체제
Python 3.6	python3.6	boto3-1.7.74 botocore-1.10.74	Amazon Linux
Python 3.7	python3.7	boto3-1.9.42 botocore-1.12.42	Amazon Linux
Python 2.7	python2.7	해당 사항 없음	Amazon Linux

## 주제

- [AWS Lambda 배포 패키지\(Python\) \(p. 245\)](#)
- [AWS Lambda 함수 핸들러\(Python\) \(p. 249\)](#)
- [AWS Lambda 콘텍스트 객체\(Python\) \(p. 250\)](#)
- [AWS Lambda 함수 로깅\(Python\) \(p. 251\)](#)
- [AWS Lambda 함수 오류\(Python\) \(p. 252\)](#)
- [AWS Lambda에서 Python 코드 계측 \(p. 254\)](#)

## AWS Lambda 배포 패키지(Python)

배포 패키지는 함수 코드 및 종속성이 포함되어 있는 ZIP 아카이브 파일입니다. Lambda API를 사용하여 함수를 관리하거나 코드에서 AWS SDK 이외의 라이브러리를 사용하는 경우 배포 패키지를 생성해야 합니다. 다른 라이브러리 및 종속성을 배포 패키지에 포함해야 합니다. 패키지를 Lambda에 직접 업로드하거나 Amazon S3 버킷을 사용하여 Lambda에 업로드할 수 있습니다.

Lambda [콘솔 편집기 \(p. 24\)](#)를 사용하여 함수를 작성하는 경우 콘솔에서 배포 패키지를 관리합니다. 라이브러리를 추가할 필요가 없는 한 이 방법을 사용할 수 있습니다. 또한 총 크기가 3 MB를 초과하지 않는 한, 이 방법을 사용하여 이미 배포 패키지에 라이브러리가 있는 함수를 업데이트할 수 있습니다.

### Note

AWS SAM CLI `build` 명령을 사용하여 Python 함수 코드 및 종속 프로그램을 위한 배포 패키지를 만들 수 있습니다. 자세한 내용은 AWS SAM 개발자 안내서에서 [종속 프로그램을 포함하는 애플리케이션 빌드](#)를 참조하십시오.

### 섹션

- [추가 종속 프로그램 제외 \(p. 246\)](#)

- 추가 종속 프로그램 포함 (p. 246)
- 가상 환경 포함 (p. 247)

## 추가 종속 프로그램 제외

Lambda API를 사용하여 함수를 생성하거나 업데이트하려면 함수 코드를 포함하는 아카이브를 생성한 후 AWS CLI를 사용하여 이 아카이브를 업로드합니다.

종속 프로그램이 없는 Python 함수를 업데이트하려면

1. ZIP 아카이브를 생성합니다.

```
~/my-function$ zip function.zip function.py
```

2. update-function-code 명령을 사용하여 패키지를 업로드합니다.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
    "Runtime": "python3.7",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 815,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-20T20:41:16.647+0000",
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXMx5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "d1e983e3-ca8e-434b-8dc1-7add83d72ebd"
}
```

## 추가 종속 프로그램 포함

함수가 Python용 SDK(Boto3) 외의 다른 라이브러리를 사용할 경우 [pip](#)를 사용하여 로컬 디렉터리에 해당 라이브러리를 설치한 후 배포 패키지에 포함시키십시오.

종속 프로그램을 포함하는 Python 함수를 업데이트하려면

1. 종속 프로그램을 위한 디렉터리를 만듭니다.

```
~/my-function$ mkdir package
```

2. --target 옵션을 사용하여 패키지 디렉터리에 라이브러리를 설치합니다.

```
~/my-function$ cd package
~/my-function/package$ pip install Pillow --target .
```

```
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
  Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
  Installing collected packages: Pillow
  Successfully installed Pillow-5.3.0
```

3. ZIP 아카이브를 생성합니다.

```
package$ zip -r9 ..function.zip .
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
  adding: PIL/.libs/liblcms2-a6801db4.so.2.0.8 (deflated 67%)
  ...
```

4. 아카이브에 함수 코드를 추가합니다.

```
~/my-function/package$ cd ../
~/my-function$ zip -g function.zip function.py
  adding: function.py (deflated 56%)
```

5. 함수 코드를 업데이트합니다.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file
fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
    "Runtime": "python3.7",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSize": 2269409,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-11-20T20:51:35.871+0000",
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [],
        "SecurityGroupIds": [],
        "VpcId": ""
    },
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "a9c05ffd-8ad6-4d22-b6cd-d34a00c1702c"
}
```

## 가상 환경 포함

경우에 따라 함수에 대한 종속 프로그램을 설치하기 위해 [가상 환경](#)을 사용해야 할 수 있습니다. 이 상황은 해당 기능 또는 그 종속 프로그램이 기본 라이브러리에 종속되어 있거나 Homebrew를 사용하여 Python을 설치한 경우에 발생할 수 있습니다.

가상 환경을 사용하여 Python 함수를 업데이트하려면

1. 가상 환경을 생성합니다.

```
~/my-function$ virtualenv v-env
Using base prefix '/.local/python-3.7.0'
New python executable in v-env/bin/python3.7
Also creating executable in v-env/bin/python
Installing setuptools, pip, wheel...
done.
```

- 환경을 활성화합니다.

```
~/my-function$ source v-env/bin/activate
(v-env) ~/my-function$
```

Windows 명령줄의 경우 활성화 스크립트는 Scripts 디렉터리에 있습니다.

```
> v-env\Scripts\activate.bat
```

- pip를 사용하여 라이브러리를 설치합니다.

```
~/my-function$ pip install Pillow
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
  Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
  Installing collected packages: Pillow
  Successfully installed Pillow-5.3.0
```

- 가상 환경을 비활성화합니다.

```
(v-env)~/my-function$ deactivate
```

- 라이브러리의 내용을 포함하는 ZIP 아카이브를 만듭니다.

```
~/my-function$ cd v-env/lib/python3.7/site-packages/
~/my-function/v-env/lib/python3.7/site-packages$ zip -r9 ../../../../function.zip .
  adding: easy_install.py (deflated 17%)
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
  ...
```

라이브러리에 따라 site-packages 또는 dist-packages에 종속 항목이 나타날 수 있으며 가상 환경의 첫 폴더는 lib 또는 lib64가 될 수 있습니다. pip show 명령을 사용하여 특정 패키지를 찾을 수 있습니다.

- 아카이브에 함수 코드를 추가합니다.

```
~/my-function/v-env/lib/python3.7/site-packages$ cd ../../..
~/my-function$ zip -g function.zip function.py
  adding: function.py (deflated 56%)
```

- 함수 코드를 업데이트합니다.

```
~/my-function$ aws lambda update-function-code --function-name python37 --zip-file
fileb://function.zip
{
    "FunctionName": "python37",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:python37",
    "Runtime": "python3.7",
```

```
"Role": "arn:aws:iam::123456789012:role/lambda-role",
"Handler": "function.handler",
"CodeSize": 5912988,
"Description": "",
"Timeout": 3,
"MemorySize": 128,
"LastModified": "2018-11-20T21:08:26.326+0000",
"CodeSha256": "A2PONUWq1J+LtSbkuP8tm9uNYqs1TAa3M76ptmZCw5g=",
"Version": "$LATEST",
"VpcConfig": {
    "SubnetIds": [],
    "SecurityGroupIds": [],
    "VpcId": ""
},
"TracingConfig": {
    "Mode": "Active"
},
"RevisionId": "5afdc7dc-2fcb-4ca8-8f24-947939ca707f"
}
```

## AWS Lambda 함수 핸들러(Python)

Lambda 함수를 만드는 시점에 서비스가 코드를 실행할 때 AWS Lambda가 호출할 수 있는 코드의 함수인 핸들러를 지정합니다. Python에서 핸들러 함수를 생성할 때 다음과 같은 일반적인 구문 구조를 사용합니다.

```
def handler_name(event, context):
    ...
    return some_value
```

구문에서 다음 사항에 유의하십시오.

- **event** – AWS Lambda는 이 파라미터를 사용하여 이벤트 데이터를 핸들러에 전달합니다. 이 파라미터는 일반적인 Python dict 유형입니다. 또한 list, str, int, float 또는 NoneType 유형이 될 수 있습니다.
- **context** – AWS Lambda는 이 파라미터를 사용하여 런타임 정보를 핸들러에 제공합니다. 이 파라미터는 LambdaContext 유형입니다.
- 선택적으로, 핸들러는 값을 반환할 수 있습니다. 반환된 값은 Lambda 함수를 호출할 때 사용하는 호출 유형에 따라 달라집니다.
  - **RequestResponse** 호출 유형(동기식 실행)을 사용하는 경우에는 AWS Lambda가 Python 함수 호출의 결과를 클라이언트에 반환하여 Lambda 함수를 호출합니다(호출 요청에 대한 HTTP 응답이 JSON에 직렬화). 예를 들어 AWS Lambda 콘솔은 RequestResponse 호출 유형을 사용하기 때문에 콘솔을 사용하여 함수를 호출할 때 콘솔에 반환 값이 표시됩니다.

핸들러가 **NONE**을 반환하면 AWS Lambda가 **null**을 반환합니다.

- **Event** 호출 유형(비동기식 실행)을 사용하는 경우에는 해당 값이 폐기됩니다.

예를 들어 다음과 같은 Python 코드를 고려해 보십시오.

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
                                    event['last_name'])
    return {
        'message' : message
    }
```

이 예제에는 **my\_handler**라는 하나의 함수만 있습니다. 이 함수는 입력으로 받은 이벤트의 데이터가 포함된 메시지를 반환합니다.

## AWS Lambda 콘텍스트 객체(Python)

Lambda은 함수를 실행할 때 컨텍스트 객체를 [핸들러 \(p. 249\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 제공하는 메서드 및 속성들을 제공합니다.

### 컨텍스트 메서드

- `get_remaining_time_in_millis` – 실행 시간이 초과되기 전에 남은 시간(밀리초)을 반환합니다.

### 컨텍스트 속성

- `function_name` – Lambda 함수의 이름
- `function_version` – 함수의 버전 ([p. 45](#))
- `invoked_function_arn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `memory_limit_in_mb` – 함수에 구성된 메모리 양
- `aws_request_id` – 호출 요청의 식별자
- `log_group_name` – 함수에 대한 로그 그룹
- `log_stream_name` – 함수 인스턴스에 대한 로그 스트림
- `identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
  - `cognito_identity_id` – 인증된 Amazon Cognito 자격 증명
  - `cognito_identity_pool_id` – 호출에 대한 권한을 부여한 Amazon Cognito ID 풀
- `client_context` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`
- `custom` – 모바일 클라이언트 애플리케이션에서 설정된 사용자 지정 값의 dict입니다.
- `env` – AWS SDK가 제공하는 환경 정보의 dict입니다.

다음 예제는 컨텍스트 정보를 기록하는 핸들러 함수를 보여줍니다.

### 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())
```

상기에 열거한 옵션들 외에도 [AWS Lambda에서 Python 코드 계측 \(p. 254\)](#)용 AWS X-Ray SDK를 사용하면 중요한 코드 경로를 식별하고 그 성능을 추적하며 분석용 데이터를 수집할 수도 있습니다.

## AWS Lambda 함수 로깅(Python)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드에서 로그를 출력하려면 [인쇄 메서드](#)를 사용하거나, `stdout` 또는 `stderr`에 쓰는 로깅 라이브러리를 사용합니다. 다음 예제는 환경 변수의 값과 이벤트 객체를 로깅합니다.

Example `lambda_function.py`

```
import json
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

자세한 로그를 확인하려면 [로깅 라이브러리를](#) 사용합니다.

```
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)
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

`logger`의 출력에는 로그 레벨, 타임스탬프와 요청 ID가 포함됩니다.

```
[INFO] 2019-04-21T23:24:14.135Z 00d3cdad-8aaf-42b2-af4e-6f8b2cae00a5 ## EVENT
[INFO] 2019-04-21T23:24:14.135Z 00d3cdad-8aaf-42b2-af4e-6f8b2cae00a5 {'key1': 'value1',
'key2': 'value2', 'key3': 'value3'}
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스](#) (p. 101)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션](#) (p. 116) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 `--log-type` 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 `LogResult` 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms          Billed
Duration: 100 ms      Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 `base64 -D`입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(Python)

Lambda 함수가 예외를 발생시키면 AWS Lambda는 실패를 인식하고 예외 정보를 JSON으로 직렬화하여 반환합니다. 다음 예제를 고려하십시오.

```
def always_failed_handler(event, context):
    raise Exception('I failed!')
```

이 Lambda 함수를 호출하면 예외가 발생하고 AWS Lambda는 다음과 같은 오류 메시지를 반환합니다.

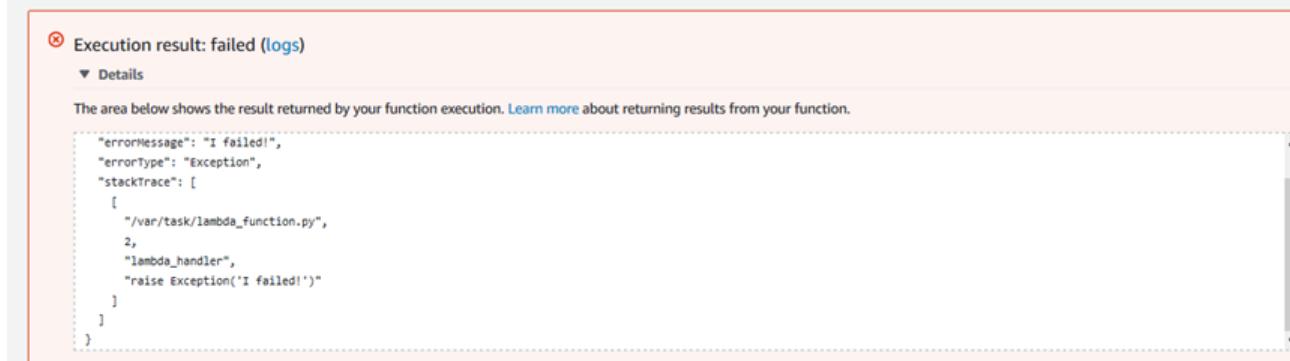
```
{
    "errorMessage": "I failed!",
    "stackTrace": [
        [
            "/var/task/lambda_function.py",
            3,
            "my_always_fails_handler",
            "raise Exception('I failed!')"
        ]
    ],
    "errorType": "Exception"
}
```

스택 추적은 스택 추적 요소의 stackTrace JSON 배열로 반환됩니다.

오류 정보를 다시 가져오는 방법은 클라이언트가 함수를 호출할 때 지정하는 호출 유형에 따라 다릅니다.

- 클라이언트가 호출 유형을 RequestResponse로 지정한 경우(동기식 실행), 호출을 한 클라이언트로 결과가 반환됩니다.

예를 들어 콘솔은 항상 RequestResponse 호출 유형을 사용하기 때문에 콘솔은 [Execution result] 섹션에 다음과 같이 오류 메시지를 표시합니다.



같은 정보가 CloudWatch에도 전송되며 로그 출력 섹션에 같은 로그가 표시됩니다.



- 클라이언트가 호출 유형을 Event(즉, 비동기식 실행)로 지정한 경우에는 AWS Lambda가 어떤 결과도 반환하지 않습니다. 대신, 오류 정보를 CloudWatch Logs에 기록합니다. CloudWatch Metrics에서 오류 측정치를 볼 수도 있습니다.

이벤트 소스에 따라 AWS Lambda는 실패한 Lambda 함수를 다시 시도할 수 있습니다. 예를 들어 Kinesis가 이벤트 소스인 경우, AWS Lambda는 Lambda 함수가 성공하거나 스트림의 레코드가 만료될 때까지 실패한 호출을 재시도합니다.

콘솔에서 위의 Python 코드를 테스트하려면

- 콘솔에서 hello-world 블루프린트를 사용하여 Lambda 함수를 생성합니다. 런타임에 Python 3.7을 선택합니다. [Handler]에서 `lambda_function.lambda_handler`를 `lambda_function.always_failed_handler`로 대체합니다. 작업 방법에 대한 지침은 콘솔로 [Lambda 함수 만들기 \(p. 3\)](#) 단원을 참조하십시오.
- 템플릿 코드를 이 단원에서 제공되는 코드로 대체합니다.
- Lambda 콘솔에서 제공되는 Hello World라는 샘플 이벤트 템플릿을 사용하여 Lambda 함수를 테스트합니다.

## 함수 오류 처리

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 AWS Step Functions 상태 시스템 내에서 직접 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

`CreateAccount` 상태가 Lambda 함수를 사용하여 고객의 세부 정보를 데이터베이스에 기록하는 작업인 경우를 생각해 보십시오.

- 작업이 성공하면 계정이 만들어지고 환영 이메일이 전송됩니다.
- 사용자가 이미 존재하는 사용자 이름에 대해 계정을 만들려고 시도하면 Lambda 함수는 오류를 발생시켜 상태 시스템이 다른 사용자 이름을 제안하고 계정 생성 프로세스를 재시도하도록 만듭니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다. Python의 사용자 지정 오류는 `Exception` 클래스를 확장해야 합니다.

```
class AccountAlreadyExistsException(Exception): pass

def create_account(event, context):
    raise AccountAlreadyExistsException('Account is in use!')
```

`Catch` 규칙을 사용하여 오류를 파악하도록 Step Functions를 구성할 수 있습니다. Lambda는 런타임에 오류 이름을 예외의 간단한 클래스 이름으로 자동 설정합니다.

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": ["AccountAlreadyExistsException"],
                    "Next": "SuggestAccountName"
                }
            ],
            ...
        }
    }
}
```

런타임에 AWS Step Functions는 오류를 파악하여 `Next` 전환에 지정된 대로 `SuggestAccountName` 상태로 전환을 합니다.

사용자 지정 오류 처리는 [서비스 애플리케이션](#)을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda [프로그래밍 모델](#) (p. 31)에서 지원되는 모든 언어와 통합이 되기 때문에 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서비스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

## AWS Lambda에서 Python 코드 계측

Python의 경우, Lambda가 X-Ray에 하위 세그먼트를 방출하여 함수에서 이루어진 다른 AWS 서비스에 대한 다운스트림 호출 관련 정보를 표시합니다. 이를 위해서는 먼저 배포 패키지에 [Python용 AWS X-Ray SDK](#)를

포함시켜야 합니다. 뿐만 아니라, boto3(또는 세션을 사용하는 경우에는 botocore)을 패치할 수 있기 때문에 다른 AWS 서비스를 액세스하기 위해 생성한 모든 클라이언트를 X-Ray가 자동으로 트레이스합니다.

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])
```

클라이언트를 생성하기 위해 사용 중인 모듈을 패치했다면, 이제 패치한 모듈을 사용해 트레이스되는 클라이언트를 생성할 수 있습니다(Amazon S3 이하인 경우).

```
s3_client = boto3.client('s3')
```

Python용 X-Ray SDK는 호출에 대한 하위 세그먼트를 생성하고 요청 및 응답에서 나온 정보를 기록합니다. 다음 Lambda 함수에 표시된 대로 aws\_xray\_sdk\_sdk.core.xray\_recorder를 사용하여 Lambda 함수를 데코레이트해 자동으로, 또는 함수 내에서 xray\_recorder.begin\_subsegment() 및 xray\_recorder.end\_subsegment()를 호출해 수동으로 하위 세그먼트를 생성할 수 있습니다.

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    bucket_name = event['bucket_name']
    bucket_key = event['bucket_key']
    body = event['body']

    put_object_into_s3(bucket_name, bucket_key, body)
    get_object_from_s3(bucket_name, bucket_key)

# Define subsegments manually
def put_object_into_s3(bucket_name, bucket_key, body):
    try:
        xray_recorder.begin_subsegment('put_object')
        response = s3_client.put_object(Bucket=bucket_name, Key=bucket_key, Body=body)
        status_code = response['ResponseMetadata']['HTTPStatusCode']
        xray_recorder.current_subsegment().put_annotation('put_response', status_code)
    finally:
        xray_recorder.end_subsegment()

# Use decorators to automatically set the subsegments
@xray_recorder.capture('get_object')
def get_object_from_s3(bucket_name, bucket_key):
    response = s3_client.get_object(Bucket=bucket_name, Key=bucket_key)
    status_code = response['ResponseMetadata']['HTTPStatusCode']
    xray_recorder.current_subsegment().put_annotation('get_response', status_code)
```

### Note

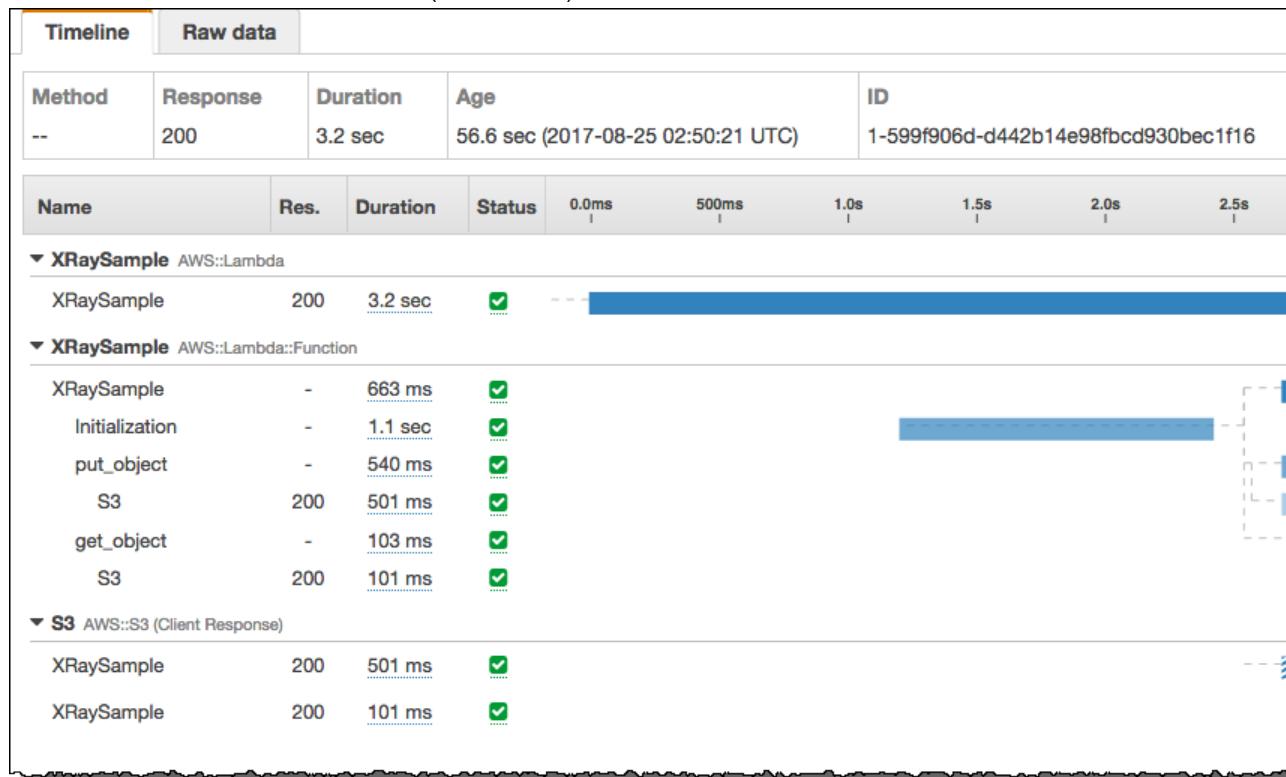
Python용 X-Ray SDK를 사용하면 다음 모듈을 패치할 수 있습니다.

- botocore
- boto3

- 요청 건
- sqlite3
- mysql

`patch_all()`을 사용하여 모든 모듈을 한 번에 패치할 수 있습니다.

다음은 이전 코드에서 방출된 트레이스(동기식 호출)를 나타낸 것입니다.



# Java를 사용하여 Lambda 함수 빌드

다음 단원에서는 [일반적인 프로그래밍 패턴 및 핵심 개념](#)이 Java에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다.

## Java 런타임

이름	식별자	JDK	운영 체제
Java 8	java8	java-1.8.0-openjdk	Amazon Linux

## 주제

- [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#)
- [AWS Lambda 함수 핸들러\(Java\) \(p. 265\)](#)
- [AWS Lambda 컨텍스트 객체\(Java\) \(p. 274\)](#)
- [AWS Lambda 함수 로깅\(Java\) \(p. 275\)](#)
- [AWS Lambda 함수 오류\(Java\) \(p. 278\)](#)
- [AWS Lambda에서 Java 코드 계측 \(p. 280\)](#)
- [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#)

뿐만 아니라, AWS Lambda는 다음과 같은 라이브러리를 제공합니다.

- aws-lambda-java-core – 이 라이브러리는 컨텍스트 객체 `RequestStreamHandler` 및 `RequestHandler` 인터페이스를 제공합니다. `Context` 객체([AWS Lambda 컨텍스트 객체 \(Java\) \(p. 274\)](#))는 Lambda 함수에 대한 런타임 정보를 제공합니다. 사전 정의된 인터페이스는 함수 핸들러를 정의하는 한 가지 방법을 제공합니다. 자세한 내용은 [핸들러를 생성하기 위해 사전 정의된 인터페이스 활용\(Java\) \(p. 271\)](#) 단원을 참조하십시오.
- aws-lambda-java-events – 이 라이브러리는 Amazon S3, Kinesis, Amazon SNS 및 Amazon Cognito가 개시한 이벤트를 처리하도록 Lambda 함수를 작성할 때 사용할 수 있도록 사전 정의된 유형을 제공합니다. 이를 클래스를 사용하면 사용자 지정 직렬화 로직을 자체적으로 작성하지 않고도 이벤트를 처리할 수 있습니다.
- Log4j2.8용 사용자 지정 Appender – Lambda 함수에서의 로깅을 위해 AWS Lambda에서 제공되는 사용자 지정 Log4j([Apache Log4j 2 참조](#)) appender를 사용할 수 있습니다. `log.info()` 또는 `log.error()`와 같은 Log4j 메서드를 호출할 때마다 CloudWatch Logs 이벤트가 발생합니다. 사용자 지정 appender를 `LambdaAppender`라고 하며, `log4j2.xml` 파일에서 사용되어야 합니다. 배포 패키지(.jar 파일)에 `aws-lambda-java-log4j2` 아티팩트(`artifactId:aws-lambda-java-log4j2`)를 포함해야 합니다. 자세한 내용은 [AWS Lambda 함수 로깅\(Java\) \(p. 275\)](#) 단원을 참조하십시오.
- Log4j1.2용 사용자 지정 Appender – Lambda 함수에서의 로깅을 위해 AWS Lambda에서 제공되는 사용자 지정 Log4j([Apache Log4j 1.2 참조](#)) appender를 사용할 수 있습니다. 자세한 내용은 [AWS Lambda 함수 로깅\(Java\) \(p. 275\)](#) 단원을 참조하십시오.

## Note

Log4j v1.2 사용자 지정 appender에 대한 지원이 EOL(End of Life)로 표시되었습니다. 지속적인 업데이트를 받을 수 없으며 사용하지 않는 것이 좋습니다.

이들 라이브러리는 [Maven Central Repository](#)를 통해 제공되며 [GitHub](#)에서도 찾아볼 수 있습니다.

## AWS Lambda 배포 패키지(Java)

배포 패키지는 .zip 파일 또는 독립 실행형 jar 파일 중 원하는 형식을 선택할 수 있습니다. 익숙한 빌드 및 패키징 도구를 사용하여 배포 패키지를 생성할 수 있습니다.

Maven을 사용하여 독립 실행형 jar 파일을 생성하거나 Gradle을 사용하여 .zip 파일을 생성하는 예제가 여기 나와 있습니다. 자세한 내용은 다음 주제를 참조하십시오.

### 주제

- [IDE 없이 Maven을 사용하여 .jar 배포 패키지 만들기\(Java\) \(p. 258\)](#)
- [Maven 및 Eclipse IDE를 사용하여 .jar 배포 패키지 생성\(Java\) \(p. 260\)](#)
- [Java 함수용 ZIP 배포 패키지 만들기 \(p. 262\)](#)
- [Eclipse IDE 및 AWS SDK 플러그인을 사용하여 Lambda 함수 작성\(Java\) \(p. 265\)](#)

## IDE 없이 Maven을 사용하여 .jar 배포 패키지 만들기 (Java)

이 단원에서는 명령줄에서 Maven을 사용하여 Java 코드를 배포 패키지에 패키징하는 방법을 보여 줍니다.

### 주제

- [시작하기 전에 \(p. 258\)](#)
- [프로젝트 구조 개요 \(p. 258\)](#)
- [1단계: 프로젝트 생성 \(p. 259\)](#)
- [2단계: 프로젝트 빌드\(배포 패키지 생성\) \(p. 260\)](#)

## 시작하기 전에

Maven 명령줄 빌드 도구를 설치해야 합니다. 자세한 내용은 [Maven](#)을 참조하십시오. Linux를 사용하는 경우 패키지 관리자를 확인합니다.

```
sudo apt-get install mvn
```

Homebrew를 사용하는 경우

```
brew install maven
```

## 프로젝트 구조 개요

프로젝트를 설정한 후에는 다음과 같은 폴더 구조가 있어야 합니다.

```
project-dir/pom.xml  
project-dir/src/main/java/ (your code goes here)
```

코드는 /java 폴더에 있습니다. 예를 들어 패키지 이름이 example이고 Hello.java 클래스가 있는 경우 구조는 다음과 같습니다.

```
project-dir/src/main/java/example>Hello.java
```

프로젝트를 빌드한 후 결과 .jar 파일(즉, 배포 패키지)은 *project-dir*/target 하위 디렉터리에 있습니다.

## 1단계: 프로젝트 생성

이번 섹션의 단계를 따르면 Java 프로젝트를 생성할 수 있습니다.

1. 프로젝트 디렉터리(*project-dir*)를 생성합니다.
2. *project-dir* 디렉터리에서 다음과 같이 생성합니다.
  - Project Object Model 파일, pom.xml. Maven에 대한 다음 프로젝트 정보 및 구성 정보를 추가하여 프로젝트를 빌드합니다.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>doc-examples</groupId>
  <artifactId>lambda-java-example</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>lambda-java-example</name>

  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.1.0</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

### Note

- dependencies 섹션에서 groupId(즉, com.amazonaws)는 Maven 중앙 리포지토리에 있는 Maven 아티팩트에 대한 Amazon AWS 그룹 ID입니다. artifactId(즉, aws-lambda-java-core)는 Java 애플리케이션에서 사용하기 위해 RequestHandler, RequestStreamHandler 및 Context AWS Lambda 인터페이스의 정의를 제공하는 AWS Lambda 코어 라이브러리입니다. 빌드 시 Maven은 이러한 종속성을 해결합니다.

- 플러그인 섹션에서 Apache maven-shade-plugin은 Maven이 빌드 프로세스 중에 다른 로드하여 사용할 플러그인입니다. 이 플러그인은 jar를 패키징하여 배포 패키지인 드립 실행형 .jar(.zip 파일)을 만드는 데 사용됩니다.
- 이 안내서의 다른 자습서 주제를 수행하는 중이면 특정 자습서에서 종속성을 더 추가해야 할 수도 있습니다. 필요에 따라 이러한 종속성을 추가합니다.

3. *project-dir*에서 다음 구조를 만듭니다.

```
project-dir/src/main/java
```

4. /java 하위 디렉터리에 Java 파일과 폴더 구조가 있으면 추가합니다. 예를 들어 Java 패키지 이름이 example이고 소스 코드가 Hello.java인 경우 디렉터리 구조는 다음과 같습니다.

```
project-dir/src/main/java/example>Hello.java
```

## 2단계: 프로젝트 빌드(배포 패키지 생성)

이제 명령줄에서 Maven을 사용하여 프로젝트를 빌드할 수 있습니다.

1. 명령줄 프롬프트에서 디렉터리를 프로젝트 디렉터리로 변경합니다(\dev).
2. 다음 mvn 명령을 실행하여 프로젝트를 빌드합니다.

```
$ mvn package
```

결과로 나타나는 .jar는 *project-dir*/target/lambda-java-example-1.0-SNAPSHOT.jar로 저장됩니다. jar 이름은 pom.xml 파일의 artifactId 및 version을 연결하여 생성됩니다.

해당 빌드는 필요한 변환을 수행하기 위해 pom.xml에 있는 정보를 사용하여 이러한 결과 .jar 파일을 만듭니다. 이는 모든 종속성을 포함하는 드립 실행형 .jar(.zip 파일)입니다. 이는 Lambda 함수를 생성하기 위해 AWS Lambda에 업로드할 수 있는 배포 패키지입니다.

## Maven 및 Eclipse IDE를 사용하여 .jar 배포 패키지 생성(Java)

이 단원에서는 Eclipse IDE 및 Eclipse용 Maven 플러그인을 사용하여 배포 패키지에 Java 코드를 패키징하는 방법을 보여 줍니다.

주제

- [시작하기 전에 \(p. 260\)](#)
- [1단계: 프로젝트 생성 및 빌드 \(p. 261\)](#)

## 시작하기 전에

Eclipse용 [Maven] 플러그인을 설치합니다.

1. Eclipse를 시작합니다. Eclipse의 [Help] 메뉴에서 [Install New Software]를 선택합니다.
2. 설치 창의 Work with:(작업 대상:) 상자에 <http://download.eclipse.org/technology/m2e/releases>를 입력하고 추가를 선택합니다.
3. 단계에 따라 설정을 완료합니다.

## 1단계: 프로젝트 생성 및 빌드

이 단계에서는 Eclipse를 시작하고 Maven 프로젝트를 생성합니다. 필요한 종속 프로그램을 추가하고 프로젝트를 빌드합니다. 빌드는 배포 패키지인 .jar를 만듭니다.

1. Eclipse에서 Maven 프로젝트를 새로 생성합니다.
  - a. [File] 메뉴에서 [New]를 선택한 다음, [Project]를 선택합니다.
  - b. [New Project] 창에서 [Maven Project]를 선택합니다.
  - c. [New Maven Project] 창에서 [Create a simple project]를 선택하고 다른 기본 선택은 그대로 둡니다.
  - d. [New Maven Project]의 [Configure project] 창에서 다음과 같은 [Artifact] 정보를 입력합니다.
    - [Group Id]: doc-examples
    - [Artifact Id]: lambda-java-example
    - [Version]: 0.0.1-SNAPSHOT
    - [Packaging]: jar
    - [Name]: lambda-java-example
2. aws-lambda-java-core 종속 프로그램을 pom.xml 파일에 추가합니다.

RequestHandler, RequestStreamHandler 및 Context 인터페이스의 정의를 제공합니다. 이렇게 하면 에서 사용할 수 있는 코드를 컴파일할 수 있습니다.

- a. 마우스 오른쪽 버튼을 클릭하여 pom.xml 파일의 컨텍스트 메뉴를 열고 [Maven]을 선택한 다음, [Add Dependency]를 선택합니다.
- b. [Add Dependency] 창에서 다음 값을 입력합니다.

[Group Id:] com.amazonaws

[Artifact Id:] aws-lambda-java-core

버전: 1.1.0

### Note

이 안내서의 다른 자습서 주제를 수행하는 중이면 특정 자습서에서 종속성을 더 추가해야 할 수도 있습니다. 필요에 따라 이러한 종속성을 추가합니다.

3. 프로젝트에 Java 클래스를 추가합니다.
  - a. 마우스 오른쪽 버튼을 클릭하여 프로젝트의 src/main/java 하위 디렉터리 컨텍스트 메뉴를 열고 [New]를 선택한 다음, [Class]를 선택합니다.
  - b. [New Java Class] 창에서 다음 값을 입력합니다.
    - 패키지: **example**
    - 이름: **Hello**

### Note

이 안내서의 다른 자습서 주제를 따르고 있는 중이라면 다른 패키지 이름이나 클래스 이름을 사용하는 것이 좋습니다.

- c. Java 코드를 추가합니다. 이 안내서의 다른 자습서 주제를 따르고 있는 중이라면 제공된 코드를 추가합니다.

4. 프로젝트를 빌드합니다.

---

[Package Explorer]에서 마우스 오른쪽 버튼을 클릭하여 프로젝트의 컨텍스트 메뉴를 열고 [Run As]를 선택한 다음, [Maven Build ...]를 선택합니다.<sup>261</sup> 구성 편집 창의 목표 상자에 **package**를 입력합니다.

### Note

그 결과 생성된 .jar, `lambda-java-example-0.0.1-SNAPSHOT.jar`는 배포 패키지로 사용할 수 있는 최종적인 독립 실행형 .jar이 아닙니다. 다음 단계에서 Apache maven-shade-plugin를 추가하여 독립 실행형 .jar를 생성합니다. 자세한 내용은 [Apache Maven Shade 플러그인](#)을 참조하십시오.

5. maven-shade-plugin 플러그인을 추가하고 다시 빌드합니다.

maven-shade-plugin은 패키지 목적으로 만든 아티팩트(jar)를 가져와서 컴파일된 고객 코드와 `pom.xml`에서 확인된 종속 프로그램을 포함하는 독립 실행형 .ajr를 생성합니다.

- a. 마우스 오른쪽 버튼을 클릭하여 `pom.xml` 파일의 컨텍스트 메뉴를 열고 [Maven]을 선택한 다음, [Add Plugin]을 선택합니다.
- b. [Add Plugin] 창에서 다음 값을 입력합니다.
  - [Group Id:] org.apache.maven.plugins
  - [Artifact Id:] maven-shade-plugin
  - 버전: 2.3
- c. 이제 다시 빌드합니다.

이번에는 전과 같이 jar를 생성하고 maven-shade-plugin을 사용하여 독립 실행형 .jar를 만들기 위한 종속 프로그램을 추가합니다.

- i. 마우스 오른쪽 버튼을 클릭하여 프로젝트의 컨텍스트 메뉴를 열고 [Run As]를 선택한 다음, [Maven build ...]를 선택합니다.
- ii. 구성 편집 창의 목표 상자에 `package shade:shade`를 입력합니다.
- iii. Run을 선택합니다.

그 결과, `/target` 하위 디렉터리에서 독립 실행형 .jar(즉, 배포 패키지)를 찾아볼 수 있습니다.

마우스 오른쪽 버튼을 클릭하여 `/target` 하위 프로젝트의 컨텍스트 메뉴를 열고 [Show In]을 선택한 다음 [System Explorer]를 선택하면 `lambda-java-example-0.0.1-SNAPSHOT.jar`가 표시됩니다.

## Java 함수용 ZIP 배포 패키지 만들기

이 단원에서는 배포 패키지로서 .zip 파일을 생성하는 작업에 대한 예시를 제공합니다. 원하는 빌드 및 패키징 도구를 사용하여 다음 구조를 갖춘 배포 패키지를 만들 수 있습니다.

- 루트 수준에 있는 모든 컴파일된 클래스 파일과 리소스 파일.
- `/lib` 디렉터리에서 코드를 실행하는 데 필요한 모든 jar.

### Note

Lambda는 JAR를 유니코드 알파벳순으로 로드합니다. lib 폴더의 여러 JAR 파일에 동일한 클래스가 있을 경우 첫 번째 클래스가 사용됩니다. 다음 셀 스크립트를 사용하여 중복 클래스를 식별할 수 있습니다.

#### 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
```

다음 예에서는 Gradle 빌드 및 배포 도구를 사용하여 배포 패키지를 만듭니다.

## 시작하기 전에

Gradle 버전 2.0 이상을 다운로드해야 합니다. 자침은 Gradle 웹 사이트 <https://gradle.org/>를 참조하십시오.

## 예제 1: Gradle과 Maven Central Repository를 사용하여 .zip 만들기

이 연습이 끝나면 다음과 같은 구조의 콘텐츠가 있는 프로젝트 디렉터리(*project-dir*)를 갖게 됩니다.

```
project-dir/build.gradle  
project-dir/src/main/java/
```

/java 폴더는 코드를 포함합니다. 예를 들어 패키지 이름이 example이고 Hello.java 클래스가 있는 경우 구조는 다음과 같습니다.

```
project-dir/src/main/java/example>Hello.java
```

프로젝트를 빌드한 후 결과 .zip 파일(즉, 배포 패키지)은 *project-dir*/build/distributions 하위 디렉터리에 있습니다.

1. 프로젝트 디렉터리(*project-dir*)를 생성합니다.
2. *project-dir*에서 build.gradle 파일을 만들고 다음 콘텐츠를 추가합니다.

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

dependencies {
    compile (
        'com.amazonaws:aws-lambda-java-core:1.1.0',
        'com.amazonaws:aws-lambda-java-events:1.1.0'
    )
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.compileClasspath
    }
}
build.dependsOn buildZip
```

### Note

- 리포지토리 섹션은 Maven Central Repository를 참조합니다. 빌드 시, Maven Central에서 종속 항목(즉, 두 가지 AWS Lambda 라이브러리)을 가져옵니다.
- buildZip 작업은 배포 패키지 .zip 파일을 만드는 방법을 설명합니다.

예를 들어 결과 .zip 파일의 압축을 풀면 루트 수준에서 컴파일된 클래스 파일과 리소스 파일을 찾아야 합니다. 또한 코드를 실행하는 데 필요한 jar 파일이 있는 /lib 디렉터리를 찾아야 합니다.

- 이 안내서의 다른 자습서 주제를 수행하는 중이면 특정 자습서에서 종속성을 더 추가해야 할 수도 있습니다. 필요에 따라 이러한 종속성을 추가합니다.
3. *project-dir*에서 다음 구조를 만듭니다.

```
project-dir/src/main/java/
```

4. /java 하위 디렉터리에 Java 파일과 폴더 구조가 있으면 추가합니다. 예를 들어 Java 패키지 이름이 example이고 소스 코드가 Hello.java이면 디렉터리 구조는 다음과 같습니다.

```
project-dir/src/main/java/example>Hello.java
```

5. gradle 명령을 실행하여 .zip 파일에 프로젝트를 빌드하고 패키징합니다.

```
project-dir> gradle build
```

6. *project-dir*/build/distributions 하위 디렉터리에서 결과로 얻은 *project-dir.zip* 파일을 확인합니다.  
7. AWS Lambda에 .zip 파일을 배포 패키지에 업로드하여 Lambda 함수를 만들고 샘플 이벤트 데이터를 사용하여 수동으로 호출하는 방식으로 테스트할 수 있습니다. 자침은 [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 단원을 참조하십시오.

## 예제 2: 로컬 JAR를 통해 Gradle을 사용하여 .zip 만들기

Maven Central repository를 사용하지 않도록 선택할 수도 있습니다. 대신, 프로젝트 폴더에 모든 종속성을 가져옵니다. 이 경우 프로젝트 폴더(*project-dir*)의 구조는 다음과 같습니다.

```
project-dir/jars/          (all jars go here)
project-dir/build.gradle
project-dir/src/main/java/ (your code goes here)
```

Java 코드에 example 패키지 및 Hello.java 클래스가 있는 경우 코드는 다음 하위 디렉터리에 위치합니다.

```
project-dir/src/main/java/example>Hello.java
```

build.gradle 파일은 다음과 같아야 합니다.

```
apply plugin: 'java'

dependencies {
    compile fileTree(dir: 'jars', include: '*.jar')
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.compileClasspath
    }
}

build.dependsOn buildZip
```

종속성은 필요한 모든 jar를 포함하는 하위 디렉터리로서 *project-dir/jars*를 식별하는 fileTree를 지정합니다.

이제 패키지를 빌드합니다. gradle 명령을 실행하여 .zip 파일에 프로젝트를 빌드하고 패키징합니다.

```
project-dir> gradle build
```

## Eclipse IDE 및 AWS SDK 플러그인을 사용하여 Lambda 함수 작성(Java)

AWS SDK Eclipse Toolkit은 배포 패키지를 생성하고 업로드하여 Lambda 함수를 생성하는 Eclipse 플러그인을 제공합니다. Eclipse IDE를 개발 환경으로 사용할 수 있는 경우 이 플러그인을 사용하면 Java 코드를 작성하고, 배포 패키지를 작성 및 업로드하고, 함수를 생성할 수 있습니다. 자세한 내용은 [AWS Toolkit for Eclipse 시작 안내서](#)를 참조하십시오. Lambda 함수를 작성하기 위해 도구 키트를 사용하는 예제는 [AWS Toolkit for Eclipse에서 AWS Lambda 사용](#)을 참조하십시오.

## AWS Lambda 함수 핸들러(Java)

Lambda 함수를 생성하는 시점에 서비스가 사용자를 대신하여 Lambda 함수를 실행할 때 AWS Lambda가 호출할 수 있는 핸들러를 지정합니다.

Lambda는 핸들러 생성을 위해 두 가지 접근 방법을 지원합니다.

- 인터페이스를 구현할 필요 없이 직접 핸들러 메서드를 로딩합니다. 이 단원에서는 이 접근 방법에 대해 설명합니다.
- `aws-lambda-java-core` 라이브러리의 일부로 제공되는 표준 인터페이스를 구현합니다(인터페이스 접근 방법). 자세한 내용은 [핸들러를 생성하기 위해 사전 정의된 인터페이스 활용\(Java\) \(p. 271\)](#) 단원을 참조하십시오.

핸들러에 대한 일반 구문은 다음과 같습니다.

```
outputType handler-name(inputType input, Context context) {  
    ...  
}
```

AWS Lambda가 핸들러를 성공적으로 호출할 수 있으려면 `input` 파라미터의 데이터 유형으로 직렬화가 가능한 입력 데이터로 함수를 호출해야 합니다.

구문에서 다음 사항에 유의하십시오.

- `inputType` – 첫 번째 핸들러 파라미터는 핸들러에 대한 입력으로, 이벤트 데이터(이벤트 소스에서 게시 또는 문자열이나 사용자 지정 데이터 객체 같은 사용자 지정 입력이 될 수 있습니다. AWS Lambda가 핸들러를 성공적으로 호출할 수 있으려면 `input` 파라미터의 데이터 유형으로 직렬화가 가능한 입력 데이터를 함수를 호출해야 합니다.
- `outputType` – Lambda 함수를 동기식으로 호출할 계획인 경우(`RequestResponse` 호출 유형 사용), 지원되는 데이터 유형 중 하나를 사용하여 함수의 출력을 반환할 수 있습니다. 예를 들어 함수를 모바일 애플리케이션 백엔드로 사용하는 경우에는 이를 동기식으로 호출합니다. 출력 데이터 유형이 JSON으로 직렬화됩니다.

Lambda 함수를 비동기식으로 호출할 계획인 경우(`Event` 호출 유형 사용), `outputType`은 `void`여야 합니다. 예를 들어 Amazon S3나 Amazon SNS 같은 이벤트 소스에서 AWS Lambda를 사용할 경우, 이러한 이벤트 소스는 `Event` 호출 유형을 사용하여 Lambda 함수를 호출합니다.

- `inputType` 및 `outputType`은 다음 중 하나일 수 있습니다.
  - 기본 Java 유형(문자열 또는 `int`).
  - AWS 이벤트 유형은 `aws-lambda-java-events` 라이브러리에 사전 정의되어 있습니다.

예를 들어 `S3Event`는 라이브러리에 사전 정의된 POJO 중 하나로서, 수신 Amazon S3 이벤트에서 손쉽게 정보를 읽어 들일 수 있는 방법을 제공합니다.

- 또한 자체 POJO 클래스를 작성할 수도 있습니다. AWS Lambda는 POJO 유형에 따라 입력 및 출력 JSON을 자동으로 직렬화 및 역직렬화합니다.

자세한 내용은 [핸들러 입력/출력 유형\(Java\) \(p. 267\)](#) 단원을 참조하십시오.

- 필요하지 않을 경우 핸들러 메서드 서명에서 `Context` 객체를 생략할 수 있습니다. 자세한 내용은 [AWS Lambda 컨텍스트 객체\(Java\) \(p. 274\)](#) 단원을 참조하십시오.

예를 들어 다음과 같은 Java 코드의 예제를 고려해 보십시오.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Hello implements RequestHandler<Integer, String>{
    public String myHandler(int myCount, Context context) {
        return String.valueOf(myCount);
    }
}
```

이 예제에서 입력의 유형은 정수이고 출력의 유형은 문자열입니다. 이 코드와 종속성을 패키징하여 Lambda 함수를 생성하는 경우 `example.Hello::myHandler`(*package.class::method-reference*)를 핸들러로 지정합니다.

Java 코드 예제에서 첫 번째 핸들러 파라미터는 핸들러에 대한 입력(`myHandler`)인데, 이벤트 데이터(Amazon S3 같이 이벤트 소스에서 게시)나 정수 객체(예제 참조)나 사용자 지정 데이터 객체 같은 사용자 지정 입력이 될 수 있습니다.

이 Java 코드를 사용하여 Lambda 함수를 생성하는 방법에 대한 자침은 [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 단원을 참조하십시오.

## 핸들러 오버로드 해결

Java 코드에 `handler` 이름과 같은 이름의 메서드가 여러 개 포함되어 있으면 AWS Lambda는 다음의 규칙에 따라 호출할 메서드를 선택합니다.

- 파라미터 수가 가장 많은 메서드를 선택합니다.
- 두 개 이상의 메서드가 같은 수의 파라미터를 가지고 있는 경우, AWS Lambda는 `Context`를 마지막 파라미터로 가지고 있는 메서드를 선택합니다.

이들 메서드가 전부 `Context` 파라미터를 가지고 있거나 전부 이 파라미터를 가지고 있지 않을 경우, 동작이 정의되지 않습니다.

## 추가 정보

다음 주제에서는 핸들러에 대한 추가 정보를 제공합니다.

- 핸들러 입력 및 출력 유형에 대한 자세한 내용은 [핸들러 입력/출력 유형\(Java\) \(p. 267\)](#)을 참조하십시오.
- 사전 정의된 인터페이스를 사용하여 핸들러를 생성하는 방법에 대한 자세한 내용은 [핸들러를 생성하기 위해 사전 정의된 인터페이스 활용\(Java\) \(p. 271\)](#)을 참조하십시오.

이들 인터페이스를 구현한 경우에는 컴파일 시 핸들러 메서드 서명을 검증할 수 있습니다.

- Lambda 함수에서 예외가 발생한 경우, AWS Lambda는 CloudWatch에 측정치를 기록하여 오류가 발생했음을 나타냅니다. 자세한 내용은 [AWS Lambda 함수 오류\(Java\) \(p. 278\)](#) 단원을 참조하십시오.

## 핸들러 입력/출력 유형(Java)

AWS Lambda는 Lambda 함수를 실행할 때 핸들러를 호출합니다. 첫 번째 파라미터는 핸들러에 대한 입력으로, 이벤트 데이터(이벤트 소스에서 게시) 또는 문자열이나 사용자 지정 데이터 객체 같은 사용자 지정 입력이 될 수 있습니다.

AWS Lambda는 핸들러에 대해 다음과 같은 입력/출력 유형을 지원합니다.

- 단순한 Java 유형(AWS Lambda는 문자열, 정수, 부울, 맵, 목록 등의 유형을 지원)
- POJO(Plain Old Java Object) 유형
- 스트림 유형(POJO를 사용하고 싶지 않거나 Lambda의 직렬화 접근 방식이 요구를 충족시키지 않는 경우, 바이트 스트림 구현을 사용할 수 있습니다. 자세한 내용은 [예: 핸들러 입력/출력을 위한 스트림 사용\(Java\) \(p. 270\)](#) 단원을 참조하십시오.

### 핸들러 입력/출력: 문자열 유형

다음의 Java 클래스는 입력 및 출력에서 문자열 유형을 사용하는 `myHandler`라는 핸들러를 보여줍니다.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class Hello {
    public String myHandler(String name, Context context) {
        return String.format("Hello %s.", name);
    }
}
```

기타 단순한 Java 유형에서도 유사한 핸들러 기능을 사용할 수 있습니다.

#### Note

비동기식으로 Lambda 함수를 호출하면 Lambda 함수에 의한 반환 값이 무시됩니다. 따라서 코드에서 이를 명확하게 하기 위해 반환 유형을 무효로 설정하고 싶을 수 있습니다. 자세한 내용은 [Invoke \(p. 392\)](#) 단원을 참조하십시오.

예제 전체를 테스트하려면 [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 단원을 참조하십시오.

### 핸들러 입력/출력: POJO 유형

다음의 Java 클래스는 입력 및 출력에서 POJO 유형을 사용하는 `myHandler`라는 핸들러를 보여줍니다.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class HelloPojo {

    // Define two classes/POJOs for use with Lambda function.
    public static class RequestClass {
        ...
    }

    public static class ResponseClass {
        ...
    }
}
```

```
    }

    public static ResponseClass myHandler(RequestClass request, Context context) {
        String greetingString = String.format("Hello %s, %s.", request.getFirstName(),
request.getLastName());
        return new ResponseClass(greetingString);
    }
}
```

AWS Lambda는 표준 빔 이름 지정 규칙에 따라 직렬화를 수행합니다([Java EE 6 자습서](#) 참조). 퍼블릭 getter 및 setter를 통해 변경 가능한 POJO를 사용해야 합니다.

Note

주석 추가와 같은 직렬화 프레임워크의 여타 기능들을 사용할 수 없습니다. 직렬화 동작을 사용자 지정해야 하는 경우에는 원시 바이트 스트림을 통해 자체 직렬화를 사용할 수 있습니다.

입력 및 출력에 POJO를 사용하는 경우에는 RequestClass 및 ResponseClass 유형을 구현해야 합니다. 예제: 핸들러 입력/출력에서 POJO 사용(Java) (p. 268) 단원을 참조하십시오.

## 예제: 핸들러 입력/출력에서 POJO 사용(Java)

애플리케이션 이벤트가 다음과 같이 이름과 성이 포함된 데이터를 생성한다고 가정해 봅시다.

```
{ "firstName": "John", "lastName": "Doe" }
```

이 예제에서는 핸들러가 이 JSON을 수신하고 문자열 "Hello John Doe"를 반환합니다.

```
public static ResponseClass handleRequest(RequestClass request, Context context){
    String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
    return new ResponseClass(greetingString);
}
```

이 핸들러로 Lambda 함수를 생성하려면 다음의 Java 예제와 같이 입력 및 출력 유형을 구현해야 합니다. HelloPojo 클래스는 handler 메서드를 정의합니다.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class HelloPojo implements RequestHandler<RequestClass, ResponseClass>{

    public ResponseClass handleRequest(RequestClass request, Context context){
        String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
        return new ResponseClass(greetingString);
    }
}
```

입력 유형을 구현하려면 별도의 파일에 다음 코드를 추가하고 RequestClass.java라고 이름을 지정합니다. 디렉터리 구조의 HelloPojo.java 클래스 옆에 이를 배치합니다.

```
package example;

public class RequestClass {
    String firstName;
    String lastName;

    public String getFirstName() {
```

```
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public RequestClass(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public RequestClass() {
    }
}
```

출력 유형을 구현하려면 별도의 파일에 다음 코드를 추가하고 ResponseClass.java라고 이름을 지정합니다. 디렉터리 구조의 HelloPojo.java 클래스 옆에 이를 배치합니다.

```
package example;

public class ResponseClass {
    String greetings;

    public String getGreetings() {
        return greetings;
    }

    public void setGreetings(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass(String greetings) {
        this.greetings = greetings;
    }

    public ResponseClass() {
    }
}
```

#### Note

AWS Lambda에 내장된 JSON 직렬 변환기에서 POJO를 사용하려면 `get` 및 `set` 메서드가 필요합니다. 일반적으로 인수가 없는 생성자는 필요하지 않지만, 이 예제에서는 다른 생성자를 제공하고 있기 때문에 인수가 없는 생성자를 명시적으로 제공해야 합니다.

이 코드를 Lambda 함수로 업로드하여 다음과 같이 테스트할 수 있습니다.

- 위의 코드 파일을 사용하여 배포 패키지를 생성합니다.
- AWS Lambda로 배포 패키지를 업로드하고 Lambda 함수를 생성합니다. 이를 위해 콘솔 또는 AWS CLI를 사용할 수 있습니다.
- 콘솔 또는 CLI를 사용하여 Lambda 함수를 수동으로 호출합니다. 함수를 수동으로 호출할 때 샘플 JSON 이벤트 데이터를 사용할 수 있습니다. 예:

```
{ "firstName": "John", "lastName": "Doe" }
```

자세한 내용은 [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 단원을 참조하십시오. 다음 차이점을 참고합니다.

- 배포 패키지를 생성할 때 aws-lambda-java-core 라이브러리 종속 프로그램을 잊어서는 안 됩니다.
- Lambda 함수를 생성할 때 핸들러 값으로 example.HelloPojo::handleRequest(*package.class::method*)를 지정합니다.

## 예: 핸들러 입력/출력을 위한 스트림 사용(Java)

POJO를 사용하고 싶지 않거나 Lambda의 직렬화 접근 방식이 요구를 충족시키지 않는 경우, 바이트 스트림 구현을 사용할 수 있습니다. 이 경우 핸들러의 입력 및 출력 유형으로 `InputStream` 및 `OutputStream`을 사용할 수 있습니다. 예시 핸들러 함수는 다음과 같습니다.

```
public void handler(InputStream inputStream, OutputStream outputStream, Context context)
    throws IOException{
    ...
}
```

이 경우 핸들러 함수는 요청 스트림과 응답 스트림 모두에 파라미터를 사용합니다.

다음은 `input` 및 `output` 파라미터에 대해 `InputStream` 및 `OutputStream` 유형을 사용하는 핸들러를 구현하는 Lambda 함수 예제입니다.

### Note

입력 페이로드는 유효한 JSON이어야 하지만 출력 스트림에는 이러한 제한이 없습니다. 모든 바이트가 지원됩니다.

```
package example;

import java.io.InputStream;
import java.io.OutputStream;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler{
    public void handler(InputStream inputStream, OutputStream outputStream, Context context) throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

다음을 수행하여 코드를 테스트할 수 있습니다.

- 위의 코드를 사용하여 배포 패키지를 생성합니다.
- AWS Lambda로 배포 패키지를 업로드하고 Lambda 함수를 생성합니다. 이를 위해 콘솔 또는 AWS CLI를 사용할 수 있습니다.
- 샘플 입력을 제공하여 코드를 수동으로 호출할 수 있습니다. 예:

```
test
```

시작하기에서 제공되는 지침을 따릅니다. 자세한 내용은 [Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 단원을 참조하십시오. 다음 차이점을 참고합니다.

- 배포 패키지를 생성할 때 aws-lambda-java-core 라이브러리 종속 프로그램을 잊어서는 안 됩니다.
- Lambda 함수를 생성할 때 핸들러 값으로 example.Hello::handler(*package.class::method*)를 지정합니다.

## 핸들러를 생성하기 위해 사전 정의된 인터페이스 활용 (Java)

AWS Lambda Java 코드 라이브러리(aws-lambda-java-core)에서 제공되는 사전 정의된 인터페이스 중 하나를 사용하여 임의의 이름과 파라미터를 갖는 자체 핸들러 메서드를 작성하는 대신 Lambda 함수 핸들러를 생성할 수 있습니다. 핸들러에 대한 자세한 내용은 [AWS Lambda 함수 핸들러\(Java\) \(p. 265\)](#)을 참조하십시오.

사전 정의된 인터페이스인 RequestStreamHandler 또는 RequestHandler 중 하나를 구현하고 인터페이스가 제공하는 handleRequest 메서드에 구현된 인터페이스를 제공할 수 있습니다. 핸들러 입력/출력에서 표준 Java 유형을 사용하고 싶은지 사용자 지정 POJO 유형을 사용하고 싶은지에 따라 이들 인터페이스 중 하나를 구현하거나(AWS Lambda는 데이터 유형을 일치시키기 위해 입력 및 출력을 자동으로 직렬화 및 역직렬화) Stream 유형을 사용하여 직렬화를 사용자 지정합니다.

### Note

이들 인터페이스는 aws-lambda-java-core 라이브러리에서 사용할 수 있습니다.

표준 인터페이스를 구현하면 컴파일 시 메서드 서명을 검증하는 데 도움이 됩니다.

인터페이스 중 하나를 구현하는 경우 Lambda 함수를 생성할 때 Java 코드에서 *package.class*를 핸들러로 지정합니다. 예를 들어 다음은 시작하기에서 수정된 create-function CLI 명령입니다. --handler 파라미터는 다음과 같이 "example.Hello" 값을 지정합니다.

```
aws lambda create-function \
--region region \
--function-name getting-started-lambda-function-in-java \
--zip-file fileb://deployment-package (zip or jar)
  path \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler example.Hello \
--runtime java8 \
--timeout 15 \
--memory-size 512
```

다음 단원에서는 이들 인터페이스를 구현하는 방법에 대한 예제를 제공합니다.

## 예제 1: 사용자 지정 POJO 입력/출력을 통해 핸들러 생성 (RequestHandler 인터페이스 활용)

이 단원에 예제로 제공되는 Hello 클래스는 RequestHandler 인터페이스를 구현합니다. 인터페이스는 이벤트 데이터에서 Request 유형의 입력 파라미터로 취급되는 handleRequest() 메서드를 정의하고 Response 유형의 POJO 객체를 반환합니다.

```
public Response handleRequest(Request request, Context context) {
    ...
}
```

handleRequest() 메서드의 샘플이 구현된 Hello 클래스가 표시됩니다. 이 예제에서 이벤트 데이터는 이 틀과 성으로 이루어져 있다고 가정합니다.

```
package example;

import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestHandler<Request, Response> {

    public Response handleRequest(Request request, Context context) {
        String greetingString = String.format("Hello %s %s.", request.firstName,
request.lastName);
        return new Response(greetingString);
    }
}
```

예를 들어 Request 객체의 이벤트 데이터가 다음과 같은 경우입니다.

```
{
    "firstName": "value1",
    "lastName" : "value2"
}
```

메서드는 다음과 같이 Response 객체를 반환합니다.

```
{
    "greetings": "Hello value1 value2."
}
```

다음으로 Request 및 Response 클래스를 구현해야 합니다. 다음과 같이 구현된 클래스는 테스트하는 데 사용할 수 있습니다.

요청 클래스:

```
package example;

public class Request {
    String firstName;
    String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Request(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Request() {
    }
}
```

```
}
```

응답 클래스:

```
package example;

public class Response {
    String greetings;

    public String getGreetings() {
        return greetings;
    }

    public void setGreetings(String greetings) {
        this.greetings = greetings;
    }

    public Response(String greetings) {
        this.greetings = greetings;
    }

    public Response() {
    }
}
```

이 코드에서 Lambda 함수를 생성하고 다음과 같이 전체 경험을 테스트할 수 있습니다.

- 위의 코드를 사용하여 배포 패키지를 생성합니다. 자세한 내용은 [AWS Lambda 배포 패키지 \(Java\) \(p. 258\)](#) 단원을 참조하십시오.
- AWS Lambda로 배포 패키지를 업로드하고 Lambda 함수를 생성합니다.
- 콘솔이나 CLI를 사용하여 Lambda 함수를 테스트합니다. 예를 들어 Request 클래스의 getter 및 setter에 부합하는 샘플 JSON 데이터를 지정할 수 있습니다.

```
{
    "firstName": "John",
    "lastName" : "Doe"
}
```

Lambda 함수는 응답으로 다음의 JSON을 반환합니다.

```
{
    "greetings": "Hello John, Doe."
}
```

시작하기에서 제공되는 지침을 따릅니다([Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 참조). 다음 차이점을 참고합니다.

- 배포 패키지를 생성할 때 aws-lambda-java-core 라이브러리 종속 프로그램을 잊어서는 안 됩니다.
- Lambda 함수를 생성할 때 핸들러 값으로 example.Hello([package.class](#))를 지정합니다.

## 예제 2: 스트림 POJO 입력/출력을 통해 핸들러 생성 (RequestStreamHandler 인터페이스 활용)

이 예제의 Hello 클래스는 RequestStreamHandler 인터페이스를 구현합니다. 인터페이스는 다음과 같이 handleRequest 메서드를 정의합니다.

```
public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
    throws IOException {
    ...
}
```

`handleRequest()` 핸들러의 샘플이 구현된 `Hello` 클래스가 표시됩니다. 핸들러는 수신 이벤트 데이터(예: 문자열 "hello")를 간단하게 대문자로 변환하는 방법으로 처리하여 반환합니다.

```
package example;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler {
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
        throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

이 코드에서 Lambda 함수를 생성하고 다음과 같이 전체 경험을 테스트할 수 있습니다.

- 위의 코드를 사용하여 배포 패키지를 생성합니다.
- AWS Lambda로 배포 패키지를 업로드하고 Lambda 함수를 생성합니다.
- 콘솔이나 CLI를 사용하여 Lambda 함수를 테스트합니다. 예를 들어 어떤 샘플 문자열 데이터도 지정할 수 있습니다.

```
"test"
```

Lambda 함수는 응답에서 TEST를 반환합니다.

시작하기에서 제공되는 지침을 따릅니다([Java로 작성된 Lambda 함수 생성 \(p. 281\)](#) 참조). 다음 차이점을 참고합니다.

- 배포 패키지를 생성할 때 `aws-lambda-java-core` 라이브러리 종속 프로그램을 잊어서는 안 됩니다.
- Lambda 함수를 생성할 때 핸들러 값으로 `example.Hello`(*package.class*)를 지정합니다.

## AWS Lambda 컨텍스트 객체(Java)

Lambda은 함수를 실행할 때 컨텍스트 객체를 [핸들러 \(p. 265\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 제공하는 메서드 및 속성들을 제공합니다.

### 컨텍스트 메서드

- `getRemainingTimeInMillis()` – 실행 시간이 초과되기 전에 남은 시간(밀리초)을 반환합니다.

- `getFunctionName()` – Lambda 함수의 이름을 반환합니다.
- `getFunctionVersion()` – 함수의 버전 (p. 45)을 반환합니다.
- `getInvokedFunctionArn()` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN)을 반환합니다. 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `getMemoryLimitInMB()` – 함수에 구성된 메모리 양을 반환합니다.
- `getAwsRequestId()` – 호출 요청의 식별자를 반환합니다.
- `getLogGroupName()` – 함수에 대한 로그 그룹을 반환합니다.
- `getLogStreamName()` – 함수 인스턴스에 대한 로그 스트림을 반환합니다.
- `getIdentity()` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보를 반환합니다.
- `getClientContext()` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트를 반환합니다.
- `getLogger()` – 함수에 대한 로거 객체 (p. 275)를 반환합니다.

## AWS Lambda 함수 로깅(Java)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드의 로그를 출력하려면 `java.lang.System`에서 메서드를 사용하거나, `stdout` 또는 `stderr`에 쓰는 토깅 모듈을 사용합니다. 다음 예에는 `System.out.println`가 사용됩니다.

Example Hello.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(String name, Context context) {
        System.out.println("Event received.");
        return String.format("Hello %s.", name);
    }
}
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스](#) (p. 101)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션](#) (p. 116) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 `--log-type` 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 `LogResult` 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOjA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms      Billed
Duration: 100 ms      Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## LambdaLogger

Lambda는 컨텍스트 객체에서 검색할 수 있는 로거 객체를 제공합니다. LambdaLogger는 여러 줄의 로그를 지원합니다. System.out.println을 이용하는 줄 바꿈이 있는 문자열을 로깅하는 경우, 각 줄 바꿈은 CloudWatch Logs의 개별 항목으로 나타납니다. LambdaLogger를 사용한다면 여러 줄이 있는 항목 하나가 나타납니다.

다음 예제 함수는 컨텍스트 정보를 로깅합니다.

### Example ContextLogger.java

```
package example;
import java.io.InputStream;
import java.io.OutputStream;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class ContextLogger {
    public static void myHandler(InputStream inputStream, OutputStream outputStream,
        Context context) {
        LambdaLogger logger = context.getLogger();
        int letter;
        try {
            while((letter = inputStream.read()) != -1)
            {
                outputStream.write(Character.toUpperCase(letter));
            }
            Thread.sleep(3000); // Intentional delay for testing the
        getRemainingTimeInMillis() result.
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        logger.log("Log data from LambdaLogger \n with multiple lines");
        // Print info from the context object
    }
}
```

```
        logger.log("Function name: " + context.getFunctionName());
        logger.log("Max mem allocated: " + context.getMemoryLimitInMB());
        logger.log("Time remaining in milliseconds: " +
context.getRemainingTimeInMillis());

        // Return the log stream name so you can look up the log later.
        return String.format("Hello %s. log stream = %s", name,
context.getLogStreamName());
    }
}
```

### 종속성

- aws-lambda-java-core

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## Log4j 2용 사용자 지정 Appender

AWS Lambda에서는 Log4j 2가 사용자 지정 appender를 제공하는 것이 좋습니다. 함수에서의 로깅을 위해 Lambda가 제공하는 사용자 지정 [Apache log4j](#) appender를 사용할 수 있습니다. 사용자 지정 appender는 LambdaAppender라고 하며, log4j2.xml 파일에서 사용해야 합니다. 배포 패키지에 aws-lambda-java-log4j2 아티팩트(artifactId:aws-lambda-java-log4j2)를 넣어야 합니다.

### Example Hello.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Hello {
    // Initialize the Log4j logger.
    static final Logger logger = LogManager.getLogger(Hello.class);

    public String myHandler(String name, Context context) {
        logger.error("log data from log4j err.");

        // Return will include the log stream name so you can look
        // up the log later.
        return String.format("Hello %s. log stream = %s", name,
context.getLogStreamName());
    }
}
```

위의 예제에서는 다음 log4j2.xml 파일을 사용하여 속성을 로드합니다.

### Example log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration packages="com.amazonaws.services.lambda.runtime.log4j2">
<Appenders>
    <Lambda name="Lambda">
        <PatternLayout>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1}:%L - %m%n</pattern>
        </PatternLayout>
    </Lambda>
</Appenders>
```

```
</Appenders>
<Loggers>
    <Root level="info">
        <AppenderRef ref="Lambda" />
    </Root>
</Loggers>
</Configuration>
```

### 종속성

- aws-lambda-java-log4j2
- log4j-core
- log4j-api

Lambda 라이브러리 종속성으로 코드를 빌드하여 배포 패키지를 만듭니다. 자침은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

## AWS Lambda 함수 오류(Java)

Lambda 함수가 예외를 발생시키면 AWS Lambda는 실패를 인식하고 예외 정보를 JSON으로 직렬화하여 반환합니다. 다음은 오류 메시지의 예제입니다.

```
{
    "errorMessage": "Name John Doe is invalid. Exception occurred...",
    "errorType": "java.lang.Exception",
    "stackTrace": [
        "example.Hello.handler(Hello.java:9)",
        "sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)",
        "sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)",

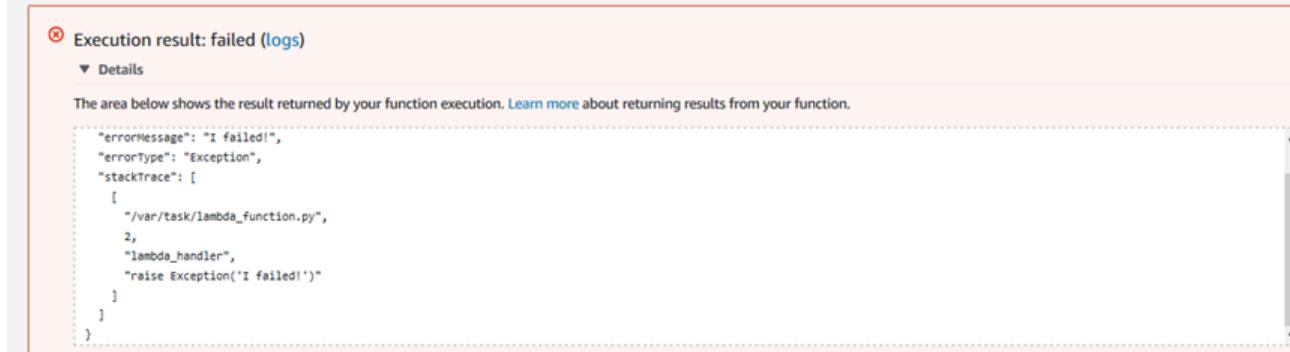
        "sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)",
        "java.lang.reflect.Method.invoke(Method.java:497)"
    ]
}
```

스택 추적은 스택 추적 요소의 stackTrace JSON 배열로 반환됩니다.

오류 정보를 다시 가져오는 방법은 함수를 호출할 때 사용자가 지정한 호출 유형에 따라 다릅니다.

- RequestResponse 호출 유형(즉, 동기식 실행): 이 경우에는 오류 메시지를 다시 수신합니다.

예를 들어 Lambda 콘솔을 사용하여 Lambda 함수를 호출한 경우에는 RequestResponse가 항상 호출 유형이 되고, 아래 그림에서와 같이 콘솔의 실행 결과 섹션에 AWS Lambda가 반환한 오류 정보가 표시됩니다.



- **Event 호출 유형(즉, 비동기식 실행)**: 이 경우 AWS Lambda가 어떤 값도 반환하지 않습니다. 대신에 CloudWatch Logs 및 CloudWatch 측정치의 오류 정보를 로깅합니다.

이벤트 소스에 따라 AWS Lambda는 실패한 Lambda 함수를 다시 시도할 수 있습니다. 예를 들어 Kinesis가 Lambda 함수에 대한 이벤트 소스인 경우, AWS Lambda는 Lambda 함수가 성공하거나 스트림의 레코드가 만료될 때까지 실패한 호출을 재시도합니다.

## 함수 오류 처리

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 AWS Step Functions 상태 시스템 내에서 직접 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

`CreateAccount` 상태가 Lambda 함수를 사용하여 고객의 세부 정보를 데이터베이스에 기록하는 작업인 경우를 생각해 보십시오.

- 작업이 성공하면 계정이 만들어지고 환영 이메일이 전송됩니다.
- 사용자가 이미 존재하는 사용자 이름에 대해 계정을 만들려고 시도하면 Lambda 함수는 오류를 발생시켜 상태 시스템이 다른 사용자 이름을 제안하고 계정 생성 프로세스를 재시도하도록 만들습니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다. Java의 사용자 지정 오류는 `Exception` 클래스를 확장해야 합니다.

```
package com.example;

public static class AccountAlreadyExistsException extends Exception {
    public AccountAlreadyExistsException(String message) {
        super(message);
    }
}

package com.example;

import com.amazonaws.services.lambda.runtime.Context;

public class Handler {
    public static void CreateAccount(String name, Context context) throws
    AccountAlreadyExistsException {
        throw new AccountAlreadyExistsException ("Account is in use!");
    }
}
```

`Catch` 규칙을 사용하여 오류를 파악하도록 Step Functions를 구성할 수 있습니다. Lambda는 런타임에 오류 이름을 예외의 정규화된 클래스 이름으로 자동 설정합니다.

```
{
    "StartAt": "CreateAccount",
    "States": {
        "CreateAccount": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
            "Next": "SendWelcomeEmail",
            "Catch": [
                {
                    "ErrorEquals": [ "com.example.AccountAlreadyExistsException" ],
                    "Next": "SuggestAccountName"
                }
            ]
        },
    }
},
```

```
    } ...  
}
```

런타임에 AWS Step Functions는 오류를 파악하여 Next 전환에 지정된 대로 SuggestAccountName 상태로 전환을 합니다.

사용자 지정 오류 처리는 [서버리스 애플리케이션](#)을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda [프로그래밍 모델](#) (p. 31)에서 지원되는 모든 언어와 통합이 되기 때문에 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서버리스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

## AWS Lambda에서 Java 코드 계측

Java의 경우, Lambda가 X-Ray에 하위 세그먼트를 방출하여 함수에서 이루어진 다른 AWS 서비스에 대한 다운스트림 호출 관련 정보를 표시합니다. 이 기능을 활용하려면 배포 패키지에 [Java용 AWS X-Ray SDK](#)를 포함시킵니다. 코드 변경은 필요하지 않습니다. AWS SDK 1.11.48 또는 이후 버전을 사용하면 추적할 함수의 다운스트림 호출에 대해 코드 줄을 추가할 필요가 없습니다.

AWS SDK는 함수에서 이루어진 다운스트림 호출에 대한 하위 세그먼트를 방출하기 위해 X-Ray SDK를 동적으로 가져옵니다. Java용 X-Ray SDK를 사용하면 사용자 지정 하위 세그먼트를 방출하거나 X-Ray 세그먼트에 주석을 추가하기 위해 코드를 계측할 수 있습니다.

다음 예제에서는 Java용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트를 방출하고 X-Ray에 사용자 지정 주석을 추가하기 위해 Lambda 함수를 계측합니다.

```
package uptime;

import java.io.IOException;
import java.time.Instant;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;

public class Hello {
    private static final Log logger = LogFactory.getLog(Hello.class);

    private static final AmazonDynamoDB dynamoClient;
    private static final HttpClient httpClient;

    static {
        dynamoClient =
AmazonDynamoDBClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
        httpClient = HttpClientBuilder.create().build();
    }
}
```

```

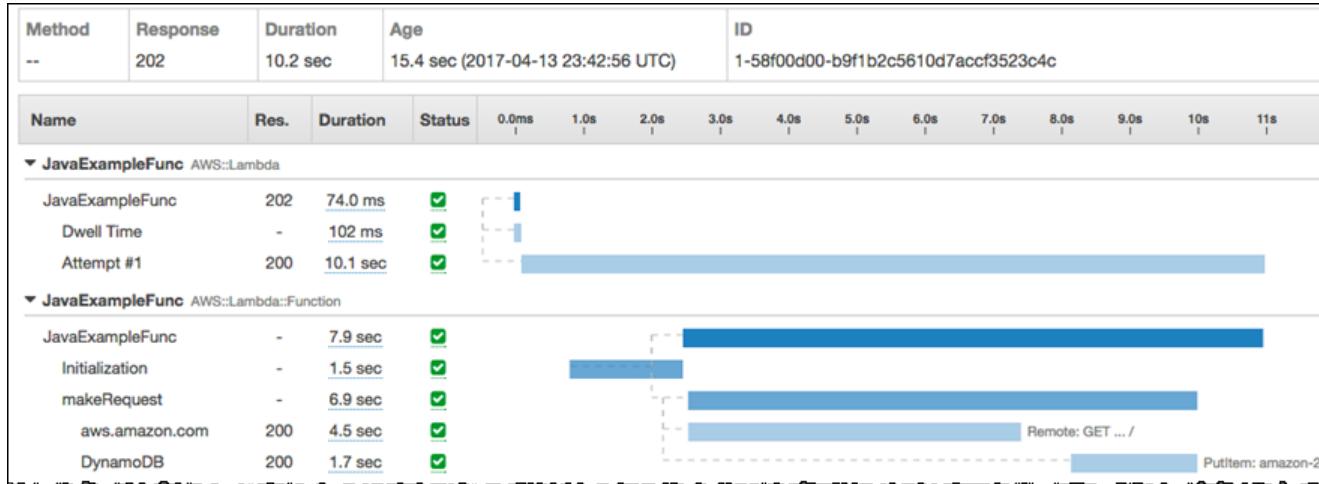
public void checkUptime(Context context) {
    AWSXRay.createSubsegment("makeRequest", (subsegment) -> {

        HttpGet request = new HttpGet("https://aws.amazon.com/");
        boolean is2xx = false;

        try {
            HttpResponse response = httpClient.execute(request);
            is2xx = (response.getStatusLine().getStatusCode() / 100) == 2;
            subsegment.putAnnotation("statusCode",
            response.getStatusLine().getStatusCode());
        } catch (IOException ioe) {
            logger.error(ioe);
        }
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Timestamp", new AttributeValue().withN("" +
        Instant.now().getEpochSecond()));
        item.put("2xx", new AttributeValue().withBOOL(is2xx));
        dynamoClient.putItem("amazon-2xx", item);
    });
}
}

```

다음은 이전 코드에서 방출된 트레이스(동기식 호출)를 나타낸 것입니다.



## Java로 작성된 Lambda 함수 생성

블루프린트는 Python 또는 Node.js로 작성된 샘플 코드를 제공합니다. 콘솔의 인라인 에디터를 사용하여 예제를 손쉽게 수정할 수 있습니다. 그러나 Java로 Lambda 함수의 코드를 작성하고 싶은 경우에는 블루프린트가 제공되지 않습니다. 또한 콘솔에서 Java 코드를 작성할 수 있는 인라인 에디터가 없습니다.

즉, Java 코드를 작성하고 콘솔 밖에서 배포 패키지를 생성해야 한다는 뜻입니다. 배포 패키지를 생성하고 나면 콘솔을 사용하여 Lambda 함수를 생성하기 위해 AWS Lambda에 패키지를 업로드할 수 있습니다. 또한 콘솔을 사용하여 수동 호출을 통해 함수를 테스트할 수도 있습니다.

이 단원에서는 다음과 같은 Java 코드 예제를 사용하여 Lambda 함수를 생성합니다.

```

package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

```

```
public class Hello {  
    public String myHandler(int myCount, Context context) {  
        LambdaLogger logger = context.getLogger();  
        logger.log("received : " + myCount);  
        return String.valueOf(myCount);  
    }  
}
```

프로그래밍 모델은 Java 코드의 작성 방법을 자세하게 설명합니다(예를 들어 AWS Lambda가 지원하는 입력/출력 유형). 프로그래밍 모델에 대한 자세한 내용은 [Java를 사용하여 Lambda 함수 빌드 \(p. 257\)](#)을 참조하십시오. 이 코드에 대해 다음에 유의하십시오.

- 이 코드를 패키징 및 업로드하여 Lambda 함수를 생성할 때, `example.Hello::myHandler` 메서드 참조를 핸들러로 지정합니다.
- 이 예제의 핸들러는 입력에는 `int` 유형을, 출력에는 `String` 유형을 사용합니다.

AWS Lambda는 JSON 직렬화가 가능한 유형과 `InputStream/OutputStream` 유형의 입력/출력을 지원합니다. 이 함수를 호출할 때 샘플 `int`(예를 들어 `123`)를 통과합니다.

- 하지만 Lambda 콘솔을 사용하여 이 Lambda 함수를 수동으로 호출할 수 있습니다. 콘솔은 항상 `RequestResponse` 호출 유형(동기식)을 사용하기 때문에 콘솔에서 응답을 볼 수 있습니다.
- 핸들러에는 `Context` 파라미터 옵션이 포함됩니다. 코드에서 `Context` 객체에서 제공된 `LambdaLogger`를 사용하여 CloudWatch 로그에 로그 항목을 기록합니다. `Context` 객체 사용에 대한 자세한 내용은 [AWS Lambda 컨텍스트 객체\(Java\) \(p. 274\)](#)을 참조하십시오.

먼저, 배포 패키지에 이 코드와 종속 프로그램을 패키징해야 합니다. 그런 다음, 시작하기 연습을 사용하여 Lambda 함수를 생성하고 콘솔을 사용하여 테스트를 할 수 있도록 패키지를 업로드할 수 있습니다. 배포 패키지 만들기에 대한 자세한 내용은 [AWS Lambda 배포 패키지\(Java\) \(p. 258\)](#) 단원을 참조하십시오.

# Go를 사용하여 Lambda 함수 빌드

다음 단원에서는 일반적인 프로그래밍 패턴 및 핵심 개념이 Go에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다.

## Go 런타임

이름	식별자	운영 체제
Go 1.x	go1.x	Amazon Linux

### 주제

- [AWS Lambda 배포 패키지\(Go\) \(p. 283\)](#)
- [AWS Lambda 함수 핸들러\(Go\) \(p. 284\)](#)
- [AWS Lambda 콘텍스트 객체\(Go\) \(p. 288\)](#)
- [AWS Lambda 함수 로깅\(Go\) \(p. 289\)](#)
- [AWS Lambda 함수 오류\(Go\) \(p. 291\)](#)
- [AWS Lambda에서 Go 코드 계측 \(p. 293\)](#)
- [환경 변수 사용 \(p. 294\)](#)

뿐만 아니라 AWS Lambda는 다음을 제공합니다.

- [github.com/aws/aws-lambda-go/lambda](#): Go용 Lambda 프로그래밍 모델을 구현한 것입니다. 이 패키지는 AWS Lambda에서 [AWS Lambda 함수 핸들러\(Go\) \(p. 284\)](#)를 호출하기 위해 사용됩니다.
- [github.com/aws/aws-lambda-go/lambdacontext](#): [AWS Lambda 콘텍스트 객체\(Go\) \(p. 288\)](#)에서 실행 콘텍스트 정보를 액세스하기 위한 헬퍼.
- [github.com/aws/aws-lambda-go/events](#): 이 라이브러리는 일반적인 이벤트 소스 통합을 위한 유형 정의를 제공합니다.

## AWS Lambda 배포 패키지(Go)

Lambda 함수를 생성하려면 먼저, 코드와 종속 프로그램으로 구성된 zip 파일인 Lambda 함수 배포 패키지를 만듭니다.

배포 패키지를 생성한 후에는 이를 직접 업로드하거나 Lambda 함수를 생성하고 싶은 동일한 AWS 리전의 Amazon S3 버킷에 .zip 파일을 먼저 업로드한 다음, 콘솔이나 AWS CLI를 사용하여 Lambda 함수를 생성할 때 버킷 이름과 객체 키 이름을 지정할 수 있습니다.

Go에 작성된 Lambda 함수의 경우, Go 런타임 디렉터리로 이동하여 Go용 Lambda 라이브러리를 다운로드 한 후 `go get github.com/aws/aws-lambda-go/lambda` 명령을 입력합니다.

다음 명령을 사용하여 CLI를 통해 Go Lambda 함수를 빌드하고 패키징 및 배포합니다. `function-name` 은 `Lambda handler`의 이름과 일치해야 합니다.

```
GOOS=linux go build lambda_handler.go
```

```
zip handler.zip ./lambda_handler
# --handler is the path to the executable inside the .zip
aws lambda create-function \
--region region \
--function-name lambda-handler \
--memory 128 \
--role arn:aws:iam::account-id:role/execution_role \
--runtime go1.x \
--zip-file fileb://path-to-your-zip-file/handler.zip \
--handler lambda-handler
```

#### Note

Windows 또는 macOS 같은 비 Linux 환경을 사용하는 경우, 핸들러 함수 코드를 컴파일할 때 GOOS (Go Operating System) 환경 변수를 'linux'로 설정하여 핸들러 함수가 Lambda 실행 콘텍스트와 호환되는지 확인합니다.

## Windows에서 배포 패키지 만들기

Windows를 사용해 AWS Lambda에서 실행할 .zip 파일을 생성할 경우, build-lambda-zip 도구를 설치할 것을 권장합니다.

#### Note

설치를 아직 수행하지 않았다면 git를 설치한 후 git 실행 파일을 Windows %PATH% 환경 변수에 추가해야 합니다.

이 도구를 다운로드하려면 다음 명령을 실행합니다.

```
go.exe get -u github.com/aws/aws-lambda-go/cmd/build-lambda-zip
```

GOPATH에서 이 도구를 사용합니다. Go가 기본적으로 설치되어 있다면 도구는 보통 %USERPROFILE%\Go\bin에 위치하게 됩니다. 그렇지 않다면 Go 런타임을 설치한 위치로 이동해 다음을 수행합니다.

cmd.exe 파일에서 다음을 실행합니다.

```
set GOOS=linux
go build -o main main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o main.zip main
```

Powershell에서 다음을 실행합니다.

```
$env:GOOS = "linux"
go build -o main main.go
~\Go\Bin\build-lambda-zip.exe -o main.zip main
```

## AWS Lambda 함수 핸들러(Go)

Go에 작성된 Lambda 함수는 Go 실행 파일로 작성됩니다. Lambda 함수 코드에서는 [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/lambda) 패키지 [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/lambda)를 포함해야 하는데, 이 패키지는 Go용 Lambda 프로그래밍 모델을 구현합니다. 또한 핸들러 함수 코드와 main() 함수를 구현해야 합니다.

```
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)
}

```

다음을 참조하십시오.

- package main: Go에서 `func main()`을 포함하는 패키지의 이름은 항상 `main`으로 지정해야 합니다.
- import: 이 명령을 사용하면 Lambda 함수에 필요한 라이브러리를 포함시킬 수 있습니다. 이 인스턴스에서는 다음을 포함합니다.
  - context: [AWS Lambda 콘텍스트 객체\(Go\) \(p. 288\)](#).
  - fmt: 함수의 반환 값 형식을 지정하는 데 사용되는 Go [서식](#) 객체입니다.
  - github.com/aws/aws-lambda-go/lambda:: 앞서 언급한 것처럼 Go용 Lambda 프로그래밍 모델을 구현합니다.
- func HandleRequest(ctx context.Context, name MyEvent) (string, error): 이것은 Lambda 핸들러 서명이며 실행할 해당 코드를 포함합니다. 또한 포함된 파라미터들은 다음을 나타냅니다.
  - ctx context.Context: Lambda 함수 호출을 위한 런타임 정보를 제공합니다. `ctx`는 [AWS Lambda 콘텍스트 객체\(Go\) \(p. 288\)](#)를 통해 사용 가능한 정보를 활용하기 위해 선언하는 변수입니다.
  - name MyEvent: `return` 명령문으로 값을 반환할 `name`의 변수 이름이 포함된 입력 유형입니다.
  - string, error: 표준 [오류](#) 정보를 반환합니다. 사용자 지정 오류 처리에 대한 자세한 내용은 [AWS Lambda 함수 오류\(Go\) \(p. 291\)](#) 단원을 참조하십시오.
  - return fmt.Sprintf("Hello %s!", name), nil: 핸들러 서명에 제공된 이름과 함께 형식이 지정된 "Hello" 인사를 반환합니다. `nil`은 오류가 없었으며 해당 함수가 성공적으로 실행되었음을 나타냅니다.
- func main(): Lambda 함수 코드를 실행하는 진입점입니다. 이 항목은 필수입니다.

`func main(){}` 코드 대괄호 사이에 `lambda.Start(HandleRequest)`를 추가하면 Lambda 함수가 실행됩니다.

#### Note

Go 언어 표준에 따라 여는 대괄호 `{`는 `main` 함수 서명의 맨 끝에 직접 위치해야 합니다.

## 구조화된 유형을 이용한 Lambda 함수 핸들러

위 예제에서 입력 유형은 단순한 문자열이었습니다. 다만 구조화된 이벤트에서 다음과 같이 함수 핸들러로 전달할 수도 있습니다.

```

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)  
}
```

요청은 다음과 같은 형식으로 나타납니다.

```
# request  
{  
    "What is your name?": "Jim",  
    "How old are you?": 33  
}
```

응답은 다음과 같습니다.

```
# response  
{  
    "Answer": "Jim is 33 years old!"  
}
```

AWS 이벤트 소스의 이벤트를 처리하는 것에 대한 자세한 내용은 [aws-lambda-go/events](#)를 참조하십시오.

## 유효한 핸들러 서명

Go에서 Lambda 함수 핸들러를 빌드할 때 몇 가지 옵션들이 있으며 다만 다음과 같은 규칙들을 반드시 준수해야 합니다.

- 핸들러는 하나의 함수여야 합니다.
- 핸들러는 0~2개의 인수를 취할 수 있습니다. 인수가 2개일 경우, 첫 번째 인수는 `context.Context`를 구현해야 합니다.
- 핸들러는 0~2개의 인수를 반환할 수 있습니다. 반환 값이 1개일 경우, 이 값은 `error`를 구현해야 합니다. 반환 값이 2개일 경우, 두 번째 값은 `error`를 구현해야 합니다. 오류 처리 정보 구현에 관한 자세한 내용은 [AWS Lambda 함수 오류\(Go\) \(p. 291\)](#)를 참조하십시오.

다음 목록은 유효한 핸들러 서명을 열거합니다. `TIn` 및 `TOut`은 `encoding/json` 표준 라이브러리와 호환 가능한 유형을 나타냅니다. 이러한 유형들이 역직렬화되는 방식에 관한 자세한 내용은 [func Unmarshal](#)을 참조하시기 바랍니다.

- `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)

## 글로벌 상태 사용

Lambda 함수의 핸들러 코드와 무관한 전역 변수들을 선언하고 수정해야 합니다. 또한 핸들러는 그것이 로드될 때 실행되는 `init` 함수를 선언할 수 있습니다. 이 함수는 표준 Go 프로그램과 마찬가지로에서도 동일하게 동작합니다. 함수의 단일 인스턴스는 여러 이벤트를 동시에 처리하지 못합니다. 예를 들면, 전역 상태는 안전하게 변경할 수 있으며 그러한 변경 사항은 새로운 실행 콘텍스트가 필요한데 이전 실행 콘텍스트로 지정된 함수 호출로부터 잠금 또는 불안정한 동작이 발생하지 않으므로 안심할 수 있습니다. 자세한 내용은 다음 단원을 참조하세요.

- AWS Lambda 실행 콘텍스트 (p. 101)
- AWS Lambda 함수 작업의 모범 사례 (p. 124)

```
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)
```

}

## AWS Lambda 콘텍스트 객체(Go)

Lambda은 함수를 실행할 때 콘텍스트 객체를 [핸들러 \(p. 284\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 메서드 및 속성에 제공합니다.

Lambda 콘텍스트 라이브러리는 다음과 같은 전역 변수, 메서드 및 속성들을 제공합니다.

### 전역 변수

- `FunctionName` – Lambda 함수의 이름
- `FunctionVersion` – 함수의 [버전 \(p. 45\)](#)
- `MemoryLimitInMB` – 함수에 구성된 메모리 양
- `LogGroupName` – 함수에 대한 로그 그룹
- `LogStreamName` – 함수 인스턴스에 대한 로그 스트림

### 콘텍스트 메서드

- `Deadline` – 실행 시간이 초과된 날짜를 Unix 시간 형식에 따른 밀리초 단위로 반환합니다.

### 콘텍스트 속성

- `InvokedFunctionArn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `AwsRequestId` – 호출 요청의 식별자
- `Identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
- `ClientContext` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트

## 호출 콘텍스트 정보 액세스

Lambda 함수들은 그것의 환경과 호출 요청에 관한 메타데이터에 액세스할 수 있습니다. 이는 [Package context](#)에서 액세스할 수 있습니다. 핸들러가 `context.Context`를 하나의 파라미터로 포함할 경우, Lambda는 함수에 관한 정보를 콘텍스트의 `Value` 속성에 삽입합니다. `context.Context` 객체의 콘텐츠에 액세스하려면 `lambdacontext` 라이브러리를 가져와야 합니다.

```
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.Println(lc.Identity.CognitoPoolID)
}

func main() {
    lambda.Start(CognitoHandler)
}
```

```
}
```

위 예제에서 `lc`는 콘텍스트 객체가 수집한 해당 정보를 소비하는 데 사용된 변수이며, `log.Println(lc.Identity.CognitoPoolID)`는 그 정보(이 경우에는 CognitoPoolID)를 출력합니다.

## 함수의 실행 시간 모니터링

다음 예제는 Lambda 함수를 실행하기 위해 소요되는 시간을 모니터링할 수 있도록 콘텍스트 객체를 사용하는 방법을 소개합니다. 이 방법을 활용하면 기대 성능을 분석할 수 있으며 필요에 따라 함수 코드를 조정할 수 있습니다.

```
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.Println("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
    lambda.Start(LongRunningHandler)
}
```

## AWS Lambda 함수 로깅(Go)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드의 로그를 출력하려면, [fmt 패키지](#)에서 메서드를 사용하거나, `stdout` 또는 `stderr`에 쓰는 로깅 라이브러리를 사용합니다. 다음 예제는 `fmt.Println`이 사용됩니다.

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)
```

```
func HandleRequest() {
    fmt.Println("Hello from Lambda")
}

func main() {
    lambda.Start(HandleRequest)
}
```

자세한 로그를 확인하려면 [로그 패키지](#)를 사용합니다.

```
package main

import (
    "log"
    "github.com/aws/aws-lambda-go/lambda"
)

func HandleRequest() {
    log.Println("Event received.")
}

func main() {
    lambda.Start(HandleRequest)
}
```

log의 출력에는 타임스탬프와 요청 ID가 포함됩니다.

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

### Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스 \(p. 101\)](#)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 --log-type 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 LogResult 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
```

```
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(Go)

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 처리할 수 있습니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다. Go의 사용자 지정 오류는 errors 모듈을 가져와야 합니다.

```
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)
}
```

다음을 반환합니다.

```
{
    "errorMessage": "something went wrong!",
    "errorType": "errorString"
}
```

## 함수 오류 처리

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 AWS Step Functions 상태 시스템 내에서 직접 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

CreateAccount 상태가 Lambda 함수를 사용하여 고객의 세부 정보를 데이터베이스에 기록하는 작업인 경우를 생각해 보십시오.

- 작업이 성공하면 계정이 만들어지고 환영 이메일이 전송됩니다.
- 사용자가 이미 존재하는 사용자 이름에 대해 계정을 만들려고 시도하면 Lambda 함수는 오류를 발생시켜 상태 시스템이 다른 사용자 이름을 제안하고 계정 생성 프로세스를 재시도하도록 만듭니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다.

```
package main
```

```
type CustomError struct {}

func (e *CustomError) Error() string {
    return "bad stuff happened..."
}

func MyHandler() (string, error) {
    return "", &CustomError{}
}
```

런타임에 AWS Step Functions는 오류를 파악하여 Next 전환에 지정된 대로 SuggestAccountName 상태로 전환을 합니다.

사용자 지정 오류 처리는 [서버리스](#) 애플리케이션을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda [프로그래밍 모델](#) (p. 31)에서 지원되는 모든 언어와 통합이 되기 때문에 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서비스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

## 예기치 않은 오류 처리

Lambda 함수는 네트워크 중단 등 통제할 수 없는 원인으로 인해 실행되지 못할 수 있습니다. 이러한 경우는 예외적 상황에 해당됩니다. Go에서 `panic`은 이러한 문제들을 처리합니다. 코드가 패닉 상태가 되면 Lambda는 해당 오류를 수집하여 이를 표준 오류 json 형식으로 직렬화하는 시도를 합니다. 또한 Lambda는 해당 함수의 CloudWatch 로그에 해당 패닉의 값을 삽입하는 시도도 합니다. 응답을 반환한 후,는 해당 함수를 자동으로 다시 생성합니다. 필요하다면 `panic` 함수를 코드에 포함시켜 오류 응답을 사용자 지정할 수 있습니다.

```
package main

import (
    "errors"

    "github.com/aws/aws-lambda-go/lambda"
)

func handler(string) (string, error) {
    panic(errors.New("Something went wrong"))
}

func main() {
    lambda.Start(handler)
}
```

그러면 json에서 다음과 같은 스택이 반환됩니다.

```
{
    "errorMessage": "Something went wrong",
    "errorType": "errorString",
    "stackTrace": [
        {
            "path": "github.com/aws/aws-lambda-go/lambda/function.go",
            "line": 27,
            "label": "(*Function).Invoke.function"
        },
        ...
    ]
}
```

## AWS Lambda에서 Go 코드 계측

Go용 X-Ray SDK는 Lambda 함수와 함께 사용할 수 있습니다. 핸들러가 AWS Lambda 콘텍스트 객체 (Go) (p. 288)를 첫 번째 인수로 포함할 경우, 해당 객체를 X-Ray SDK로 전달할 수 있습니다. Lambda는 SDK가 하위 세그먼트를 Lambda 호출 서비스 세그먼트에 연결하는 데 사용할 수 있는 이 콘텍스트를 통해 값을 전달합니다. SDK로 만든 하위 세그먼트는 트레이스의 일부로 나타납니다.

### Go용 X-Ray SDK 설치

다음 명령을 사용하여 Go용 X-Ray SDK를 설치합니다 (SDK의 비테스트 종속성이 포함될 것입니다).

```
go get -u github.com/aws/aws-xray-sdk-go/...
```

테스트 종속성을 포함하고 싶다면 다음 명령을 사용합니다.

```
go get -u -t github.com/aws/aws-xray-sdk-go/...
```

Glide를 사용하여 종속성을 관리할 수도 있습니다.

```
glide install
```

### Go용 X-Ray SDK 구성

다음 코드 샘플은 Lambda 함수에서 Go용 X-Ray SDK를 구성하는 방법을 보여줍니다.

```
import (
    "github.com/aws/aws-xray-sdk-go/xray"
)
func myHandlerFunction(ctx context.Context, sample string) {
    xray.Configure(xray.Config{
        LogLevel:      "info",           // default
        ServiceVersion: "1.2.3",
    })
    ... //remaining handler code
}
```

### 하위 세그먼트 생성

다음 코드는 하위 세그먼트를 시작하는 방법을 설명합니다.

```
// Start a subsegment
ctx, subSeg := xray.BeginSubsegment(ctx, "subsegment-name")
// ...
// Add metadata or annotation here if necessary
// ...
subSeg.Close(nil)
```

### Capture

다음 코드는 중요 코드 경로를 추적 및 수집하는 방법을 설명합니다.

```
func criticalSection(ctx context.Context) {
```

```
// This example traces a critical code path using a custom subsegment
xray.Capture(ctx, "MyService.criticalSection", func(ctx1 context.Context) error {
    var err error

    section.Lock()
    result := someLockedResource.Go()
    section.Unlock()

    xray.AddMetadata(ctx1, "ResourceResult", result)
}
}
```

## HTTP 요청 추적

아래에서 보는 바와 같이 HTTP 클라이언트를 추적하고 싶다면 `xray.Client()` 메서드를 사용할 수도 있습니다

```
func myFunction (ctx context.Context) ([]byte, error) {
    resp, err := ctxhttp.Get(ctx, xray.Client(nil), "https://aws.amazon.com")
    if err != nil {
        return nil, err
    }
    return ioutil.ReadAll(resp.Body), nil
}
```

## 환경 변수 사용

Go에서 AWS Lambda 환경 변수 (p. 39)에 액세스하려면 `Getenv` 함수를 사용합니다.

다음 단원에서는 그 방법에 대해 설명합니다. 해당 함수는 출력된 결과들의 형식을 지정하기 위한 `fmt` 패키지 를 가져오며, 환경 변수에 액세스할 수 있도록 플랫폼에 독립적인 시스템 인터페이스인 `os` 패키지도 가져옵니다.

```
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"))
}
```

기본적으로 Lambda는 환경 변수(Lambda 함수에서 사용할 수 있는 환경 변수 (p. 100))를 구성합니다.

# C#을 사용하여 Lambda 함수 빌드

다음 단원에서는 [일반적인 프로그래밍 패턴 및 핵심 개념](#)이 C#에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다.

## .NET 런타임

이름	식별자	언어	운영 체제
.NET Core 2.1	dotnetcore2.1	C# PowerShell Core 6.0	Amazon Linux
.NET Core 1.0	dotnetcore1.0	C#	Amazon Linux

### 주제

- [AWS Lambda 배포 패키지\(C#\)](#) (p. 295)
- [AWS Lambda 함수 핸들러\(C#\)](#) (p. 304)
- [AWS Lambda 콘텍스트 객체\(C#\)](#) (p. 308)
- [AWS Lambda 함수 로깅\(C#\)](#) (p. 308)
- [AWS Lambda 함수 오류\(C#\)](#) (p. 310)

뿐만 아니라, AWS Lambda는 다음을 제공합니다.

- Amazon.Lambda.Core – 이 라이브러리는 정적 Lambda 로거, 직렬화 인터페이스 및 콘텍스트 객체를 제공합니다. Context 객체([AWS Lambda 콘텍스트 객체\(C#\)](#) (p. 308))는 Lambda 함수에 대한 런타임 정보를 제공합니다.
- Amazon.Lambda.Serialization.Json – Amazon.Lambda.Core에 직렬화 인터페이스를 구현한 것입니다.
- Amazon.Lambda.Logging.AspNetCore – ASP.NET에서의 로깅을 위해 라이브러리를 제공합니다.
- 몇 가지 AWS 서비스를 위한 이벤트 객체(POCO)로는 다음과 같은 것들이 있습니다.
  - Amazon.Lambda.APIGatewayEvents
  - Amazon.Lambda.CognitoEvents
  - Amazon.Lambda.ConfigEvents
  - Amazon.Lambda.DynamoDBEvents
  - Amazon.Lambda.KinesisEvents
  - Amazon.Lambda.S3Events
  - Amazon.Lambda.SQSEvents
  - Amazon.Lambda.SNSEvents

이들 패키지는 [Nuget 패키지](#)에서 다운로드가 가능합니다.

## AWS Lambda 배포 패키지(C#)

.NET Core Lambda 배포 패키지는 함수의 컴파일된 어셈블리와 그것의 모든 어셈블리 종속성을 함께 수록한 zip 파일입니다. 패키지는 `proj.deps.json` 파일도 포함합니다. 이 패키지는 함수의 모든 종속성과 `proj.runtimeconfig.json` 파일을 .NET Core 런타임으로 전송하며, 해당 파일은 .NET Core 런타임을 구성하는 데 사용됩니다. .NET CLI의 `publish` 명령은 이들 파일이

모두 포함된 하나의 폴더를 생성할 수 있으며 다만 Lambda 프로젝트는 대체로 하나의 클래스 라이브러리에 구성되기 때문에 `proj.runtimeconfig.json`은 기본적으로 포함되지 않습니다.  
`proj.runtimeconfig.json`을 publish 프로세스의 일부로 강제로 작성하려면 해당 명령줄 인수(/p:GenerateRuntimeConfigurationFiles=true to the publish command)를 전달합니다.

#### Note

`dotnet publish` 명령을 사용해 배포 패키지를 생성할 수는 있지만 [AWS Toolkit for Visual Studio \(p. 303\)](#) 또는 [.NET Core CLI \(p. 296\)](#)를 사용해 배포 패키지를 생성하는 것이 좋습니다.  
이들 도구는 `lambda-project.runtimeconfig.json` 파일이 존재하며 비 Linux 기반 종속성 제거를 포함한 패키지 번들을 최적화할 수 있도록 Lambda에 맞게 특별히 최적화되어 있습니다.

## .NET Core CLI

.NET Core CLI는 .NET 기반의 Lambda 애플리케이션을 생성할 수 있도록 크로스 플랫폼 방법을 제공합니다. 이 단원에서는 .NET Core CLI가 설치되었다고 가정합니다. 아직 설치하지 않았다면 [여기](#)에서 설치합니다.

.NET CLI에서는 `new` 명령을 사용해 하나의 명령줄에서 .NET 프로젝트를 생성합니다. 이는 특히 Visual Studio 외부에서 플랫폼에 독립적인 프로젝트를 생성하려고 할 때 유용합니다. 사용 가능한 프로젝트 유형들의 목록을 보려면 하나의 명령줄을 열어 .NET Core 런타임을 설치한 위치로 이동한 다음, 다음을 입력합니다.

```
dotnet new -all
```

다음과 같은 모양이어야 합니다.

```
dotnet new -all
Usage: new [options]

Options:
  -h, --help           Displays help for this command.
  -l, --list            Lists templates containing the specified name. If no name is
                        specified, lists all templates.
  -n, --name            The name for the output being created. If no name is specified, the
                        name of the current directory is used.
  -o, --output          Location to place the generated output.
  -i, --install         Installs a source or a template pack.
  -u, --uninstall       Uninstalls a source or a template pack.
  --nuget-source        Specifies a NuGet source to use during install.
  --type                Filters templates based on available types. Predefined values are
"project", "item" or "other".
  --force               Forces content to be generated even if it would change existing
files.
  --lang, --language    Filters templates based on language and specifies the language of the
template to create.
```

Templates	Short Name	Language	Tags
<hr/>			
Console Application	console	[C#], F#, VB	
Common/Console			
Class library	classlib	[C#], F#, VB	
Common/Library			
Unit Test Project	mstest	[C#], F#, VB	
Test/MSTest			
xUnit Test Project	xunit	[C#], F#, VB	
Test/xUnit			
Razor Page	page	[C#]	Web/
ASP.NET			

MVC ViewImports	viewimports	[C#]	Web/
ASP.NET			
MVC ViewStart	viewstart	[C#]	Web/
ASP.NET			
ASP.NET Core Empty	web	[C#], F#	Web/
Empty			
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/
MVC			
ASP.NET Core Web App	razor	[C#]	Web/
MVC/Razor Pages			
ASP.NET Core with Angular	angular	[C#]	Web/
MVC/SPA			
ASP.NET Core with React.js	react	[C#]	Web/
MVC/SPA			
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/
MVC/SPA			
Razor Class Library	razorclasslib	[C#]	Web/
Razor/Library/Razor Class Library			
ASP.NET Core Web API	webapi	[C#], F#	Web/
WebAPI			
global.json file	globaljson		
Config			
NuGet Config	nugetconfig		
Config			
Web Config	webconfig		
Config			
Solution File	sln		
Solution			
<b>Examples:</b>			
dotnet new mvc --auth Individual			
dotnet new viewstart			
dotnet new --help			

예를 들면, 하나의 콘솔 프로젝트를 생성하려고 할 경우, 다음과 같이 진행하면 됩니다.

1. `mkdir ##` 명령을 사용하여 프로젝트가 생성되는 디렉터리를 만듭니다.
2. `cd ##` 명령을 사용하여 해당 디렉터리로 이동합니다.
3. `dotnet new console -o myproject` 명령을 입력합니다.

이렇게 하면 `##` 디렉터리에 다음과 같은 파일들이 생성됩니다.

- Lambda 함수 코드를 작성하는 파일인 `Program.cs`
- .NET 애플리케이션을 구성하는 해당 파일 및 종속성들을 열거한 XML 파일인 `MyProject.csproj`.

AWS Lambda는 [Amazon.Lambda.Templates](#) nuget 패키지를 통해 템플릿을 추가로 제공합니다. 이 패키지를 설치하려면 다음 명령을 실행합니다.

```
dotnet new -i Amazon.Lambda.Templates
```

설치가 완료되면 Lambda 템플릿은 `dotnet new`의 일부로 표시됩니다. 이를 확인하려면 다음 명령을 다시 실행합니다.

```
dotnet new -all
```

이제 다음과 같은 내용이 나타나야 합니다.

```
dotnet new -all
Usage: new [options]
```

```
Options:
-h, --help           Displays help for this command.
-l, --list            Lists templates containing the specified name. If no name is
specified, lists all templates.
-n, --name             The name for the output being created. If no name is specified, the
name of the current directory is used.
-o, --output            Location to place the generated output.
-i, --install            Installs a source or a template pack.
-u, --uninstall          Uninstalls a source or a template pack.
--nuget-source          Specifies a NuGet source to use during install.
--type                  Filters templates based on available types. Predefined values are
"project", "item" or "other".
--force                 Forces content to be generated even if it would change existing
files.
-lang, --language        Filters templates based on language and specifies the language of the
template to create.
```

Templates	Language	Tags	Short Name
Order Flowers Chatbot Tutorial			lambda.OrderFlowersChatbot
Lambda Detect Image Labels	[C#]	AWS/Lambda/Function	lambda.DetectImageLabels
Lambda Empty Function	[C#, F#]	AWS/Lambda/Function	lambda.EmptyFunction
Lex Book Trip Sample	[C#]	AWS/Lambda/Function	lambda.LexBookTripSample
Lambda Simple DynamoDB Function	[C#, F#]	AWS/Lambda/Function	lambda.DynamoDB
Lambda Simple Kinesis Firehose Function	[C#]	AWS/Lambda/Function	lambda.KinesisFirehose
Lambda Simple Kinesis Function	[C#, F#]	AWS/Lambda/Function	lambda.Kinesis
Lambda Simple S3 Function	[C#, F#]	AWS/Lambda/Function	lambda.S3
Lambda ASP.NET Core Web API	[C#, F#]	AWS/Lambda/Serverless	serverless.AspNetCoreWebAPI
Lambda ASP.NET Core Web Application with Razor Pages	[C#]	AWS/Lambda/Serverless	serverless.AspNetCoreWebApp
Serverless Detect Image Labels	[C#, F#]	AWS/Lambda/Serverless	serverless.DetectImageLabels
Lambda DynamoDB Blog API	[C#]	AWS/Lambda/Serverless	serverless.DynamoDBBlogAPI
Lambda Empty Serverless	[C#, F#]	AWS/Lambda/Serverless	serverless.EmptyServerless
Lambda Giraffe Web App	F#	AWS/Lambda/Serverless	serverless.Giraffe
Serverless Simple S3 Function	[C#, F#]	AWS/Lambda/Serverless	serverless.S3
Step Functions Hello World	serverless.StepFunctionsHelloWorld	[C#, F#]	AWS/Lambda/Serverless
Console Application	[C#, F#, VB]	Common/Console	console
Class library	[C#, F#, VB]	Common/Library	classlib
Unit Test Project	[C#, F#, VB]	Test/MSTest	mstest
xUnit Test Project	[C#, F#, VB]	Test/xUnit	xunit
Razor Page	[C#]	Web/ASP.NET	page
MVC ViewImports	[C#]	Web/ASP.NET	viewimports

MVC ViewStart		viewstart
[C#]	Web/ASP.NET	
ASP.NET Core Empty		web
[C#], F#	Web/Empty	
ASP.NET Core Web App (Model-View-Controller)		mvc
[C#], F#	Web/MVC	
ASP.NET Core Web App		razor
[C#]	Web/MVC/Razor Pages	
ASP.NET Core with Angular		angular
[C#]	Web/MVC/SPA	
ASP.NET Core with React.js		react
[C#]	Web/MVC/SPA	
ASP.NET Core with React.js and Redux		reactredux
[C#]	Web/MVC/SPA	
Razor Class Library		razorclasslib
[C#]	Web/Razor/Library/Razor Class Library	
ASP.NET Core Web API		webapi
[C#], F#	Web/WebAPI	
global.json file		globaljson
	Config	
NuGet Config		nugetconfig
Web Config		webconfig
Solution File		sln
	Solution	
<b>Examples:</b>		
dotnet new mvc --auth Individual		
dotnet new viewimports --namespace		
dotnet new --help		

특정 템플릿에 관한 자세한 내용을 확인하려면 다음 명령을 사용합니다.

```
dotnet new lambda.EmptyFunction --help
```

다음을 참조하십시오.

```
-p|--profile  The AWS credentials profile set in aws-lambda-tools-defaults.json and used
as the default profile when interacting with AWS.
string - Optional

-r|--region   The AWS region set in aws-lambda-tools-defaults.json and used as the default
region when interacting with AWS.
string - Optional
```

이들 값은 Lambda 함수를 만들 때 설정할 수 있는 선택적 값으로서 함수 생성 프로세스의 일부로 빌드된 `aws-lambda-tools-defaults.json` 파일에 자동으로 작성됩니다. 다음 단원에서는 이들 값의 의미에 대해 설명합니다.

- `--profile` – 실행 역할 (p. 9).
- `--region` – 함수가 상주할 리전.

예를 들어 하나의 Lambda 함수를 생성하려면 `--region` 파라미터의 값들을 선택한 리전으로 대체하고 `--profile`을 IAM 프로파일로 대체하면서 다음 명령을 실행합니다.

#### Note

Lambda 함수의 요구 사항에 대한 자세한 내용은 [CreateFunction \(p. 347\)](#) 단원을 참조하십시오.

```
dotnet new lambda.EmptyFunction --name MyFunction --profile default --region region
```

이는 다음과 비슷한 디렉터리 구조를 생성해야 합니다.

```
<dir>myfunction
  /src/myfunction
  /test/myfunction
```

src/myfunction 디렉터리 아래에서 다음과 같은 파일들을 검사합니다.

- aws-lambda-tools-defaults.json: Lambda 함수를 배포할 때 이 파일에 명령줄 옵션을 지정합니다. 예:

```
"profile": "iam profile",
  "region" : "region",
  "configuration" : "Release",
  "framework" : "netcoreapp2.1",
  "function-runtime": "dotnetcore2.1",
  "function-memory-size" : 256,
  "function-timeout" : 30,
  "function-handler" : "MyFunction::MyFunction.Function::FunctionHandler"
```

- Function.cs: Lambda 핸들러 함수 코드입니다. 이 코드는 기본 Amazon.Lambda.Core 라이브러리와 기본 LambdaSerializer 속성을 포함하는 C# 템플릿에 속합니다. 직렬화 요구 사항들과 옵션들에 관한 자세한 내용은 [Lambda 함수 직렬화 \(p. 302\)](#) 단원을 참조하십시오. 이 단원은 함수 코드를 적용하기 위해 편집할 수 있는 하나의 샘플 함수도 포함하고 있습니다.

```
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: 애플리케이션을 구성하는 파일 및 어셈블리들의 목록을 열거한 MSBuild 파일입니다.

```
<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>
```

```
</ItemGroup>  
</Project>
```

- Readme: 이 파일을 사용하면 Lambda 함수를 문서화할 수 있습니다.

myfunction/test directory, examine the following files: 아래에서

- myFunction.Tests.csproj: 앞서 언급한 것처럼 이것은 테스트 프로젝트를 구성하는 여러 파일 및 어셈블리를 열거한 MSBuild 파일입니다. 이 파일은 Amazon.Lambda.Core 라이브러리를 포함하고 있기 때문에 함수를 테스트하는 데 필요한 Lambda 템플릿을 원활하게 통합할 수 있습니다.

```
<Project Sdk="Microsoft.NET.Sdk">  
...  
  
<PackageReference Include="Amazon.Lambda.Core" Version="1.0.0 " />  
...
```

- FunctionTest.cs: src 디렉터리에 포함된 것과 똑같은 C# 코드 템플릿 파일입니다. 함수를 프로덕션 환경에 업로드하기 전에 함수의 프로덕션 코드를 미러링하고 테스트할 수 있도록 이 파일을 편집합니다.

```
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);  
        }  
    }  
}
```

사용 중인 함수가 테스트를 통과하면 Amazon.Lambda.Tools .NET Core Global Tool을 사용해 함수를 빌드하고 배포할 수 있습니다. .NET Core Global Tool을 실행하려면 다음 명령을 실행합니다.

```
dotnet tool install -g Amazon.Lambda.Tools
```

이 도구가 이미 설치되어 있으면 다음 명령을 실행해 최신 버전을 사용 중인지 확인할 수 있습니다.

```
dotnet tool update -g Amazon.Lambda.Tools
```

Amazon.Lambda.Tools .NET Core Global에 대한 자세한 내용은 이 도구의 [GitHub 리포지토리](#)를 참조하십시오.

Amazon.Lambda.Tools를 설치하면 다음 명령을 사용해 함수를 배포할 수 있습니다.

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

배포 후에는 다음 명령을 사용해 하나의 프로덕션 환경에서 이 함수를 다시 테스트할 수 있으며 다른 값의 함수를 Lambda 함수 핸들러로 전달할 수 있습니다.

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
```

모든 작업이 성공했다면 다음과 같은 메시지가 표시되어야 합니다.

```
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
```

## Lambda 함수 직렬화

Stream 객체 이외의 입력 또는 출력 유형을 사용하는 Lambda 함수의 경우, 애플리케이션에 직렬화 라이브러리를 추가해야 합니다. 다음과 같은 방법으로 추가가 가능합니다.

- Json.NET을 사용합니다. Lambda는 JSON.NET을 NuGet 패키지로 사용하여 JSON serializer를 구현합니다.
- Amazon.Lambda.Core 라이브러리의 일부로 사용할 수 있는 ILambdaSerializer 인터페이스를 구현하여 자체 직렬화 라이브러리를 생성합니다. 인터페이스는 두 가지 메서드를 정의합니다.

- T Deserialize<T>(Stream requestStream);

이 메서드를 구현하여 Invoke API에서 Lambda 함수 핸들러로 전달되는 객체로 요청 페이로드를 역직렬화합니다.

- T Serialize<T>(T response, Stream responseStream);

이 메서드를 구현하여 Lambda 함수 핸들러에서 반환된 결과를 Invoke API에서 반환된 응답 페이로드로 직렬화합니다.

이를 종속 프로그램으로 MyProject.csproj 파일에 추가하여 원하는 serializer는 무엇이든 사용할 수 있습니다.

```
...
<ItemGroup>
  <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
  <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
</ItemGroup>
```

그런 다음, AssemblyInfo.cs 파일에 이를 추가합니다. 예를 들어 기본 Json.NET serializer를 사용하고 있는 경우에는 이를 추가했을 것입니다.

```
[assembly:LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
```

#### Note

어셈블리 수준에서 지정된 기본 serializer를 재정의하도록 메서드 수준에서 사용자 지정 직렬화 속성을 정의할 수 있습니다. 자세한 내용은 [표준 데이터 유형 처리 \(p. 305\)](#) 단원을 참조하십시오.

## AWS Toolkit for Visual Studio

AWS Toolkit for Visual Studio에 대한 Lambda 플러그인을 사용하여 .NET 기반 Lambda 애플리케이션을 빌드할 수 있습니다. 플러그인은 [Nuget](#) 패키지의 일부로 사용할 수 있습니다.

### 1단계: 프로젝트 생성 및 빌드

1. Microsoft Visual Studio를 실행하고 [New project]를 선택합니다.
  - a. [File] 메뉴에서 [New]를 선택한 다음, [Project]를 선택합니다.
  - b. [New Project] 창에서 [AWS Lambda Project(.NET Core)]를 선택한 다음 [OK]를 선택합니다.
  - c. [Select Blueprint] 창에 샘플 애플리케이션 목록에서 선택을 할 수 있는 옵션이 나타나서 .NET 기반의 Lambda 애플리케이션 생성을 시작하기 위한 샘플 코드를 제공합니다.
  - d. 처음부터 Lambda 애플리케이션을 생성하려면 빈 함수를 선택한 다음 마침을 생성합니다.
2. 프로젝트의 일부로 생성된 aws-lambda-tools-defaults.json 파일을 검사합니다. 기본적으로 도구가 읽는 옵션들을 이 파일에서 설정할 수 있습니다. Visual Studio에서 생성된 프로젝트 템플릿은 이러한 여러 필드를 기본값으로 설정합니다. 다음 필드를 참조합니다.
  - profile – [실행 역할 \(p. 9\)](#)
  - function-handler – 이 위치에 function handler가 지정되므로 마법사에서 설정할 필요가 없습니다. 그러나 사용 중인 함수 코드에서 #####, #####, ## 또는 ##의 이름을 변경할 때마다 aws-lambda-tools-defaults.json file에서 그에 상응하는 파일들을 업데이트해야 합니다.

```
{  
    "profile": "iam-execution-profile",  
    "region" : "region",  
    "configuration" : "Release",  
    "framework" : "netcoreapp2.1",  
    "function-runtime": "dotnetcore2.1",  
    "function-memory-size" : 256,  
    "function-timeout" : 30,  
    "function-handler" : "Assembly::Namespace.Class::Function"  
}
```

3. [Function.cs] 파일을 업니다. 함수 핸들러 코드를 구현할 수 있도록 템플릿이 제공됩니다.

```
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. Lambda 함수를 표현하는 코드를 작성하고 나면 애플리케이션의 프로젝트 노드를 마우스 오른쪽 버튼으로 클릭한 다음 Publish to AWS Lambda(AWS Lambda에 게시)를 선택하여 이를 업로드할 수 있습니다.
5. [Upload Lambda Function] 창에서 함수의 이름을 입력하거나 이전에 게시된 함수를 선택하여 다시 게시합니다. 그런 다음 [Next]를 선택합니다
6. [Advanced Function Details] 창에서 다음을 수행하십시오.
  - 앞서 언급한 IAM 역할인 [Role Name:]을 지정합니다.
  - (선택 사항) [Environment:]에서 사용하고자 하는 환경 변수를 지정합니다. 자세한 내용은 [AWS Lambda 환경 변수 \(p. 39\)](#) 단원을 참조하십시오.
  - (선택 사항) [Memory(MB):] 또는 [Timeout(Secs):] 구성을 지정합니다.
  - (선택 사항) Lambda 함수가 VPC 내부에서 실행 중인 리소스에 액세스해야 하는 경우 VPC 구성을 지정합니다. 자세한 내용은 [Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 \(p. 66\)](#) 단원을 참조하십시오.
  - [Next]를 선택한 다음 [Upload]를 선택하여 애플리케이션을 배포합니다.

자세한 내용은 [.NET Core CLI를 사용하여 AWS Lambda 프로젝트 배포](#)를 참조하십시오.

## AWS Lambda 함수 핸들러(C#)

Lambda 함수를 생성하는 시점에 서비스가 사용자를 대신하여 함수를 실행할 때 AWS Lambda가 호출할 수 있는 핸들러를 지정합니다.

클래스에서 Lambda 함수 핸들러를 인스턴스나 정적 메서드로 정의합니다. 원활 경우, 현재 실행에 대한 정보(예: 현재 함수의 이름, 메모리 한도, 남아 있는 실행 시간, 로깅 등)를 액세스하는 데 사용할 수 있는 인터페이스인 `ILambdaContext` 유형의 메서드 파라미터를 정의하여 Lambda 컨텍스트 객체에 액세스할 수 있습니다.

```
returnType handler-name(inputType input, ILambdaContext context) {  
    ...  
}
```

구문에서 다음 사항에 유의하십시오.

- `inputType` – 첫 번째 핸들러 파라미터는 핸들러에 대한 입력으로, 이벤트 데이터(이벤트 소스에서 게시) 또는 문자열이나 사용자 지정 데이터 객체 같은 사용자 지정 입력이 될 수 있습니다.
- `returnType` – Lambda 함수를 동기식으로 호출할 계획인 경우(`RequestResponse` 호출 유형 사용), 지원되는 데이터 유형 중 하나를 사용하여 함수의 출력을 반환할 수 있습니다. 예를 들어 함수를 모바일 애플리케이션 백엔드로 사용하는 경우에는 이를 동기식으로 호출합니다. 출력 데이터 유형이 JSON으로 직렬화됩니다.

Lambda 함수를 비동기식으로 호출할 계획인 경우(`Event` 호출 유형 사용), `returnType`은 `void`여야 합니다. 예를 들어 Amazon S3나 Amazon SNS 같은 이벤트 소스에서 AWS Lambda를 사용할 경우, 이러한 이벤트 소스는 `Event` 호출 유형을 사용하여 Lambda 함수를 호출합니다.

## 스트림 처리

기본적으로 `System.IO.Stream` 유형만 입력 파라미터로 지원됩니다.

예를 들어 다음과 같은 C# 코드 예제를 고려해 보십시오.

```
using System.IO;
```

```
namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

C# 코드 예제에서 첫 번째 핸들러 파라미터는 핸들러에 대한 입력(MyHandler)인데, 이벤트 데이터(Amazon S3 같이 이벤트 소스에서 계시)나 Stream(예제 참조) 또는 사용자 지정 데이터 객체와 같이 사용자가 제공한 사용자 지정 입력이 될 수 있습니다. 출력은 Stream 유형입니다.

## 표준 데이터 유형 처리

아래에 나와 있듯이 기타 모든 유형에서는 직렬 변환기를 지정해야 합니다.

- 기본 .NET 유형(문자열 또는 int).
- 모음 및 맵 - IList, IEnumerable, IList<T>, Array, IDictionary, IDictionary< TKey, TValue >
- POCO 유형(POCO)
- 사전 정의된 AWS 이벤트 유형
- 비동기식 호출의 경우, Lambda가 반환 유형을 무시합니다. 이러한 경우에 반환 유형은 무효로 설정될 수 있습니다.
- .NET 비동기식 프로그래밍을 사용하고 있는 경우에는 반환 유형이 Task and Task<T> 유형이고 async 및 await 키워드를 사용할 수 있습니다. 자세한 내용은 [AWS Lambda에서 C# 함수로 작성된 비동기식 핸들러 사용 \(p. 307\)](#) 단원을 참조하십시오.

함수 입력 및 출력 파라미터가 System.IO.Stream 유형이 아닌 경우에는 이를 직렬화해야 합니다. AWS Lambda가 애플리케이션의 어셈블리 또는 메서드 수준에서 적용이 가능한 기본 직렬 변환기를 제공하거나, 사용자가 Amazon.Lambda.Core 라이브러리에서 제공되는 ILambdaSerializer 인터페이스를 구현하여 자체적으로 정의할 수 있습니다. 자세한 내용은 [AWS Lambda 배포 패키지\(C#\) \(p. 295\)](#) 단원을 참조하십시오.

메서드에 기본 직렬 변환기 속성을 추가하려면 먼저 project.json 파일에서 Amazon.Lambda.Serialization.Json에 대한 종속 프로그램을 추가합니다.

```
{
    "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"
        }
    }
}
```

아래 예제는 하나의 메서드와 또 다른 메서드에서 기본 Json.NET 직렬 변환기를 지정함으로써 유연성을 높일 수 있음을 보여줍니다.

```
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);
    }
}
```

## 핸들러 서명

Lambda 함수를 생성할 때 핸들러 문자열을 제공하여 호출하려는 코드를 어디에서 찾을지 AWS Lambda에게 알려줘야 합니다. C#에서 형식은 다음과 같습니다.

**ASSEMBLY::TYPE::METHOD** 위치:

- **ASSEMBLY**는 애플리케이션에 대한 .NET 어셈블리 파일의 이름입니다. .NET Core CLI를 사용하여 애플리케이션을 빌드할 때 project.json에서 buildOptions.outputName 설정을 사용하여 어셈블리 이름을 설정하지 않은 경우에는 **ASSEMBLY** 이름이 project.json 파일을 포함하는 폴더의 이름이 됩니다. 자세한 내용은 [.NET Core CLI \(p. 296\)](#) 단원을 참조하십시오. 이 예제의 경우, 폴더 이름이 HelloWorldApp라고 가정해 봅시다.
- **TYPE**는 핸들러 유형의 전체 이름으로, **Namespace**와 **ClassName**으로 이루어져 있습니다. 이 예제의 경우 `Example.Hello`입니다.
- **METHOD**는 함수 핸들러의 이름으로, 이 예제의 경우 `MyHandler`입니다.

궁극적으로 서명은 **Assembly::Namespace.ClassName::MethodName**라는 형식을 갖게 됩니다.

다시 한번 다음 예제를 고려해보십시오.

```
using System.IO;

namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

핸들러 문자열은 `HelloWorldApp::Example.Hello::MyHandler`가 됩니다.

### Important

핸들러 문자열에 지정된 메서드가 오버로드된 경우에는 Lambda가 호출해야 하는 메서드의 정확한 서명을 제공해야 합니다. 그렇지 않으면 확인 시 여러 개의 서명 중에서 선택을 해야 하는 경우(오버로드), AWS Lambda가 유효한 서명을 거부하게 됩니다.

## Lambda 함수 핸들러 제한 사항

핸들러 서명에는 몇 가지 제한이 있습니다.

- `unsafe`이어서는 안 되고, 핸들러 메서드 및 종속 프로그램 내에서 `unsafe` 콘텍스트가 사용될 수는 있지만 핸들러 서명에서 포인터 유형을 사용해서는 안 됩니다. 자세한 내용은 [unsafe\(C# Reference\)](#)를 참조하십시오.
- `params` 키워드를 사용하여 다수의 파라미터를 전달하거나 `ArgIterator`를 입력으로 사용하거나 다수의 파라미터를 지원하는 데 사용되는 파라미터를 반환하지 않을 수 있습니다.
- 핸들러는 일반 메서드(예: `IList<T> Sort<T>(IList<T> input)`)가 아닙니다.
- 서명 `async void`이 있는 비동기식 핸들러는 지원되지 않습니다.

## AWS Lambda에서 C# 함수로 작성된 비동기식 핸들러 사용

Lambda 함수에 대용량 파일을 Amazon S3에 업로드하거나 대규모 레코드 스트림을 DynamoDB에서 읽어오는 등의 장기 실행 프로세스가 필요하다고 판단되는 경우에는 `async/await` 패턴을 활용할 수 있습니다. 이 서명을 사용하면 Lambda는 함수를 동기식으로 실행하고 그 함수가 응답을 반환할 때까지 또는 실행이 [시간 초과 \(p. 35\)](#)될 때까지 기다립니다.

```
public async Task<Response> ProcessS3ImageResizeAsync(SimpleS3Event input)
{
    var response = await client.DoAsyncWork(input);
    return response;
}
```

이 패턴을 사용하는 경우에는 몇 가지 고려해야 할 사항이 있습니다.

- AWS Lambda는 `async void` 메서드를 지원하지 않습니다.
- `await` 연산자를 구현하지 않고 비동기식 Lambda 함수를 생성한 경우, .NET은 컴파일러 경고를 발행하고 사용자는 예상하지 않은 동작을 관찰하게 됩니다. 예를 들어, 일부 비동기 작업은 다른 작업이 실행되지 않을 때 실행됩니다. 또는 일부 비동기 작업은 함수 실행이 완료되어야 완료됩니다.

```
public async Task ProcessS3ImageResizeAsync(SimpleS3Event event) // Compiler warning
{
    client.DoAsyncWork(input);
}
```

- Lambda 함수에는 동시 호출이 가능한 비동기식 호출이 여러 개 포함될 수 있습니다. `Task.WhenAll` 및 `Task.WhenAny` 메서드를 사용하여 여러 작업을 수행할 수 있습니다. `Task.WhenAll` 메서드를 사용하면서 작업 목록을 메서드에 배열로 전달해야 합니다. 아래 예제와 같이 배열에 어떤 작업도 포함시키지 않으면 작업이 완료되기 전에 해당 호출이 반환될 수 있습니다.

```
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);
}
```

`Task.WhenAny` 메서드를 사용하려면 다시 한 번 작업 목록을 메서드에 배열로 전달해야 합니다. 다른 작업들은 여전히 실행 중이더라도 첫 번째 작업이 완료되는 즉시 호출이 반환됩니다.

```
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);  
}
```

## AWS Lambda 콘텍스트 객체(C#)

Lambda은 함수를 실행할 때 컨텍스트 객체를 [핸들러 \(p. 304\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 속성에 제공합니다.

### 컨텍스트 속성

- `FunctionName` – Lambda 함수의 이름
- `FunctionVersion` – 함수의 [버전 \(p. 45\)](#)
- `InvokedFunctionArn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `MemoryLimitInMB` – 함수에 구성된 메모리 양
- `AwsRequestId` – 호출 요청의 식별자
- `LogGroupName` – 함수에 대한 로그 그룹
- `LogStreamName` – 함수 인스턴스에 대한 로그 스트림
- `RemainingTime(TimeSpan)` – 실행 시간이 초과되기 전에 남은 시간(밀리초)
- `Identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
- `ClientContext` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트
- `Logger` 함수에 대한 [로거 객체 \(p. 308\)](#)입니다.

다음 C# 코드 조각은 일부 콘텍스트 정보를 인쇄하는 간단한 핸들러 함수를 보여줍니다.

```
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);  
}
```

## AWS Lambda 함수 로깅(C#)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드의 로그를 출력하려면 [콘솔 클래스](#)에서 메서드를 사용하거나, `stdout` 또는 `stderr`에 쓰는 로깅 라이브러리를 사용합니다. 다음 예시는 호출에 대한 요청 ID를 로깅합니다.

```
public class ProductService
```

```
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request)  
    {  
        Console.WriteLine("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

또한 Lambda는(는) Amazon.Lambda.Core 라이브러리에 로거 클래스를 제공합니다.  
Amazon.Lambda.Core.LambdaLogger 클래스에서 Log 메서드를 사용하여 로그를 작성합니다.

```
using Amazon.Lambda.Core;  
  
public class ProductService  
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request)  
    {  
        LambdaLogger.Log("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

이 클래스의 인스턴스는 [컨텍스트 객체 \(p. 308\)](#)에서도 사용할 수 있습니다.

```
public class ProductService  
{  
    public async Task<Product> DescribeProduct(DescribeProductRequest request,  
        ILambdaContext context)  
    {  
        context.Logger.Log("DescribeProduct invoked with Id " + request.Id);  
        return await catalogService.DescribeProduct(request.Id);  
    }  
}
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

#### Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스 \(p. 101\)](#)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 --log-type 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 LogResult 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail  
{  
    "StatusCode": 200,  
    "LogResult":  
    "U1RBUL0gUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"
```

}

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms          Billed
Duration: 100 ms           Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(C#)

Lambda 함수에서 예외가 발생하면 Lambda는 예외 정보를 사용자에게 보고합니다. 두 가지 경우에 예외가 발생할 수 있습니다.

- 초기화(Lambda가 코드를 로딩하고 핸들러 문자열을 검증한 다음, 정적 객체가 아닐 경우 클래스의 인스턴스를 생성).
- Lambda 함수 호출.

직렬화된 예외 정보는 모델링된 JSON 객체로서 페이로드 형태로 반환되고 CloudWatch 로그에 출력됩니다.

초기화 단계에서 유효하지 않은 핸들러 문자열, 규칙 위반 유형 또는 메서드([Lambda 함수 핸들러 제한 사항 \(p. 306\)](#) 참조), 기타 검증 메서드(직렬 변환기 속성을 잊었거나 입력 및 출력 유형으로 POCO를 사용하는 경우)에 대해 예외가 발생할 수 있습니다. 이러한 예외는 `LambdaException` 유형을 가집니다. 예:

```
{
  "errorType": "LambdaException",
  "errorMessage": "Invalid lambda function handler: 'http://this.is.not.a.valid.handler/'.
  The valid format is 'ASSEMBLY::TYPE::METHOD'."}
```

생성자에서 예외가 발생한 경우의 오류 유형 역시 `LambdaException`이지만, 구성 중에 발생한 예외는 모델링된 예외 객체 그 자체인 `cause` 속성에 제공됩니다.

```
{
  "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": "ArithmaticException",
        "errorMessage": "Sorry, 2 + 2 = 5",
        "stackTrace": [
            "at LambdaExceptionTestFunction.ThrowExceptionInConstructor..ctor()"
        ]
    }
}
```

예제에서 알 수 있듯이, 내부 예외는 항상 보존되며(cause 속성으로) 깊숙이 중첩될 수 있습니다.

예외는 호출 중에도 발생할 수 있습니다. 이 경우, 예외 유형이 보존되며 예외가 CloudWatch 로그에서 페이지로드 형태로 직접 반환됩니다. 예:

```
{
  "errorType": "AggregateException",
  "errorMessage": "One or more errors occurred. (An unknown web exception occurred!)",
  "stackTrace": [
    "at System.Threading.Tasks.Task.ThrowIfExceptional(Boolean
includeTaskCanceledExceptions)",
    "at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)",
    "at lambda_method(Closure , Stream , Stream , ContextInfo )"
  ],
  "cause": {
    "errorType": "UnknownWebException",
    "errorMessage": "An unknown web exception occurred!",
    "stackTrace": [
      "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()",
      "--- End of stack trace from previous location where exception was thrown ---",
      "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
      "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
      "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",
      "at LambdaDemo107.LambdaEntryPoint.<CheckWebsiteStatus>d__0.MoveNext()"
    ],
    "cause": {
      "errorType": "WebException",
      "errorMessage": "An error occurred while sending the request. SSL peer certificate or
SSH remote key was not OK",
      "stackTrace": [
        "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)",
        "at System.Threading.Tasks.TaskFactory`1.FromAsyncCoreLogic(IAsyncResult iar,
Func`2 endFunction, Action`1 endAction, Task`1 promise, Boolean requiresSynchronization)",
        "--- End of stack trace from previous location where exception was thrown ---",
        "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
        "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",
        "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()"
      ],
      "cause": {
        "errorType": "HttpRequestException",
        "errorMessage": "An error occurred while sending the request.",
        "stackTrace": [
          "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
          "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)"
        ]
      }
    }
  }
}
```

```
"at System.Net.Http.HttpClient.<FinishSendAsync>d__58.MoveNext()",  
"--- End of stack trace from previous location where exception was thrown ---",  
"at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",  
"at  
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task  
task)",  
"at System.Net.HttpWebRequest.<SendRequest>d__63.MoveNext()",  
"--- End of stack trace from previous location where exception was thrown ---",  
"at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",  
"at  
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task  
task)",  
"at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)"  
],  
"cause":  
{  
"errorType": "CurlException",  
"errorMessage": "SSL peer certificate or SSH remote key was not OK",  
"stackTrace": [  
"at System.Net.Http.CurlHandler.ThrowIfCURLError(CURLcode error)",  
"at  
System.Net.Http.CurlHandler.MultiAgent.FinishRequest(StrongToWeakReference`1 easyWrapper,  
CURLcode messageResult)"  
]  
}  
}  
}  
}  
}
```

오류 정보가 전달되는 메서드는 호출 유형에 따라 다릅니다.

- **RequestResponse** 호출 유형(즉, 동기식 실행): 이 경우에는 오류 메시지를 다시 수신합니다.

예를 들어 Lambda 콘솔을 사용하여 Lambda 함수를 호출한 경우에는 RequestResponse가 항상 호출 유형이 되고, 콘솔의 실행 결과 섹션에 AWS Lambda가 반환한 오류 정보가 표시됩니다.

- **Event** 호출 유형(즉, 비동기식 실행): 이 경우 AWS Lambda가 어떤 값도 반환하지 않습니다. 대신에 CloudWatch Logs 및 CloudWatch 측정치의 오류 정보를 로깅합니다.

이벤트 소스에 따라 AWS Lambda는 실패한 Lambda 함수를 다시 시도할 수 있습니다. 자세한 내용은 [AWS Lambda 재시도 동작 \(p. 83\)](#) 단원을 참조하십시오.

## 함수 오류 처리

사용자 지정 오류 처리를 생성하여 Lambda 함수에서 직접 예외를 발생시키고 AWS Step Functions 상태 시스템 내에서 직접 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

CreateAccount 상태가 Lambda 함수를 사용하여 고객의 세부 정보를 데이터베이스에 기록하는 작업인 경우를 생각해 보십시오.

- 작업이 성공하면 계정이 만들어지고 환영 이메일이 전송됩니다.
- 사용자가 이미 존재하는 사용자 이름에 대해 계정을 만들려고 시도하면 Lambda 함수는 오류를 발생시켜 상태 시스템이 다른 사용자 이름을 제안하고 계정 생성 프로세스를 재시도하도록 만들습니다.

다음 코드 샘플은 이 작업을 수행하는 방법을 보여줍니다. C#의 사용자 지정 오류는 `Exception` 클래스를 확장해야 합니다.

```
namespace Example {  
    public class AccountAlreadyExistsException : Exception {
```

```
        public AccountAlreadyExistsException(String message) :  
            base(message) {}  
    }  
  
    namespace Example {  
        public class Handler {  
            public static void CreateAccount() {  
                throw new AccountAlreadyExistsException("Account is in use!");  
            }  
        }  
    }
```

Catch 규칙을 사용하여 오류를 파악하도록 Step Functions를 구성할 수 있습니다. Lambda는 런타임에 오류 이름을 예외의 간단한 클래스 이름으로 자동 설정합니다.

```
{  
    "StartAt": "CreateAccount",  
    "States": {  
        "CreateAccount": {  
            "Type": "Task",  
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",  
            "Next": "SendWelcomeEmail",  
            "Catch": [  
                {  
                    "ErrorEquals": [ "AccountAlreadyExistsException" ],  
                    "Next": "SuggestAccountName"  
                }  
            ]  
        },  
        ...  
    }  
}
```

런타임에 AWS Step Functions는 오류를 파악하여 Next 전환에 지정된 대로 SuggestAccountName 상태로 [전환](#)을 합니다.

사용자 지정 오류 처리는 [서버리스](#) 애플리케이션을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda [프로그래밍 모델](#) (p. 31)에서 지원되는 모든 언어와 통합이 되기 때문에 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서비스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

# PowerShell을 사용하여 Lambda 함수 빌드

다음 단원에서는 [일반적인 프로그래밍 패턴 및 핵심 개념](#)이 PowerShell에서 Lambda 함수 코드를 작성할 때 어떻게 적용되는지 설명합니다.

## .NET 런타임

이름	식별자	언어	운영 체제
.NET Core 2.1	dotnetcore2.1	C# PowerShell Core 6.0	Amazon Linux
.NET Core 1.0	dotnetcore1.0	C#	Amazon Linux

PowerShell에서 Lambda 함수를 사용하려면 PowerShell Core 6.0이 필요합니다. Windows PowerShell은 지원되지 않습니다.

시작하기 전에 PowerShell 개발 환경을 먼저 설정해야 합니다. 작업 방법에 대한 지침은 [PowerShell 개발 환경 설정 \(p. 315\)](#) 단원을 참조하십시오.

AWSLambdaPSCore 모듈을 사용해 템플릿에서 샘플 PowerShell 프로젝트를 다운로드하고, PowerShell 배포 패키지를 생성하고, AWS 클라우드에 PowerShell 함수를 배포하는 방법을 알아보려면 [AWSLambdaPSCore 모듈 사용 \(p. 315\)](#) 단원을 참조하십시오.

## 주제

- [AWS Lambda 배포 패키지\(PowerShell\) \(p. 314\)](#)
- [AWS Lambda 함수 핸들러\(PowerShell\) \(p. 317\)](#)
- [AWS Lambda 콘텍스트 객체\(PowerShell\) \(p. 318\)](#)
- [AWS Lambda 함수 로깅\(PowerShell\) \(p. 319\)](#)
- [AWS Lambda 함수 오류\(PowerShell\) \(p. 320\)](#)

## AWS Lambda 배포 패키지(PowerShell)

PowerShell Lambda 배포 패키지는 PowerShell 스크립트, PowerShell 스크립트에 필요한 PowerShell 모듈과 PowerShell Core를 호스팅하는 데 필요한 어셈블리가 포함된 ZIP 파일입니다.

AWSLambdaPSCore는 [PowerShell Gallery](#)에서 설치할 수 있는 PowerShell 모듈입니다. 이 모듈을 사용해 PowerShell Lambda 배포 패키지를 생성할 수 있습니다.

스크립트에서 사용하는 모듈을 나타내기 위해서는 PowerShell 스크립트 내에서 `#Requires` 문을 사용해야 합니다. 이 문은 두 가지 중요한 작업을 수행하는데 바로, 1) 다른 개발자에게 스크립트가 사용하는 모듈을 알리는 작업과 2) 배포의 일부로 스크립트를 패키징하는 데 AWS PowerShell 도구에 필요한 종속 모듈을 식별하는 작업입니다. PowerShell의 `#Requires` 문에 대한 자세한 내용은 [Requires 정보](#)를 참조하십시오.

PowerShell 배포 패키지에 대한 자세한 내용은 [AWS Lambda 배포 패키지\(PowerShell\) \(p. 314\)](#) 단원을 참조하십시오.

PowerShell Lambda 함수가 AWS PowerShell cmdlet을 사용하는 경우 Windows PowerShell만 지원하는 `AWSPowerShell` 모듈이 아니라 PowerShell Core를 지원하는 `AWSPowerShell.NetCore` 모듈을 참조하는 `#Requires` 문을 설정해야 합니다. cmdlet 가져오기 프로세스를 최적화하는 `AWSPowerShell.NetCore` 버전 3.3.270.0 이상을 사용해야 합니다. 이전 버전을 사용하는 경우 콜드 부팅 시간이 더 길어집니다. 자세한 내용은 [PowerShell용 AWS 도구](#) 페이지를 참조하십시오.

시작하기 전에 PowerShell 개발 환경을 먼저 설정해야 합니다. 작업 방법에 대한 지침은 [PowerShell 개발 환경 설정 \(p. 315\)](#) 단원을 참조하십시오.

## PowerShell 개발 환경 설정

PowerShell 스크립트 작성하기 위한 개발 환경을 설정하려면 다음 작업을 수행합니다.

1. 올바른 버전의 PowerShell을 설치합니다. Lambda의 PowerShell 지원은 크로스 플랫폼 PowerShell Core 6.0 릴리스를 기반으로 합니다. 즉, Windows, Linux 또는 Mac에서 PowerShell Lambda 함수를 개발할 수 있습니다. 이 버전의 PowerShell이 설치되어 있지 않은 경우 [PowerShell Core 설치](#)에서 지침을 확인할 수 있습니다.
2. .NET Core 2.1 SDK를 설치합니다. PowerShell Core는 .NET Core를 기반으로 빌드되었기 때문에 Lambda의 PowerShell 지원은 .NET Core와 PowerShell Lambda 함수 모두에 동일한 .NET Core 2.1 Lambda 런타임을 사용합니다. .NET Core 2.1 SDK는 새 Lambda PowerShell 게시 cmdlet에서 Lambda 배포 패키지를 생성하는데 사용합니다. .NET Core 2.1 SDK는 Microsoft 웹사이트의 [.NET 다운로드](#)에서 구할 수 있습니다. 런타임이 아니라 SDK를 설치해야 합니다.
3. AWSLambdaPSCore 모듈을 설치합니다. 이 모듈은 [PowerShell Gallery](#)에서 또는 다음 PowerShell Core 셀 명령을 사용하여 설치할 수 있습니다.

```
Install-Module AWSLambdaPSCore -Scope CurrentUser
```

## 다음 단계

- PowerShell로 Lambda 함수 작성에 대해 알아보려면 [PowerShell을 사용하여 Lambda 함수 빌드 \(p. 314\)](#) 단원을 참조하십시오.
- AWSLambdaPSCore 모듈을 사용해 템플릿에서 샘플 PowerShell 프로젝트를 다운로드하고, PowerShell 배포 패키지를 생성하고, AWS 클라우드에 PowerShell 함수 배포에 대한 자세한 내용은 [AWSLambdaPSCore 모듈 사용 \(p. 315\)](#) 단원을 참조하십시오.

## AWSLambdaPSCore 모듈 사용

AWSLambdaPSCore 모듈에는 PowerShell Lambda 함수 작성 및 게시에 유용하게 사용할 수 있는 다음과 같은 새 cmdlet이 있습니다.

Cmdlet 이름	설명
<code>Get#AWSPowerShellLambdaTemplate</code>	시작 템플릿 목록을 반환합니다.
<code>New#AWSPowerShellLambda</code>	템플릿을 기반으로 최초의 PowerShell 스크립트를 생성합니다.
<code>Publish#AWSPowerShellLambda</code>	지정한 PowerShell 스크립트를 Lambda에 게시합니다.

Cmdlet 이름	설명
New-AWSPowerShellLambdaPackage	배포를 위해 CI/CD 시스템에서 사용할 수 있는 Lambda 배포 패키지를 만듭니다.

Lambda를 사용하여 PowerShell 스크립트를 작성해 배포하려면 New-AWSPowerShellLambda cmdlet을 사용하여 템플릿을 기반으로 스타터 스크립트를 생성할 수 있습니다. Publish-AWSPowerShellLambda cmdlet을 사용해 스크립트를 AWS Lambda에 배포할 수 있습니다. 그런 다음 명령줄 또는 콘솔에서 스크립트를 테스트할 수 있습니다.

새 PowerShell 스크립트를 생성한 다음 업로드 및 테스트하려면 다음 절차를 수행합니다.

#### 1. 사용 가능한 템플릿 확인

다음 명령을 실행해 사용 가능한 템플릿 목록을 확인합니다.

```
PS C:\> Get-AWSPowerShellLambdaTemplate
Template          Description
-----
Basic            Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
DetectLabels     Use Amazon Rekognition service to tag image files in Amazon S3
                  with detected labels.
KinesisStreamProcessor Script to process an Amazon Kinesis stream
S3Event          Script to process S3 events
SNSSubscription Script to be subscribed to an Amazon SNS topic
SQSQueueProcessor Script to be subscribed to an Amazon SQS queue
```

새 템플릿은 항상 표시되고, 위의 출력 내용은 예일 뿐입니다.

#### 2. 기본 스크립트 작성

다음 명령을 실행해 Basic 템플릿을 토대로 샘플 스크립트를 작성합니다.

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

현재 디렉터리의 새 하위 디렉터리에 MyFirstPSScript.ps1이라는 새 파일이 생성됩니다. 이 디렉터리의 이름은 -ScriptName 파라미터를 기반으로 지정됩니다. -Directory 파라미터를 사용하면 대체 디렉터리를 선택할 수 있습니다.

새 파일의 내용은 다음과 같습니다.

```
# PowerShell script file to be executed as a AWS Lambda function.
#
# When executing in Lambda the following variables will be predefined.
#   $LambdaInput - A PSObject that contains the Lambda function input data.
#   $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
#                   information about the currently running Lambda environment.
#
# The last item in the PowerShell pipeline will be returned as the result of the Lambda
# function.
#
# To include PowerShell modules with your Lambda function, like the
# AWSPowerShell.NetCore module, add a "#Requires" statement
# indicating the module and version.

#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}

# Uncomment to send the input to CloudWatch Logs
```

```
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

### 3. 샘플 스크립트 편집

PowerShell 스크립트의 로그 메시지가 CloudWatch Logs로 전송되는 방법을 알아보려면 샘플 스크립트의 `Write-Host` 행의 주석 처리를 해제하십시오.

Lambda 함수에서 데이터를 다시 반환하는 방법을 확인하려면 `$PSVersionTable`을 사용해 스크립트 끝에 새 행을 추가합니다. 그러면 PowerShell 파이프라인에 `$PSVersionTable` 테이블이 추가됩니다. PowerShell 스크립트가 완성되면 PowerShell 파이프라인의 마지막 객체가 Lambda 함수의 반환 데이터입니다. `$PSVersionTable`은 실행 환경에 대한 정보도 제공하는 PowerShell 전역 변수입니다.

이와 같이 변경한 후 샘플 스크립트의 마지막 두 개 행은 다음과 같습니다.

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)  
$PSVersionTable
```

### 4. AWS Lambda에 게시

`MyFirstPSScript.ps1` 파일 편집 후 디렉터리를 스크립트가 있는 위치로 변경합니다. 다음 명령을 실행하여 스크립트를 AWS Lambda에 게시합니다.

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name MyFirstPSScript -Region us-east-1
```

`-Name` 매개변수는 Lambda 콘솔에 나타나는 Lambda 함수 이름을 지정합니다. 이 함수를 사용해 스크립트를 수동으로 호출할 수 있습니다.

### 5. Lambda 함수 테스트

명령 프롬프트에서 `dotnet CLI`를 사용하여 방금 게시한 PowerShell Lambda 함수를 테스트할 수 있습니다. `lambda invoke-function` 명령을 사용하여 함수를 호출합니다.

```
> dotnet lambda invoke-function MyFirstPSScript
```

`dotnet CLI` 확장 기능에 대한 자세한 내용은 [.NET Core CLI \(p. 296\)](#) 단원을 참조하십시오.

## AWS Lambda 함수 핸들러(PowerShell)

Lambda 함수가 호출되면 Lambda 핸들러가 PowerShell 스크립트를 호출합니다.

PowerShell 스크립트가 호출되면 다음 변수가 미리 정의되어 있습니다.

- `$LambdaInput` – 핸들러에 대한 입력이 포함된 PSObject입니다. 이 입력은 이벤트 데이터(이벤트 소스에서 게시)가 되거나, 문자열이나 사용자 지정 데이터 객체 같은 사용자 지정 입력이 될 수 있습니다.
- `$LambdaContext` – 현재 실행에 대한 정보(예: 현재 함수의 이름, 메모리 제한, 남아 있는 실행 시간, 로깅 등)에 액세스하는 데 사용할 수 있는 `Amazon.Lambda.Core.ILambdaContext` 객체입니다.

예를 들어 다음과 같은 PowerShell 예제 코드를 고려해 보십시오.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}  
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

이 스크립트는 `$LambdaContext` 변수에서 얻은 `FunctionName` 속성을 반환합니다.

### Note

스크립트에서 사용하는 모듈을 나타내기 위해서는 PowerShell 스크립트 내에서 `#Requires` 문을 사용해야 합니다. 이 문은 두 가지 중요한 작업을 수행하는데 바로, 1) 다른 개발자에게 스크립트가 사용하는 모듈을 알리는 작업과 2) 배포의 일부로 스크립트를 패키징하는데 AWS PowerShell 도구에 필요한 종속 모듈을 식별하는 작업입니다. PowerShell의 `#Requires` 문에 대한 자세한 내용은 [Requires 정보](#)를 참조하십시오. PowerShell 배포 패키지에 대한 자세한 내용은 [AWS Lambda 배포 패키지\(PowerShell\)](#) (p. 314) 단원을 참조하십시오.

PowerShell Lambda 함수가 AWS PowerShell cmdlet을 사용하는 경우 Windows PowerShell만 지원하는 `AWS PowerShell` 모듈이 아니라 PowerShell Core를 지원하는 `AWS PowerShell .NetCore` 모듈을 참조하는 `#Requires` 문을 설정해야 합니다. cmdlet 가져오기 프로세스를 최적화하는 `AWS PowerShell .NetCore` 버전 3.3.270.0 이상을 사용해야 합니다. 이전 버전을 사용하는 경우 콜드 부팅 시간이 더 길어집니다. 자세한 내용은 [PowerShell용 AWS 도구](#) 페이지를 참조하십시오.

## 데이터 반환

일부 Lambda 호출은 호출자에게 데이터가 반환됨을 의미합니다. 예를 들어, API 게이트웨이의 웹 요청에 대한 응답으로 호출이 발생한 경우 Lambda 함수는 해당 응답을 다시 반환해야 합니다. PowerShell Lambda의 경우 PowerShell 파이프라인에 추가된 마지막 객체는 Lambda 호출의 반환 데이터입니다. 객체가 문자열인 경우 데이터는 있는 그대로 반환됩니다. 그렇지 않으면 `ConvertTo-Json` cmdlet을 사용해 객체가 JSON으로 변환됩니다.

예를 들어, PowerShell 파이프라인에 `$PSVersionTable`을 추가하는 다음 PowerShell 문에 대해 생각해 보십시오.

```
$PSVersionTable
```

PowerShell 스크립트가 완성되면 PowerShell 파이프라인의 마지막 객체가 Lambda 함수의 반환 데이터입니다. `$PSVersionTable`은 실행 환경에 대한 정보도 제공하는 PowerShell 전역 변수입니다.

## AWS Lambda 컨텍스트 객체(PowerShell)

Lambda 함수를 실행하면 `$LambdaContext` 변수를 [핸들러](#) (p. 317)에 제공하여 컨텍스트 정보를 전달합니다. 이 변수는 호출, 함수 및 실행 환경에 관한 정보를 메서드 및 속성에 제공합니다.

### 컨텍스트 속성

- `FunctionName` – Lambda 함수의 이름
- `FunctionVersion` – 함수의 [버전](#) (p. 45)
- `InvokedFunctionArn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `MemoryLimitInMB` – 함수에 구성된 메모리 양
- `AwsRequestId` – 호출 요청의 식별자
- `LogGroupName` – 함수에 대한 로그 그룹
- `LogStreamName` – 함수 인스턴스에 대한 로그 스트림
- `RemainingTime` – 실행 시간이 초과되기 전에 남은 시간(밀리초)
- `Identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
- `ClientContext` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트
- `Logger` – 함수에 대한 [로거 객체](#) (p. 319)입니다.

다음 PowerShell 코드 조각은 일부 컨텍스트 정보를 인쇄하는 간단한 핸들러 함수를 보여줍니다.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.343.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

## AWS Lambda 함수 로깅(PowerShell)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드에서 로그를 출력하려면 [Microsoft.PowerShell.Utility](#)에서 cmdlets를 사용하거나, `stdout` 또는 `stderr`에 쓰는 로깅 모듈을 사용합니다. 다음 예에는 `Write-Host`가 사용됩니다.

```
Write-Host 'Event received.'
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 호출에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/*function-name*)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스 \(p. 101\)](#)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오픈 소스 샘플 애플리케이션 \(p. 116\)](#) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 `--log-type` 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 `LogResult` 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms          Billed
Duration: 100 ms           Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(PowerShell)

Lambda 함수에 종료 오류가 있는 경우 AWS Lambda에서는 오류를 인식하고 오류 정보를 JSON으로 직렬화한 다음 반환합니다.

다음과 같은 PowerShell 스크립트 예제 문을 생각해 보십시오.

```
throw 'The Account is not found'
```

이 Lambda 함수를 호출하면 종료 오류가 발생하고 AWS Lambda는 다음과 같은 오류 메시지를 반환합니다.

```
{
    "errorMessage": "The Account is not found",
    "errorType": "RuntimeException"
}
```

errorType은 PowerShell에서 생성하는 기본 예외인 `RuntimeException`입니다. 다음과 같은 오류를 발생시켜 사용자 지정 오류 유형을 사용할 수 있습니다.

```
throw @{'Exception'='AccountNotFound'; 'Message'='The Account is not found'}
```

이 오류 메시지는 `AccountNotFound`로 설정된 `errorType`으로 직렬화됩니다.

```
{
    "errorMessage": "The Account is not found",
    "errorType": "AccountNotFound"
}
```

오류 메시지가 필요 없는 경우에는 오류 코드 형식으로 문자열을 표시할 수 있습니다. 이러한 오류 코드 형식을 사용할 때 문자열은 문자로 시작해야 하고 그 뒤에는 공백 또는 기호 없이 문자와 숫자만 와야 합니다.

예를 들어, Lambda 함수에 다음 내용이 포함된 경우

```
throw 'AccountNotFound'
```

오류는 다음과 같이 직렬화됩니다.

```
{
    "errorMessage": "AccountNotFound",
    "errorType": "AccountNotFound"
}
```

## 함수 오류 처리

Lambda 함수에 사용자 지정 `errorType`을 사용해 AWS Step Functions 상태 시스템 내에서 직접 함수 오류를 처리(Retry 또는 Catch)할 수 있습니다. 자세한 정보는 [상태 머신을 사용하여 오류 조건 처리](#)를 참조하십시오.

사용자 지정 오류 처리는 [서버리스](#) 애플리케이션을 보다 손쉽게 생성할 수 있게 해줍니다. 이 기능은 Lambda 프로그래밍 모델(p. 31)에서 지원하는 모든 언어와 통합됩니다. 따라서 진행 상황에 따라 믹스 앤 매치하여 선택한 프로그래밍 언어로 애플리케이션을 설계할 수 있습니다.

AWS Step Functions 및 AWS Lambda를 사용하여 자체 서비스 애플리케이션을 생성하는 방법에 대한 자세한 내용은 [AWS Step Functions](#)를 참조하십시오.

# Ruby를 사용하여 Lambda 함수 빌드

Ruby 코드는 AWS Lambda에서 실행할 수 있습니다. Lambda은 Ruby에서 이벤트를 처리하기 위해 코드를 실행하는 [런타임 \(p. 99\)](#)을 제공합니다. 코드는 사용자가 관리하는 AWS Identity and Access Management(IAM) 역할의 자격 증명을 사용하여 Ruby용 AWS SDK가 포함된 환경에서 실행됩니다.

Lambda은 다음과 같은 Ruby 런타임을 지원합니다.

## Ruby 런타임

이름	식별자	운영 체제
Ruby 2.5	ruby2.5	Amazon Linux

함수 개발을 위한 실행 역할이 아직 없다면 이 역할을 하나 생성합니다.

## 실행 역할을 만들려면

1. IAM 콘솔에서 [역할 페이지](#)를 엽니다.
2. 역할 생성을 선택합니다.
3. 다음 속성을 사용하여 역할을 만듭니다.
  - 신뢰할 수 있는 엔터티 – Lambda.
  - 권한 – AWSLambdaBasicExecutionRole.
  - 역할 이름 – **lambda-role**.

AWSLambdaBasicExecutionRole 정책은 함수가 CloudWatch Logs에 로그를 쓰는 데 필요한 권한을 가집니다.

나중에 권한을 역할에 추가하거나 단일 함수와 관련된 다른 역할로 바꿀 수 있습니다.

## Ruby 함수를 만들려면

1. [Lambda 콘솔](#)을 엽니다.
2. 함수 생성을 선택합니다.
3. 다음 설정을 구성합니다:
  - 이름 – **ruby-function**.
  - 런타임 – Ruby 2.5.
  - 역할 – 기존 역할을 선택합니다.
  - 기존 역할 – **lambda-role**.
4. 함수 생성을 선택합니다.
5. 테스트 이벤트를 구성하려면 테스트를 선택합니다.
6. 이벤트 이름에 **test**를 입력합니다.
7. Create를 선택합니다.
8. 함수를 실행하려면 테스트를 선택합니다.

콘솔은 `lambda_function.rb`라는 단일 소스 파일로 Lambda 함수를 생성합니다. 이 파일을 편집하고 기본 제공 [코드 편집기 \(p. 24\)](#)에서 더 많은 파일을 추가할 수 있습니다. 저장을 선택하여 변경 사항을 저장하고, 테스트를 선택하여 코드를 실행하십시오.

### Note

Lambda 콘솔은 AWS Cloud9을 사용하여 브라우저에서 통합 개발 환경(IDE)을 제공합니다. AWS Cloud9를 사용하면 자신의 환경에서 Lambda 함수를 개발할 수도 있습니다. 자세한 내용은 AWS Cloud9 사용 설명서에서 [AWS Lambda 함수 작업 단원](#)을 참조하십시오.

`lambda_function.rb`는 이벤트 객체와 컨텍스트 객체를 취하는 `lambda_handler`라는 이름의 함수를 정의합니다. 이것은 함수가 호출될 때 Lambda이 호출하는 [핸들러 함수 \(p. 323\)](#)입니다. Ruby 함수 런타임은 Lambda에서 호출 이벤트를 가져와 핸들러로 전달합니다.

함수 코드를 저장할 때마다 Lambda 콘솔은 함수 코드가 포함된 ZIP 아카이브인 배포 패키지를 생성합니다. 함수 개발이 진행되면 소스 제어에 함수 코드를 저장하고 라이브러리를 추가하며 배포를 자동화할 필요가 있습니다. 먼저 [배포 패키지를 생성 \(p. 324\)](#)하고 명령줄에서 코드를 업데이트하십시오.

함수 런타임은 호출 이벤트 외에도 컨텍스트 객체를 핸들러에 전달합니다. [컨텍스트 객체 \(p. 326\)](#)에는 호출, 함수 및 실행 환경에 관한 추가 정보가 포함되어 있습니다. 자세한 내용은 환경 변수에서 확인할 수 있습니다.

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공됩니다. 함수 런타임은 각 호출에 대한 세부 정보를 CloudWatch Logs로 전송하고 호출 중에 [함수가 출력하는 모든 로그 \(p. 326\)](#)를 중계합니다. 함수가 [오류를 반환 \(p. 327\)](#)하면 Lambda는 오류를 포맷한 후 이를 호출자에게 반환합니다.

#### 주제

- [AWS Lambda 함수 핸들러\(Ruby\) \(p. 323\)](#)
- [AWS Lambda 배포 패키지\(Ruby\) \(p. 324\)](#)
- [AWS Lambda 컨텍스트 객체\(Ruby\) \(p. 326\)](#)
- [AWS Lambda 함수 로깅\(Ruby\) \(p. 326\)](#)
- [AWS Lambda 함수 오류\(Ruby\) \(p. 327\)](#)

## AWS Lambda 함수 핸들러(Ruby)

Lambda 함수의 핸들러는 함수가 호출될 때 Lambda이 호출하는 메서드입니다. 다음 예제에서 파일 `function.rb`는 `handler`라는 이름의 핸들러 메서드를 정의합니다. 핸들러 함수는 2개의 객체를 입력으로 사용하며 JSON 문서를 반환합니다.

#### Example function.rb

```
require 'json'

def handler(event:, context:)
  { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

함수 구성에서 `handler` 설정은 핸들러를 찾을 위치를 Lambda에 알려줍니다. 앞의 예제에서 이 설정의 올바른 값은 `function.handler`입니다. 여기에는 점으로 구분된 2개의 이름 즉, 파일 이름과 핸들러 메서드의 이름이 포함됩니다.

하나의 클래스에서 핸들러 메서드를 정의할 수도 있습니다. 다음 예제에서는 `LambdaFunctions`라는 이름의 모듈에서 `Handler`라는 이름의 클래스에 `process`라는 이름의 핸들러 메서드를 정의합니다.

#### Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:, context:)
      "Hello!"
    end
  end
end
```

```
    end
  end
end
```

이 경우, 핸들러 설정은 `source.LambdaFunctions::Handler.process`입니다.

핸들러가 허용하는 2개의 객체로는 호출 이벤트 및 컨텍스트가 있습니다. 해당 이벤트는 호출자가 제공하는 페이로드가 포함된 Ruby 객체입니다. 페이로드가 JSON 문서인 경우, 이벤트 객체는 Ruby 해시입니다. 그렇지 않은 경우, 이 객체는 문자열입니다. [컨텍스트 객체 \(p. 326\)](#)에는 호출, 함수 및 실행 환경에 관한 정보를 제공하는 메서드와 속성이 있습니다.

함수 핸들러는 Lambda 함수가 호출될 때마다 실행됩니다. 핸들러 외부의 정적 코드는 함수의 인스턴스당 한번씩 실행됩니다. 핸들러가 SDK 클라이언트 및 데이터베이스 연결과 같은 리소스를 사용하는 경우, 핸들러 메서드 외부에서 그러한 리소스를 생성하면 다중 호출 시 이 리소스를 다시 사용할 수 있습니다.

함수의 각 인스턴스는 다중 호출 이벤트를 처리할 수 있으며 다만 이벤트를 한 번에 하나씩만 처리합니다. 주어진 시간에 하나의 이벤트를 처리하는 인스턴스의 수는 함수의 동시성을 나타냅니다. Lambda 실행 컨텍스트에 관한 자세한 내용은 [AWS Lambda 실행 컨텍스트 \(p. 101\)](#) 단원을 참조하십시오.

## AWS Lambda 배포 패키지(Ruby)

배포 패키지는 함수 코드 및 종속성이 포함되어 있는 ZIP 아카이브 파일입니다. Lambda API를 사용하여 함수를 관리하거나 코드에서 AWS SDK 이외의 라이브러리를 사용하는 경우 배포 패키지를 생성해야 합니다. 다른 라이브러리 및 종속성을 배포 패키지에 포함해야 합니다. 패키지를 Lambda에 직접 업로드하거나 Amazon S3 버킷을 사용하여 Lambda에 업로드할 수 있습니다.

Lambda 콘솔 편집기 ([p. 24](#))를 사용하여 함수를 작성하는 경우 콘솔에서 배포 패키지를 관리합니다. 라이브러리를 추가할 필요가 없는 한 이 방법을 사용할 수 있습니다. 또한 총 크기가 3 MB를 초과하지 않는 한, 이 방법을 사용하여 이미 배포 패키지에 라이브러리가 있는 함수를 업데이트할 수 있습니다.

### 섹션

- [종속 프로그램이 없는 함수를 업데이트 \(p. 324\)](#)
- [추가 종속 프로그램이 있는 함수를 업데이트 \(p. 325\)](#)

## 종속 프로그램이 없는 함수를 업데이트

Lambda API를 사용하여 함수를 생성하거나 업데이트하려면 함수 코드를 포함하는 아카이브를 생성한 후 AWS CLI를 사용하여 이 아카이브를 업로드합니다.

종속 프로그램이 없는 Ruby 함수를 업데이트하려면

1. ZIP 아카이브를 생성합니다.

```
~/my-function$ zip function.zip function.rb
```

2. `update-function-code` 명령을 사용하여 패키지를 업로드합니다.

```
~/my-function$ aws lambda update-function-code --function-name ruby25 --zip-file
fileb://function.zip
{
  "FunctionName": "ruby25",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:ruby25",
  "Runtime": "ruby2.5",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
```

```
"CodeSize": 300,  
"Description": "",  
"Timeout": 3,  
"MemorySize": 128,  
"LastModified": "2018-11-23T21:00:10.248+0000",  
"CodeSha256": "Qf0haXm3JtGTmcDbjMc1I2di6YFMi9niEuiYonYptAk=",  
"Version": "$LATEST",  
"TracingConfig": {  
    "Mode": "Active"  
},  
"RevisionId": "d1e983e3-ca8e-434b-8dc1-7add83d72ebd"  
}
```

## 추가 종속 프로그램이 있는 함수를 업데이트

함수가 Ruby용 AWS SDK 외의 다른 라이브러리를 사용할 경우, [Bundler](#)를 사용하여 로컬 디렉터리에 해당 라이브러리를 설치한 후 배포 패키지에 포함시키십시오.

종속 프로그램이 있는 Ruby 함수를 업데이트하려면

- bundle 명령을 사용하여 벤더 디렉터리에 라이브러리를 설치합니다.

```
~/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  
...
```

--path는 시스템 위치가 아닌 프로젝트 디렉터리에 Gem을 설치하며, 이 디렉터리를 차후 설치를 위한 기본 경로로 설정합니다. 차후에 Gem을 전역으로 설치하려면 --system 옵션을 사용하십시오.

- ZIP 아카이브를 생성합니다.

```
package$ zip -r function.zip function.rb vendor  
adding: function.rb (deflated 37%)  
adding: vendor/ (stored 0%)  
adding: vendor/bundle/ (stored 0%)  
adding: vendor/bundle/ruby/ (stored 0%)  
adding: vendor/bundle/ruby/2.5.0/ (stored 0%)  
adding: vendor/bundle/ruby/2.5.0/build_info/ (stored 0%)  
adding: vendor/bundle/ruby/2.5.0/cache/ (stored 0%)  
adding: vendor/bundle/ruby/2.5.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)  
...
```

- 함수 코드를 업데이트합니다.

```
~/my-function$ aws lambda update-function-code --function-name ruby25 --zip-file  
file:///function.zip  
{  
    "FunctionName": "ruby25",  
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:ruby25",  
    "Runtime": "ruby2.5",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Handler": "function.handler",  
    "CodeSize": 998918,  
    "Description": "",  
    "Timeout": 3,  
    "MemorySize": 128,  
    "LastModified": "2018-11-20T20:51:35.871+0000",  
}
```

```
"CodeSha256": "fJ3TxYnFosnnpN483dz9/rTzcXrbOiuu4iOZx34nXZI=",
"Version": "$LATEST",
"VpcConfig": {
    "SubnetIds": [],
    "SecurityGroupIds": [],
    "VpcId": ""
},
"TracingConfig": {
    "Mode": "Active"
},
"RevisionId": "9ca7c45b-bcda-4e51-ab5f-7c42fa916e39"
}
```

## AWS Lambda 컨텍스트 객체(Ruby)

Lambda는 함수를 실행할 때 컨텍스트 객체를 [핸들러 \(p. 323\)](#)로 전달합니다. 이 객체는 호출, 함수 및 실행 환경에 관한 정보를 제공하는 메서드 및 속성들을 제공합니다.

### 컨텍스트 메서드

- `get_remaining_time_in_millis` – 실행 시간이 초과되기 전에 남은 시간(밀리초)을 반환합니다.

### 컨텍스트 속성

- `function_name` – Lambda 함수의 이름
- `function_version` – 함수의 [버전 \(p. 45\)](#)
- `invoked_function_arn` – 함수를 호출할 때 사용하는 Amazon 리소스 이름(ARN) 호출자가 버전 번호 또는 별칭을 지정했는지 여부를 나타냅니다.
- `memory_limit_in_mb` – 함수에 구성된 메모리 양
- `aws_request_id` – 호출 요청의 식별자
- `log_group_name` – 함수에 대한 로그 그룹
- `log_stream_name` – 함수 인스턴스에 대한 로그 스트림
- `deadline_ms` – 실행 시간이 초과된 날짜를 Unix 시간 형식에 따른 밀리초 단위로 나타낸 값
- `identity` – (모바일 앱) 요청을 승인한 Amazon Cognito 자격 증명에 대한 정보
- `client_context` – (모바일 앱) 클라이언트 애플리케이션이 Lambda 호출자에게 제공한 클라이언트 컨텍스트

## AWS Lambda 함수 로깅(Ruby)

Lambda 함수는 CloudWatch Logs 로그 그룹과 함께 제공되며, 함수의 각 인스턴스에 대한 로그 스트림을 포함합니다. 런타임은 각 호출에 관한 세부 정보를 로그 스트림에 전송하며, 함수 코드에서 로그 및 그 외 출력을 중계합니다.

함수 코드에서 로그를 출력하려는 경우, `puts` 명령문을 사용하거나 `stdout` 또는 `stderr`에 쓰는 로깅 라이브러리를 사용할 수 있습니다. 다음 예제는 환경 변수의 값과 이벤트 객체를 로깅합니다.

### 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
```

함수 구성 페이지에서 함수를 테스트할 때 Lambda 콘솔에 로그 출력이 표시됩니다. 모든 출력에 대한 로그를 보려면 CloudWatch Logs 콘솔을 사용합니다.

### Lambda 함수의 로그를 보려면

1. [CloudWatch 콘솔의 로그 페이지](#)를 엽니다.
2. 함수(/aws/lambda/**function-name**)에 대한 로그 그룹을 선택합니다.
3. 목록에서 첫 번째 스트림을 선택합니다.

각 로그 스트림은 [함수의 인스턴스 \(p. 101\)](#)에 해당합니다. 새 스트림은 함수를 업데이트할 때, 그리고 여러 동시 호출을 처리하기 위해 추가 인스턴스가 생성될 때 나타납니다. 특정 호출에 대한 로그를 찾으려면 X-Ray로 함수를 계측하고 추적의 요청 및 로그 스트림에 대한 세부 정보를 기록합니다. 로그 및 추적을 X-Ray와 상호 연관시키는 샘플 애플리케이션의 경우 [AWS Lambda용 오류 처리자 샘플 애플리케이션 \(p. 116\)](#) 단원을 참조하십시오.

명령줄에서 호출에 대한 로그를 가져오려면 --log-type 옵션을 사용하십시오. 호출에서 base64로 인코딩된 로그를 최대 4KB까지 포함하는 LogResult 필드가 응답에 포함됩니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

base64 유ти리티를 사용하면 로그를 디코딩할 수 있습니다.

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Version: $LATEST
Processing event...
END RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d
REPORT RequestId: 8e827ab1-f155-11e8-b06d-018ab046158d Duration: 29.40 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 19 MB
```

base64는 Linux, macOS 및 [Ubuntu on Windows](#)에서 사용할 수 있습니다. macOS의 경우, 명령은 base64 -D입니다.

함수를 삭제해도 로그 그룹이 자동으로 삭제되지 않습니다. 로그를 무기한 저장하지 않으려면 로그 그룹을 삭제하거나 로그가 자동으로 삭제되는 [보존 기간을 구성](#)하십시오.

## AWS Lambda 함수 오류(Ruby)

코드에서 오류가 발생하면 Lambda는 해당 오류의 JSON 표현을 생성합니다. 이 오류 문서는 호출 로그에 표시되며 동기 호출 시 출력에 표시됩니다.

### Example function.rb

```
def handler(event:, context:)
```

```
    puts "Processing event..."  
    [1, 2, 3].first("two")  
    "Success"  
end
```

이 코드에서는 형식 오류가 발생합니다. Lambda은 오류를 포착하고 오류 메시지, 유형 및 스택 추적에 대한 필드가 있는 JSON 문서를 생성합니다.

```
{  
  "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'"  
  ]  
}
```

명령줄에서 함수를 호출하면 상태 코드는 변경되지 않지만 응답에서는 `FunctionError` 필드가 포함되며 출력에는 오류 문서가 포함됩니다.

```
$ aws lambda invoke --function-name ruby-function out  
{  
  "StatusCode": 200,  
  "FunctionError": "Unhandled",  
  "ExecutedVersion": "$LATEST"  
}  
$ cat out  
{"errorMessage": "no implicit conversion of String into Integer",  
 "errorType": "Function<TypeError>",  
 "stackTrace": ["/var/task/function.rb:3:in `first'",  
 "/var/task/function.rb:3:in `handler'"]}
```

오류 로그에서 오류를 보려면 `--log-type` 옵션을 사용하고 응답에서 base64 문자열을 디코딩하십시오.

```
$ aws lambda invoke --function-name ruby-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d  
START RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Version: $LATEST  
Processing event...  
Error raised from handler method  
{  
  "errorMessage": "no implicit conversion of String into Integer",  
  "errorType": "Function<TypeError>",  
  "stackTrace": [  
    "/var/task/function.rb:3:in `first'",  
    "/var/task/function.rb:3:in `handler'"  
  ]  
}  
END RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3  
REPORT RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Duration: 22.74 ms      Billed  
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 18 MB
```

자세한 내용은 [AWS Lambda 함수 로깅\(Ruby\) \(p. 326\)](#) 단원을 참조하십시오.

# API 참조

이 단원은 AWS Lambda API 참조 문서를 포함합니다. API 호출 시 서명을 제공하여 요청을 인증해야 합니다. AWS Lambda는 서명 버전 4를 지원합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [서명 버전 4 서명 프로세스](#) 단원을 참조하십시오.

서비스에 대한 개요는 [AWS Lambda란 무엇입니까?](#) (p. 1) 단원을 참조하십시오.

AWS CLI를 사용하여 AWS Lambda API를 탐색할 수 있습니다. 이 가이드는 를 사용하는 몇 가지 자습서를 제공합니다.

주제

- [Actions](#) (p. 329)
- [Data Types](#) (p. 462)

## Actions

The following actions are supported:

- [AddLayerVersionPermission](#) (p. 331)
- [AddPermission](#) (p. 335)
- [CreateAlias](#) (p. 339)
- [CreateEventSourceMapping](#) (p. 343)
- [CreateFunction](#) (p. 347)
- [DeleteAlias](#) (p. 355)
- [DeleteEventSourceMapping](#) (p. 357)
- [DeleteFunction](#) (p. 360)
- [DeleteFunctionConcurrency](#) (p. 362)
- [DeleteLayerVersion](#) (p. 364)
- [GetAccountSettings](#) (p. 366)
- [GetAlias](#) (p. 368)
- [GetEventSourceMapping](#) (p. 371)
- [GetFunction](#) (p. 374)
- [GetFunctionConfiguration](#) (p. 377)
- [GetLayerVersion](#) (p. 382)
- [GetLayerVersionByArn](#) (p. 385)
- [GetLayerVersionPolicy](#) (p. 388)
- [GetPolicy](#) (p. 390)
- [Invoke](#) (p. 392)
- [InvokeAsync](#) (p. 397)
- [ListAliases](#) (p. 399)
- [ListEventSourceMappings](#) (p. 402)
- [ListFunctions](#) (p. 405)
- [ListLayers](#) (p. 408)
- [ListLayerVersions](#) (p. 410)

- [ListTags \(p. 413\)](#)
- [ListVersionsByFunction \(p. 415\)](#)
- [PublishLayerVersion \(p. 418\)](#)
- [PublishVersion \(p. 422\)](#)
- [PutFunctionConcurrency \(p. 428\)](#)
- [RemoveLayerVersionPermission \(p. 431\)](#)
- [RemovePermission \(p. 433\)](#)
- [TagResource \(p. 436\)](#)
- [UntagResource \(p. 438\)](#)
- [UpdateAlias \(p. 440\)](#)
- [UpdateEventSourceMapping \(p. 444\)](#)
- [UpdateFunctionCode \(p. 448\)](#)
- [UpdateFunctionConfiguration \(p. 455\)](#)

## 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 \(p. 431\)](#) 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 \(p. 331\)](#)

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 \(p. 331\)](#)

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 \(p. 331\)](#)

The version number.

### Request Body

The request accepts the following data in JSON format.

#### [Action \(p. 331\)](#)

The API action that grants access to the layer. For example, lambda:GetLayerVersion.

Type: String

Pattern: lambda:GetLayerVersion

Required: Yes

### [OrganizationId \(p. 331\)](#)

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 \(p. 331\)](#)

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 \(p. 331\)](#)

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 \(p. 332\)](#)

A unique identifier for the current revision of the policy.

Type: String

### [Statement \(p. 332\)](#)

The permission statement.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### PolicyLengthExceededException

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.

HTTP Status Code: 409

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)

- AWS SDK for Ruby V2

## AddPermission

Grants an AWS service or another account permission to use a function. You can apply the policy at the function level, or specify a qualifier to restrict access to a single version or alias. If you use a qualifier, the invoker must use the full Amazon Resource Name (ARN) of that version or alias to invoke the function.

To grant permission to another account, specify the account ID as the `Principal`. For AWS services, the principal is a domain-style identifier defined by the service, like `s3.amazonaws.com` or `sns.amazonaws.com`. For AWS services, you can also specify the ARN or owning account of the associated resource as the `SourceArn` or `SourceAccount`. If you grant permission to a service principal without specifying the source, other accounts could potentially configure resources in their account to invoke your Lambda function.

This action adds a statement to a resource-based permission policy for the function. For more information about function policies, see [Lambda Function Policies](#).

## Request Syntax

```
POST /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "Action": "string",
  "EventSourceToken": "string",
  "Principal": "string",
  "RevisionId": "string",
  "SourceAccount": "string",
  "SourceArn": "string",
  "StatementId": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### FunctionName (p. 335)

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. 335)

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 \(p. 335\)](#)

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 \(p. 335\)](#)

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 \(p. 335\)](#)

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 \(p. 335\)](#)

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 \(p. 335\)](#)

For AWS services, the ID of the account that owns the resource. Use this instead of `SourceArn` to grant permission to resources that are owned by another account (for example, all of an account's Amazon S3 buckets). Or use it together with `SourceArn` to ensure that the resource is owned by the specified account. For example, an Amazon S3 bucket could be deleted by its owner and recreated by another account.

Type: String

Pattern: \d{12}

Required: No

### [SourceArn \(p. 335\)](#)

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 \(p. 335\)](#)

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 \(p. 337\)](#)

The permission statement that's added to the function policy.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### PolicyLengthExceededException

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.

HTTP Status Code: 409  
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404  
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateAlias

Creates an [alias](#) for a Lambda function version. Use aliases to provide clients with a function identifier that you can update to invoke a different version.

You can also map an alias to split invocation requests between two versions. Use the `RoutingConfig` parameter to specify a second version and the percentage of invocation requests that it receives.

### Request Syntax

```
POST /2015-03-31/functions/FunctionName/aliases HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### [FunctionName](#) (p. 339)

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. 339)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 339\)](#)

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 \(p. 339\)](#)

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 \(p. 339\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) 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 \(p. 340\)](#)

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 \(p. 340\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 340\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 340\)](#)

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 \(p. 340\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 340\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) object

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

- HTTP Status Code: 404  
ServiceException
  - The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500  
TooManyRequestsException
  - Request throughput limit exceeded.
- HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateEventSourceMapping

Creates a mapping between an event source and an AWS Lambda function. Lambda reads items from the event source and triggers the function.

For details about each event source type, see the following topics.

- [Using AWS Lambda with Amazon Kinesis](#)
- [Using AWS Lambda with Amazon SQS](#)
- [Using AWS Lambda with Amazon DynamoDB](#)

### Request Syntax

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "Enabled": boolean,
    "EventSourceArn": "string",
    "FunctionName": "string",
    "StartingPosition": "string",
    "StartingPositionTimestamp": number
}
```

### URI Request Parameters

The request does not use any URI parameters.

### Request Body

The request accepts the following data in JSON format.

#### [BatchSize \(p. 343\)](#)

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

#### [Enabled \(p. 343\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

#### [EventSourceArn \(p. 343\)](#)

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 \(p. 343\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Version or Alias ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]\*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-\_]+)(:(\\$LATEST|[a-zA-Z0-9-\_]+))?

Required: Yes

#### [StartingPosition \(p. 343\)](#)

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 \(p. 343\)](#)

With StartingPosition set to AT\_TIMESTAMP, the time from which to start reading, in Unix time seconds.

Type: Timestamp

Required: No

## Response Syntax

```
HTTP/1.1 202
Content-type: application/json
```

```
{  
    "BatchSize": number,  
    "EventSourceArn": "string",  
    "FunctionArn": "string",  
    "LastModified": number,  
    "LastProcessingResult": "string",  
    "State": "string",  
    "StateTransitionReason": "string",  
    "UUID": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

### [BatchSize \(p. 344\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

### [EventSourceArn \(p. 344\)](#)

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 \(p. 344\)](#)

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 \(p. 344\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

### [LastProcessingResult \(p. 344\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

### [State \(p. 344\)](#)

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 \(p. 344\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String  
[UUID \(p. 344\)](#)

The identifier of the event source mapping.

Type: String

## Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateFunction

Creates a Lambda function. To create a function, you need a [deployment package](#) and an [execution role](#). The deployment package contains your function code. The execution role grants the function permission to use AWS services, such as Amazon CloudWatch Logs for log streaming and AWS X-Ray for request tracing.

A function has an unpublished version, and can have published versions and aliases. 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 \(p. 455\)](#). Function-level settings apply to both the unpublished and published versions of the function, and include tags ([TagResource \(p. 436\)](#)) and per-function concurrency limits ([PutFunctionConcurrency \(p. 428\)](#)).

If another account or an AWS service invokes your function, use [AddPermission \(p. 335\)](#) 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 \(p. 392\)](#). To invoke your function in response to events in other AWS services, create an event source mapping ([CreateEventSourceMapping \(p. 343\)](#)), or configure a function trigger in the other service. For more information, see [Invoking Functions](#).

## Request Syntax

```
POST /2015-03-31/functions HTTP/1.1
Content-type: application/json

{
  "Code": {
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string",
    "ZipFile": blob
  },
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string" : "string"
    }
  },
  "FunctionName": "string",
  "Handler": "string",
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "Publish": boolean,
  "Role": "string",
  "Runtime": "string",
  "Tags": {
    "string" : "string"
  },
  "Timeout": number,
  "TracingConfig": {
    "Mode": "string"
  }
}
```

```
    },
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ]
    }
}
```

## URI Request Parameters

The request does not use any URI parameters.

## Request Body

The request accepts the following data in JSON format.

### [Code \(p. 347\)](#)

The code for the function.

Type: [FunctionCode \(p. 475\)](#) object

Required: Yes

### [DeadLetterConfig \(p. 347\)](#)

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 \(p. 469\)](#) object

Required: No

### [Description \(p. 347\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

### [Environment \(p. 347\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 470\)](#) object

Required: No

### [FunctionName \(p. 347\)](#)

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 \(p. 347\)](#)

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 \(p. 347\)](#)

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 \(p. 347\)](#)

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 \(p. 347\)](#)

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 \(p. 347\)](#)

Set to true to publish the first version of the function during creation.

Type: Boolean

Required: No

### [Role \(p. 347\)](#)

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 \(p. 347\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

Required: Yes

### [Tags \(p. 347\)](#)

A list of [tags](#) to apply to the function.

Type: String to string map

Required: No

### [Timeout \(p. 347\)](#)

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 \(p. 347\)](#)

Set `Mode` to `Active` to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 487\)](#) object

Required: No

### [VpcConfig \(p. 347\)](#)

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 \(p. 489\)](#) object

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```

```
{  
    "CodeSha256": "string",  
    "CodeSize": number,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    },  
    "Description": "string",  
    "Environment": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "Variables": {  
            "string" : "string"  
        }  
    },  
    "FunctionArn": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "KMSKeyArn": "string",  
    "LastModified": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "Timeout": number,  
    "TracingConfig": {  
        "Mode": "string"  
    },  
    "Version": "string",  
    "VpcConfig": {  
        "SecurityGroupIds": [ "string" ],  
        "SubnetIds": [ "string" ],  
        "VpcId": "string"  
    }  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### CodeSha256 (p. 350)

The SHA256 hash of the function's deployment package.

Type: String

### CodeSize (p. 350)

The size of the function's deployment package, in bytes.

Type: Long

### DeadLetterConfig (p. 350)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) object

[Description \(p. 350\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 350\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 472\)](#) object

[FunctionArn \(p. 350\)](#)

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 \(p. 350\)](#)

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 \(p. 350\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 350\)](#)

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 \(p. 350\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 350\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

[MasterArn \(p. 350\)](#)

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 \(p. 350\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 350\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 350\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]\*)?:iam::\d{12}:role/?[a-zA-Z\_0-9+=,.@\-/]+

[Runtime \(p. 350\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

[Timeout \(p. 350\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 350\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 488\)](#) object

[Version \(p. 350\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 350\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 490\)](#) object

## Errors

### CodeStorageExceeded**Exception**

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

### InvalidParameterValue**Exception**

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceConflict**Exception**

The resource already exists.

HTTP Status Code: 409

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### Service**Exception**

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequests**Exception**

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteAlias

Deletes a Lambda function [alias](#).

### Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [FunctionName](#) (p. 355)

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. 355)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-]+$)([a-zA-Z0-9-_]+)`

## Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### [InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

- HTTP Status Code: 400  
ServiceException
  - The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500  
TooManyRequestsException
  - Request throughput limit exceeded.
- HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteEventSourceMapping

Deletes an [event source mapping](#). You can get the identifier of a mapping from the output of [ListEventSourceMappings](#) (p. 402).

### 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. 357)

The identifier of the event source mapping.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

#### [BatchSize](#) (p. 357)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

#### [EventSourceArn](#) (p. 357)

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 \(p. 357\)](#)

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 \(p. 357\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 357\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 357\)](#)

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 \(p. 357\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 357\)](#)

The identifier of the event source mapping.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceInUseException

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

- HTTP Status Code: 404  
ServiceException
  - The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500  
TooManyRequestsException
  - Request throughput limit exceeded.
- HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteFunction

Deletes a Lambda function. To delete a specific function version, use the [Qualifier](#) parameter. Otherwise, all versions and aliases are deleted.

To delete Lambda event source mappings that invoke a function, use [DeleteEventSourceMapping \(p. 357\)](#). 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 \(p. 360\)](#)

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 \(p. 360\)](#)

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceConflictException

The resource already exists.

HTTP Status Code: 409

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteFunctionConcurrency

Removes a concurrent execution limit from a function.

### Request Syntax

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### FunctionName (p. 362)

The name of the Lambda function.

##### Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:(aws[a-zA-Z-]\*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-\_+]):(\(\$LATEST|[a-zA-Z0-9-\_]+\))?

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

#### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404  
ServiceException  
  
The AWS Lambda service encountered an internal error.  
  
HTTP Status Code: 500  
TooManyRequestsException  
  
Request throughput limit exceeded.  
  
HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteLayerVersion

Deletes a version of an AWS Lambda layer. Deleted versions can no longer be viewed or added to functions. To avoid breaking functions, a copy of the version remains in Lambda until no functions refer to it.

### Request Syntax

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### LayerName (p. 364)

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. 364)

The version number.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

#### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

## GetAccountSettings

Retrieves details about your account's [limits](#) and usage in 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. 366)

Limits that are related to concurrency and code storage.

Type: [AccountLimit](#) (p. 463) object

#### [AccountUsage](#) (p. 366)

The number of functions and amount of storage in use.

Type: [AccountUsage](#) (p. 464) object

## Errors

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException  
Request throughput limit exceeded.  
HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetAlias

Returns details about a Lambda function [alias](#).

### Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [FunctionName](#) (p. 368)

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. 368)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-\_]+)

## Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

}

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [AliasArn \(p. 368\)](#)

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 \(p. 368\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

### [FunctionVersion \(p. 368\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\$LATEST|[0-9]+)

### [Name \(p. 368\)](#)

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 \(p. 368\)](#)

A unique identifier that changes when you update the alias.

Type: String

### [RoutingConfig \(p. 368\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) object

## Errors

### InvalidOperationException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400  
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404  
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetEventSourceMapping

Returns details about an event source mapping. You can get the identifier of a mapping from the output of [ListEventSourceMappings \(p. 402\)](#).

### Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [UUID \(p. 371\)](#)

The identifier of the event source mapping.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

#### [BatchSize \(p. 371\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

#### [EventSourceArn \(p. 371\)](#)

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 \(p. 371\)](#)

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 \(p. 371\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 371\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 371\)](#)

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 \(p. 371\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 371\)](#)

The identifier of the event source mapping.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetFunction

Returns information about 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. 374)

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. 374)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-\_]+)

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Code": {
        "Location": "string",
        "RepositoryType": "string"
    },
    "Concurrency": {
        "ReservedConcurrentExecutions": number
    }
}
```

```
},
"Configuration": {
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
},
"Tags": {
    "string" : "string"
}
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### Code (p. 374)

The deployment package of the function or version.

Type: [FunctionCodeLocation \(p. 476\)](#) object

### Concurrency (p. 374)

The function's reserved concurrency.

Type: [Concurrency \(p. 468\)](#) object  
[Configuration \(p. 374\)](#)

The configuration of the function or version.

Type: [FunctionConfiguration \(p. 477\)](#) object  
[Tags \(p. 374\)](#)

The function's [tags](#).

Type: String to string map

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## 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 \(p. 455\)](#).

To get all of a function's details, including function-level settings, use [GetFunction \(p. 374\)](#).

### 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 \(p. 377\)](#)

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 \(p. 377\)](#)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-\_]+)

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
}
```

```
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [CodeSha256 \(p. 377\)](#)

The SHA256 hash of the function's deployment package.

Type: String

### [CodeSize \(p. 377\)](#)

The size of the function's deployment package, in bytes.

Type: Long

### [DeadLetterConfig \(p. 377\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) object

### [Description \(p. 377\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 377\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 472\)](#) object

[FunctionArn \(p. 377\)](#)

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 \(p. 377\)](#)

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 \(p. 377\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 377\)](#)

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 \(p. 377\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 377\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

[MasterArn \(p. 377\)](#)

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 \(p. 377\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 377\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 377\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]\*)?:iam::\d{12}:role/?[a-zA-Z\_0-9+=,.@/-/\_]+

[Runtime \(p. 377\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

[Timeout \(p. 377\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 377\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 488\)](#) object

[Version \(p. 377\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\$LATEST|[0-9]+)

[VpcConfig \(p. 377\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 490\)](#) object

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetLayerVersion

Returns information about a version of an [AWS Lambda layer](#), with a link to download the layer archive that's valid for 10 minutes.

### Request Syntax

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### LayerName (p. 382)

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. 382)

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 \(p. 382\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

[Content \(p. 382\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 484\)](#) object

[CreatedDate \(p. 382\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 382\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 382\)](#)

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 \(p. 382\)](#)

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 \(p. 382\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 382\)](#)

The version number.

Type: Long

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## 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. 385)

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. 385)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

#### [Content \(p. 385\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 484\)](#) object

#### [CreatedDate \(p. 385\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

#### [Description \(p. 385\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

#### [LayerArn \(p. 385\)](#)

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 \(p. 385\)](#)

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 \(p. 385\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

#### [Version \(p. 385\)](#)

The version number.

Type: Long

## Errors

### InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetLayerVersionPolicy

Returns the permission policy for a version of an AWS Lambda layer. For more information, see [AddLayerVersionPermission \(p. 331\)](#).

### 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 \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

The policy document.

Type: String

#### [RevisionId \(p. 388\)](#)

A unique identifier for the current revision of the policy.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetPolicy

Returns the resource-based IAM policy for a function, version, or alias.

### Request Syntax

```
GET /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### FunctionName (p. 390)

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. 390)

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 \(p. 390\)](#)

The resource-based policy.

Type: String

[RevisionId \(p. 390\)](#)

A unique identifier for the current revision of the policy.

Type: String

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## Invoke

Invokes a Lambda function. You can invoke a function synchronously (and wait for the response), or asynchronously. To invoke a function asynchronously, set `InvocationType` to `Event`.

For synchronous invocation, details about the function response, including errors, are included in the response body and headers. For either invocation type, you can find more information in the [execution log](#) and [trace](#). To record function errors for asynchronous invocations, configure your function with a [dead letter queue](#).

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](#).

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. 392)

Up to 3583 bytes of base64-encoded data about the invoking client to pass to the function in the `context` object.

### [FunctionName](#) (p. 392)

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 \(p. 392\)](#)

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 \(p. 392\)](#)

Set to Tail to include the execution log in the response.

Valid Values: None | Tail

#### [Qualifier \(p. 392\)](#)

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 \(p. 392\)](#)

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 \(p. 393\)](#)

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 \(p. 393\)](#)

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 \(p. 393\)](#)

If present, indicates that an error occurred during function execution. Details about the error are included in the response payload.

- **Handled** - The runtime caught an error thrown by the function and formatted it into a JSON document.
- **Unhandled** - The runtime didn't handle the error. For example, the function ran out of memory or timed out.

#### [LogResult \(p. 393\)](#)

The last 4 KB of the execution log, which is base64 encoded.

The response returns the following as the HTTP body.

#### [Payload \(p. 393\)](#)

The response from the function, or an error object.

## Errors

### EC2AccessDeniedException

Need additional permissions to configure VPC settings.

HTTP Status Code: 502

### EC2ThrottledException

AWS Lambda was throttled by Amazon EC2 during Lambda function initialization using the execution role provided for the Lambda function.

HTTP Status Code: 502

### EC2UnexpectedException

AWS Lambda received an unexpected EC2 client exception while setting up for the Lambda function.

HTTP Status Code: 502

### ENILimitReachedException

AWS Lambda was not able to create an Elastic Network Interface (ENI) in the VPC, specified as part of Lambda function configuration, because the limit for network interfaces has been reached.

HTTP Status Code: 502

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

InvalidRuntimeException

The runtime or runtime version specified is not supported.

HTTP Status Code: 502

InvalidSecurityGroupIDException

The Security Group ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidSubnetIDException

The Subnet ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidZipFileException

AWS Lambda could not unzip the deployment package.

HTTP Status Code: 502

KMSAccessDeniedException

Lambda was unable to decrypt the environment variables because KMS access was denied. Check the Lambda function's KMS permissions.

HTTP Status Code: 502

KMSDisabledException

Lambda was unable to decrypt the environment variables because the KMS key used is disabled. Check the Lambda function's KMS key settings.

HTTP Status Code: 502

KMSInvalidStateException

Lambda was unable to decrypt the environment variables because the KMS key used is in an invalid state for Decrypt. Check the function's KMS key settings.

HTTP Status Code: 502

KMSNotFoundException

Lambda was unable to decrypt the environment variables because the KMS key was not found. Check the function's KMS key settings.

HTTP Status Code: 502

RequestTooLargeException

The request payload exceeded the `Invoke` request body JSON input limit. For more information, see [Limits](#).

HTTP Status Code: 413

#### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

#### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

#### SubnetIPAddressLimitReachedException

AWS Lambda was not able to set up VPC access for the Lambda function because one or more configured subnets has no available IP addresses.

HTTP Status Code: 502

#### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

#### UnsupportedMediaTypeException

The content type of the `Invoke` request body is not JSON.

HTTP Status Code: 415

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## InvokeAsync

This action has been deprecated.

### Important

For asynchronous function invocation, use [Invoke \(p. 392\)](#).

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. 397)

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. 397)

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 \(p. 397\)](#)

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

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListAliases

Returns a list of [aliases](#) for a Lambda function.

### Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases?  
FunctionVersion=FunctionVersion&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [FunctionName](#) (p. 399)

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. 399)

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. 399)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

#### [MaxItems](#) (p. 399)

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 \(p. 399\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 465\)](#) objects

### [NextMarker \(p. 399\)](#)

The pagination token that's included if more results are available.

Type: String

## Errors

### [InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### [ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### [ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### [TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListEventSourceMappings

Lists event source mappings. Specify an `EventSourceArn` to only show event source mappings for a single event source.

### Request Syntax

```
GET /2015-03-31/event-source-mappings/?  
EventSourceArn=EventSourceArn&FunctionName=FunctionName&Marker=Marker&MaxItems=MaxItems  
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

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

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 \(p. 402\)](#)

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 \(p. 402\)](#)

A pagination token returned by a previous call.

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

The maximum number of event source mappings to return.

Valid Range: Minimum value of 1. Maximum value of 10000.

### Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "EventSourceMappings": [
    {
      "BatchSize": number,
      "EventSourceArn": "string",
      "FunctionArn": "string",
      "LastModified": number,
      "LastProcessingResult": "string",
      "State": "string",
      "StateTransitionReason": "string",
      "UUID": "string"
    }
  ],
  "NextMarker": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### EventSourceMappings (p. 403)

A list of event source mappings.

Type: Array of [EventSourceMappingConfiguration \(p. 473\)](#) objects

### NextMarker (p. 403)

A pagination token that's returned when the response doesn't contain all event source mappings.

Type: String

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

## TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListFunctions

Returns a list of Lambda functions, with the version-specific configuration of each.

Set `FunctionVersion` to `ALL` to include all published versions of each function in addition to the unpublished version. To get more information about a function or version, use [GetFunction \(p. 374\)](#).

### 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 \(p. 405\)](#)

Set to `ALL` to include entries for all published versions of each function.

Valid Values: `ALL`

#### [Marker \(p. 405\)](#)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

#### [MasterRegion \(p. 405\)](#)

For Lambda@Edge functions, the AWS Region of the master function. For example, `us-east-2` or `ALL`. If specified, you must set `FunctionVersion` to `ALL`.

Pattern: `ALL | [a-z]{2}(-gov)?-[a-z]+-\d{1}`

#### [MaxItems \(p. 405\)](#)

Specify a value between 1 and 50 to limit the number of functions in the response.

Valid Range: Minimum value of 1. Maximum value of 10000.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "Functions": [  
    {  
      "CodeSha256": "string",  
      "CodeSize": number,  
      "DeadLetterConfig": {  
        "TargetArn": "string"  
      },  
      "Description": "string",  
      "Environment": {  
        "Error": {  
          "Code": "string",  
          "Message": "string"  
        }  
      },  
      "FunctionArn": "string",  
      "Handler": "string",  
      "LastModified": "string",  
      "MemorySize": number,  
      "Runtime": "string",  
      "State": "string",  
      "Timeout": number  
    }  
  ]  
}
```

```
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
],
"NextMarker": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [Functions \(p. 405\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 477\)](#) objects

### [NextMarker \(p. 405\)](#)

The pagination token that's included if more results are available.

Type: String

## Errors

### InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400  
ServiceException  
  
The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException  
  
Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListLayers

Lists AWS Lambda layers and shows information about the latest version of each. Specify a runtime identifier to list only layers that indicate that they're compatible with that runtime.

### Request Syntax

```
GET /2018-10-31/layers?CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### CompatibleRuntime (p. 408)

A runtime identifier. For example, go1.x.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

#### Marker (p. 408)

A pagination token returned by a previous call.

#### MaxItems (p. 408)

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 \(p. 408\)](#)

A list of function layers.

Type: Array of [LayersListItem \(p. 482\)](#) objects

### [NextMarker \(p. 408\)](#)

A pagination token returned when the response doesn't contain all layers.

Type: String

## Errors

### [InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### [ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### [TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListLayerVersions

Lists the versions of an [AWS Lambda layer](#). Versions that have been deleted aren't listed. Specify a [runtime identifier](#) to list only versions that indicate that they're compatible with that runtime.

### Request Syntax

```
GET /2018-10-31/layers/LayerName/versions?  
CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### CompatibleRuntime (p. 410)

A runtime identifier. For example, go1.x.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

#### LayerName (p. 410)

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. 410)

A pagination token returned by a previous call.

#### MaxItems (p. 410)

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 \(p. 410\)](#)

A list of versions.

Type: Array of [LayerVersionsListItem \(p. 485\)](#) objects

### [NextMarker \(p. 410\)](#)

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### [ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### [ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### [TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

## ListTags

Returns a function's [tags](#). You can also view tags with [GetFunction](#) (p. 374).

### Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [Resource](#) (p. 413)

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. 413)

The function's tags.

Type: String to string map

### Errors

#### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the [CreateFunction](#) or the [UpdateFunctionConfiguration](#) API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## ListVersionsByFunction

Returns a list of [versions](#), with the version-specific configuration of each.

### Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### FunctionName (p. 415)

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. 415)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

#### MaxItems (p. 415)

Limit the number of versions that are returned.

Valid Range: Minimum value of 1. Maximum value of 10000.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "NextMarker": "string",
    "Versions": [
        {
            "CodeSha256": "string",
            "CodeSize": number,
            "DeadLetterConfig": {
                "TargetArn": "string"
            },
            "Description": "string",
            "Environment": {
```

```
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
]
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [NextMarker \(p. 415\)](#)

The pagination token that's included if more results are available.

Type: String

### [Versions \(p. 415\)](#)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration \(p. 477\)](#) objects

## Errors

### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PublishLayerVersion

Creates an [AWS Lambda layer](#) from a ZIP archive. Each time you call `PublishLayerVersion` with the same version name, a new version is created.

Add layers to your function with [CreateFunction \(p. 347\)](#) or [UpdateFunctionConfiguration \(p. 455\)](#).

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

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 \(p. 418\)](#)

A list of compatible [function runtimes](#). Used for filtering with [ListLayers \(p. 408\)](#) and [ListLayerVersions \(p. 410\)](#).

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: `nodejs8.10` | `nodejs10.x` | `java8` | `python2.7` | `python3.6` | `python3.7` | `dotnetcore1.0` | `dotnetcore2.1` | `go1.x` | `ruby2.5` | `provided`

Required: No

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

The function layer archive.

Type: [LayerVersionContentInput \(p. 483\)](#) object

Required: Yes

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

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

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

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 \(p. 419\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided  
[Content \(p. 419\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 484\)](#) object

[CreatedDate \(p. 419\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 419\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 419\)](#)

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 \(p. 419\)](#)

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 \(p. 419\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 419\)](#)

The version number.

Type: Long

## Errors

[CodeStorageExceededException](#)

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

#### InvalidOperationException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

#### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

#### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

#### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PublishVersion

Creates a [version](#) from the current code and configuration of a function. Use versions to create a snapshot of your function code and configuration that doesn't change.

AWS Lambda doesn't publish a version if the function's configuration and code haven't changed since the last version. Use [UpdateFunctionCode](#) (p. 448) or [UpdateFunctionConfiguration](#) (p. 455) to update the function before publishing a version.

Clients can invoke versions directly or with an alias. To create an alias, use [CreateAlias](#) (p. 339).

### 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. 422)

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. 422)

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. 448).

Type: String

Required: No

### Description (p. 422)

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. 422)

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",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ]
    }
}
```

```
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### [CodeSha256 \(p. 423\)](#)

The SHA256 hash of the function's deployment package.

Type: String

### [CodeSize \(p. 423\)](#)

The size of the function's deployment package, in bytes.

Type: Long

### [DeadLetterConfig \(p. 423\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) object

### [Description \(p. 423\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

### [Environment \(p. 423\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 472\)](#) object

### [FunctionArn \(p. 423\)](#)

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 \(p. 423\)](#)

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 \(p. 423\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [ ^\s ]+

[KMSKeyArn \(p. 423\)](#)

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 \(p. 423\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 423\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

[MasterArn \(p. 423\)](#)

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 \(p. 423\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 423\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 423\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]\*)?:iam::\d{12}:role/?[a-zA-Z\_0-9+=,.@\-\_/.+]

[Runtime \(p. 423\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

### [Timeout \(p. 423\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

### [TracingConfig \(p. 423\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 488\)](#) object

### [Version \(p. 423\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

### [VpcConfig \(p. 423\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 490\)](#) 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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

### ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutFunctionConcurrency

Sets the maximum number of simultaneous executions for a function, and reserves capacity for that concurrency level.

Concurrency settings apply to the function as a whole, including all published versions and the unpublished version. Reserving concurrency both ensures that your function has capacity to process the specified number of events simultaneously, and prevents it from scaling beyond that level. Use [GetFunction \(p. 374\)](#) to see the current setting for a function.

Use [GetAccountSettings \(p. 366\)](#) 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 \(p. 428\)](#)

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 \(p. 428\)](#)

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 \(p. 429\)](#)

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

### `ResourceNotFoundException`

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

### `ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

### `TooManyRequestsException`

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

## RemoveLayerVersionPermission

Removes a statement from the permissions policy for a version of an AWS Lambda layer. For more information, see [AddLayerVersionPermission \(p. 331\)](#).

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

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 \(p. 431\)](#)

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 \(p. 431\)](#)

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 \(p. 431\)](#)

The version number.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### [InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400  
PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412  
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404  
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## RemovePermission

Revokes function-use permission from an AWS service or another account. You can get the ID of the statement from the output of [GetPolicy](#) (p. 390).

### 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. 433)

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. 433)

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. 433)

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. 433)

Statement ID of the permission to remove.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-\_]+)

### Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)

- AWS SDK for Ruby V2

## TagResource

Adds [tags](#) to a function.

### Request Syntax

```
POST /2017-03-31/tags/ARN HTTP/1.1
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### Resource (p. 436)

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. 436)

A list of tags to apply to the function.

Type: String to string map

Required: Yes

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400  
ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404  
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## UntagResource

Removes [tags](#) from a function.

### Request Syntax

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### [Resource \(p. 438\)](#)

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 \(p. 438\)](#)

A list of tag keys to remove from the function.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### [InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

#### [ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

#### [ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500  
TooManyRequestsException  
Request throughput limit exceeded.  
HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## UpdateAlias

Updates the configuration of a Lambda function [alias](#).

### Request Syntax

```
PUT /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "RevisionId": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### [FunctionName](#) (p. 440)

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. 440)

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. 440)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 440\)](#)

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 \(p. 440\)](#)

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 \(p. 440\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) 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 \(p. 441\)](#)

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 \(p. 441\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 441\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 441\)](#)

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 \(p. 441\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 441\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) object

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[PreconditionFailedException](#)

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

- HTTP Status Code: 404  
ServiceException
  - The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500  
TooManyRequestsException
  - Request throughput limit exceeded.
- HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## UpdateEventSourceMapping

Updates an event source mapping. You can change the function that AWS Lambda invokes, or pause invocation and resume later from the same location.

### Request Syntax

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "Enabled": boolean,
    "FunctionName": "string"
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### [UUID \(p. 444\)](#)

The identifier of the event source mapping.

### Request Body

The request accepts the following data in JSON format.

#### [BatchSize \(p. 444\)](#)

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

#### [Enabled \(p. 444\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

#### [FunctionName \(p. 444\)](#)

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-zA-Z]{2}(-gov)?-[a-zA-Z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-\_]+)(:(\\$LATEST|[a-zA-Z0-9-\_]+))?

Required: No

## Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

### [BatchSize \(p. 445\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

### [EventSourceArn \(p. 445\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]\*):([a-zA-Z0-9\-])+:(a-zA-Z){2}(-gov)?-[a-zA-Z]+\d{1})?:(\d{12})?:(.\*?)

### [FunctionArn \(p. 445\)](#)

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 \(p. 445\)](#)

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 445\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 445\)](#)

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 \(p. 445\)](#)

The cause of the last state change, either User initiated or Lambda initiated.

Type: String

[UUID \(p. 445\)](#)

The identifier of the event source mapping.

Type: String

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists.

HTTP Status Code: 409

[ResourceInUseException](#)

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

#### ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

#### TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## UpdateFunctionCode

Updates a Lambda function's code.

The function's code is locked when you publish a version. You 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 \(p. 448\)](#)

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 \(p. 448\)](#)

Set to true to validate the request parameters and access permissions without modifying the function code.

Type: Boolean

Required: No

### [Publish \(p. 448\)](#)

Set to true to publish a new version of the function after updating the code. This has the same effect as calling [PublishVersion \(p. 422\)](#) separately.

Type: Boolean

Required: No

### [RevisionId \(p. 448\)](#)

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 \(p. 448\)](#)

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 \(p. 448\)](#)

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

### [S3ObjectVersion \(p. 448\)](#)

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 \(p. 448\)](#)

The base64-encoded contents of the deployment package. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```

```
{  
    "CodeSha256": "string",  
    "CodeSize": number,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    },  
    "Description": "string",  
    "Environment": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "Variables": {  
            "string" : "string"  
        }  
    },  
    "FunctionArn": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "KMSKeyArn": "string",  
    "LastModified": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "Timeout": number,  
    "TracingConfig": {  
        "Mode": "string"  
    },  
    "Version": "string",  
    "VpcConfig": {  
        "SecurityGroupIds": [ "string" ],  
        "SubnetIds": [ "string" ],  
        "VpcId": "string"  
    }  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### CodeSha256 (p. 449)

The SHA256 hash of the function's deployment package.

Type: String

### CodeSize (p. 449)

The size of the function's deployment package, in bytes.

Type: Long

### DeadLetterConfig (p. 449)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) object

[Description \(p. 449\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 449\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 472\)](#) object

[FunctionArn \(p. 449\)](#)

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 \(p. 449\)](#)

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 \(p. 449\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 449\)](#)

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 \(p. 449\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 449\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

[MasterArn \(p. 449\)](#)

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 \(p. 449\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 449\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 449\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]\*)?:iam::\d{12}:role/?[a-zA-Z\_0-9+=,.@\-/]+

[Runtime \(p. 449\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

[Timeout \(p. 449\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 449\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 488\)](#) object

[Version \(p. 449\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 449\)](#)

The function's networking configuration.

Type: [VpcConfigResponse](#) (p. 490) 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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PreconditionFailedException

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



# UpdateFunctionConfiguration

Modify the version-specific settings of a Lambda function.

These settings can vary between versions of a function and are locked when you publish a version. You can't modify the configuration of a published version, only the unpublished version.

To configure function concurrency, use [PutFunctionConcurrency \(p. 428\)](#). To grant invoke permissions to an account or AWS service, use [AddPermission \(p. 335\)](#).

## 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. 455)

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 \(p. 455\)](#)

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 \(p. 469\)](#) object

Required: No

### [Description \(p. 455\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

### [Environment \(p. 455\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 470\)](#) object

Required: No

### [Handler \(p. 455\)](#)

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 \(p. 455\)](#)

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 \(p. 455\)](#)

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 \(p. 455\)](#)

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 \(p. 455\)](#)

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 \(p. 455\)](#)

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 \(p. 455\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

Required: No

[Timeout \(p. 455\)](#)

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 \(p. 455\)](#)

Set `Mode` to `Active` to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 487\)](#) object

Required: No

### VpcConfig (p. 455)

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 \(p. 489\)](#) object

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 458\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 458\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 458\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) object

[Description \(p. 458\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 458\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 472\)](#) object

[FunctionArn \(p. 458\)](#)

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 \(p. 458\)](#)

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 \(p. 458\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 458\)](#)

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 \(p. 458\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Layers \(p. 458\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

[MasterArn \(p. 458\)](#)

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 \(p. 458\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 458\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 458\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\-/]+`

[Runtime \(p. 458\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

[Timeout \(p. 458\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 458\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 488\)](#) object

[Version \(p. 458\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (`\$LATEST` | [0-9]+)

[VpcConfig \(p. 458\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 490\)](#) object

## Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[PreconditionFailedException](#)

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412

[ResourceConflictException](#)

The resource already exists.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

Request throughput limit exceeded.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

## Data Types

The following data types are supported:

- AccountLimit (p. 463)
- AccountUsage (p. 464)
- AliasConfiguration (p. 465)
- AliasRoutingConfiguration (p. 467)
- Concurrency (p. 468)
- DeadLetterConfig (p. 469)
- Environment (p. 470)
- EnvironmentError (p. 471)
- EnvironmentResponse (p. 472)
- EventSourceMappingConfiguration (p. 473)
- FunctionCode (p. 475)
- FunctionCodeLocation (p. 476)
- FunctionConfiguration (p. 477)
- Layer (p. 481)
- LayersListItem (p. 482)
- LayerVersionContentInput (p. 483)
- LayerVersionContentOutput (p. 484)
- LayerVersionsListItem (p. 485)
- TracingConfig (p. 487)
- TracingConfigResponse (p. 488)
- VpcConfig (p. 489)
- VpcConfigResponse (p. 490)

## AccountLimit

Limits that are related to concurrency and code storage. All file and storage sizes are in bytes.

### Contents

#### CodeSizeUnzipped

The maximum size of your function's code and layers when 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. 428).

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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## AccountUsage

The number of functions and amount of storage in use.

### Contents

#### FunctionCount

The number of Lambda functions.

Type: Long

Required: No

#### TotalCodeSize

The amount of storage space, in bytes, 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## AliasConfiguration

Provides configuration information about a Lambda function [alias](#).

### Contents

#### AliasArn

The Amazon Resource Name (ARN) of the alias.

Type: String

Pattern: arn:(aws[a-zA-Z-]\*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-\_]+(:(\\$\\$LATEST|[a-zA-Z0-9-\_]+))?

Required: No

#### Description

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

#### FunctionVersion

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$\\$LATEST|[0-9]+)

Required: No

#### Name

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-\_]+)

Required: No

#### RevisionId

A unique identifier that changes when you update the alias.

Type: String

Required: No

#### RoutingConfig

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 467\)](#) 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## AliasRoutingConfiguration

The [traffic-shifting](#) configuration of a Lambda function alias.

### Contents

#### AdditionalVersionWeights

The name of the second alias, 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 ]<sup>+</sup>

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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## DeadLetterConfig

The [dead letter queue](#) for failed asynchronous invocations.

### Contents

#### TargetArn

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.*])|()`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# Environment

A function's environment variable settings.

## Contents

### Variables

Environment variable key-value pairs.

Type: String to string map

Key Pattern: [ a-zA-Z ]([ a-zA-Z0-9\_ ])+

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## EnvironmentResponse

The results of a configuration update that applied environment variables.

### Contents

#### Error

Error messages for environment variables that couldn't be applied.

Type: [EnvironmentError \(p. 471\)](#) 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# EventSourceMappingConfiguration

A mapping between an AWS resource and an AWS Lambda function. See [CreateEventSourceMapping \(p. 343\)](#) for details.

## Contents

### BatchSize

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

### EventSourceArn

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*?)`

Required: No

### FunctionArn

The ARN of the Lambda function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

### LastModified

The date that the event source mapping was last updated, in Unix time seconds.

Type: Timestamp

Required: No

### LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

Required: No

### State

The state of the event source mapping. It can be one of the following: `Creating`, `Enabling`, `Enabled`, `Disabling`, `Disabled`, `Updating`, or `Deleting`.

Type: String

Required: No

#### StateTransitionReason

The cause of the last state change, either `User initiated` or `Lambda initiated`.

Type: String

Required: No

#### UUID

The identifier of the event source mapping.

Type: String

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## FunctionCode

The code for the Lambda function. You can specify either an object in Amazon S3, or upload a deployment package directly.

### Contents

#### S3Bucket

An Amazon S3 bucket in the same 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# FunctionConfiguration

Details about a function's configuration.

## Contents

### CodeSha256

The SHA256 hash of the function's deployment package.

Type: String

Required: No

### CodeSize

The size of the function's deployment package, in bytes.

Type: Long

Required: No

### DeadLetterConfig

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 469\)](#) 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 \(p. 472\)](#) 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

#### Layers

The function's [layers](#).

Type: Array of [Layer \(p. 481\)](#) objects

Required: No

#### MasterArn

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-zA-Z]{2}(-gov)?-[a-zA-Z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

#### MemorySize

The memory 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: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

Required: No

Timeout

The amount of time 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 \(p. 488\)](#) 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 \(p. 490\)](#) 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# Layer

An AWS Lambda layer.

## Contents

### Arn

The Amazon Resource Name (ARN) of the function layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_]+\:[0-9]+\:`

Required: No

### CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## LayersListItem

Details about an [AWS Lambda layer](#).

### Contents

#### LatestMatchingVersion

The newest version of the layer.

Type: [LayerVersionsListItem \(p. 485\)](#) 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## LayerVersionContentInput

A ZIP archive that contains the contents of an [AWS Lambda layer](#). You can specify either an Amazon S3 location, or upload a layer archive directly.

### Contents

#### S3Bucket

The Amazon S3 bucket of the layer archive.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-\\_]\*(\?<!\.).\$

Required: No

#### S3Key

The Amazon S3 key of the layer archive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

#### S3ObjectVersion

For versioned objects, the version of the layer archive object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

#### ZipFile

The base64-encoded contents of the layer archive. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## LayerVersionContentOutput

Details about a version of an [AWS Lambda layer](#).

### Contents

#### CodeSha256

The SHA-256 hash of the layer archive.

Type: String

Required: No

#### CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

#### Location

A link to the layer archive in Amazon S3 that is valid for 10 minutes.

Type: String

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## LayerVersionsListItem

Details about a version of an [AWS Lambda layer](#).

### Contents

#### CompatibleRuntimes

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs8.10 | nodejs10.x | java8 | python2.7 | python3.6 | python3.7 | dotnetcore1.0 | dotnetcore2.1 | go1.x | ruby2.5 | provided

Required: No

#### CreatedDate

The date that the version was created, in ISO 8601 format. For example, 2018-11-27T15:10:45.123+0000.

Type: String

Required: No

#### Description

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

#### LayerVersionArn

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-\_]+:[0-9]+

Required: No

#### LicenseInfo

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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## TracingConfig

The function's AWS X-Ray tracing configuration.

### Contents

#### Mode

The tracing mode.

Type: String

Valid Values: Active | PassThrough

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## TracingConfigResponse

The function's AWS X-Ray tracing configuration.

### Contents

#### Mode

The tracing mode.

Type: String

Valid Values: Active | PassThrough

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## VpcConfig

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

### 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## 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 Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## SDK를 사용할 때의 인증서 오류

AWS SDK는 컴퓨터의 CA 인증서를 사용하기 때문에, AWS 서버의 인증서가 변경되면 SDK를 사용하려고 할 때 연결 장애가 발생할 수 있습니다. 컴퓨터의 CA 인증서와 운영 체제를 최신 상태로 유지하여 이러한 장애를 피할 수 있습니다. 본인 컴퓨터를 직접 관리하지 않는 기업 환경에서 이 문제가 발생하는 경우, 관리자에게 업데이트 프로세스를 문의해야 할 수 있습니다. 아래 목록에 최소한의 운영 체제 및 Java 버전이 나와 있습니다.

- 2005년 1월 이후 업데이트가 설치된 Microsoft Windows 버전의 경우, 신뢰할 수 있는 연결 목록에 필요한 CA가 하나 이상 들어 있습니다.
- Mac OS X 10.4 릴리스 5(2007년 2월), Mac OS X 10.5(2007년 10월) 및 그 이후 버전의 Java가 설치된 Mac OS X 10.4의 경우, 신뢰할 수 있는 연결 목록에 필요한 CA가 하나 이상 들어 있습니다.

- Red Hat Enterprise Linux 5(2007년 3월), 6, 7과 CentOS 5, 6, 7은 모두 신뢰할 수 있는 기본 CA 목록에 필요한 CA가 하나 이상 들어 있습니다.
- Java 1.4.2\_12(2006년 5월), 5 업데이트 2(2005년 3월), 그리고 Java 6(2006년 12월), 7, 8을 포함한 이후의 모든 버전은 신뢰할 수 있는 기본 CA 목록에 필요한 CA가 하나 이상 들어 있습니다.

브라우저를 통해서든 프로그래밍 방식으로든 관계없이 AWS Lambda 관리 콘솔이나 AWS Lambda API 엔드포인트에 액세스할 때 클라이언트 머신에서 다음 CA를 지원하는지 확인해야 합니다.

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

처음 두 기관의 루트 인증서는 [Amazon Trust Services](#)에서 구할 수 있지만, 컴퓨터를 최신 상태로 유지하는 것이 보다 직접적인 해결책입니다. ACM 제공 인증서에 대한 자세한 내용은 [AWS Certificate Manager FAQ](#)를 참조하십시오.

# AWS Lambda 릴리스

아래 표에 2018년 5월 이후 AWS Lambda 개발자 안내서의 주요 변경 사항이 설명되어 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 [RSS 피드](#)를 구독하면 됩니다.

update-history-change	update-history-description	update-history-date
<a href="#">Amazon Linux 2018.03</a>	Lambda 실행 환경이 Amazon Linux 2018.03을 사용하도록 업데이트되었습니다. 자세한 내용은 <a href="#">실행 환경</a> 을 참조하십시오.	May 21, 2019
<a href="#">Node.js 10</a>	새 런타임은 Node.js 10, nodejs10.x에서 사용할 수 있습니다. 이 런타임은 Node.js 10.15를 사용하며 주기적으로 Node.js 10의 최신 포인트 릴리스로 업데이트됩니다. 또한 Node.js 10은 Amazon Linux 2를 사용하는 첫 번째 런타임입니다. 자세한 내용은 <a href="#">Node.js를 사용하여 Lambda 함수 빌드</a> 를 참조하십시오.	May 13, 2019
<a href="#">GetLayerVersionByArn API</a>	GetLayerVersionByArn API를 사용하여 버전 ARN을 입력값으로 계층 버전 정보를 다운로드합니다. GetLayerVersion과는 달리, GetLayerVersionByArn을 이용하면 ARN을 분석하지 않고 바로 사용해 계층 이름과 버전 번호를 얻을 수 있습니다.	April 25, 2019
<a href="#">사용자 지정 런타임</a>	원하는 프로그래밍 언어로 Lambda 함수를 실행하는 사용자 지정 런타임을 빌드하십시오. 자세한 내용은 <a href="#">사용자 지정 AWS Lambda 런타임</a> 단원을 참조하십시오.	November 29, 2018
<a href="#">계층</a>	Lambda 계층을 사용하면 라이브러리, 사용자 지정 런타임 및 기타 종속 프로그램을 함수 코드와 별도로 패키징하고 배포할 수 있습니다. 사용 중인 계층을 귀하의 다른 계정 또는 전 세계 사용자들과 공유하십시오. 자세한 내용은 <a href="#">AWS Lambda 계층</a> 단원을 참조하십시오.	November 29, 2018
<a href="#">Ruby</a>	AWS Lambda는 새로운 런타임으로 Ruby 2.5를 지원합니다. 자세한 내용은 <a href="#">Ruby를 사용하여 Lambda 함수 빌드하기</a> 단원을 참조하십시오.	November 29, 2018

<a href="#">Application Load Balancer 트리거</a>	Elastic Load Balancing은 이제 Lambda 함수를 Application Load Balancer의 대상으로 지원합니다. 자세한 내용은 <a href="#">Application Load Balancer와 함께 Lambda 사용 단원</a> 을 참조하십시오.	November 29, 2018
<a href="#">Python 3.7</a>	AWS Lambda가 이제 새 런타임을 사용하여 Python 3.7을 지원합니다. 자세한 내용은 <a href="#">Python을 사용하여 Lambda 함수 작성</a> 을 참조하십시오.	November 19, 2018
<a href="#">Kinesis HTTP/2 스트림 소비자를 트리거로 사용</a>	Kinesis HTTP/2 데이터 스트림 소비자를 사용하여 AWS Lambda로 이벤트를 보낼 수 있습니다. 스트림 소비자는 데이터 스트림의 각 색드에서 전용 읽기 처리량이 있으며 HTTP/2를 사용하여 지연 시간을 최소화합니다. 자세한 내용은 <a href="#">Kinesis에서 AWS Lambda 사용</a> 을 참조하십시오.	November 19, 2018
<a href="#">비동기 함수 호출 페이로드 한도 향상</a>	Amazon SNS 트리거의 최대 메시지 크기에 맞추기 위해 비동기 호출에 대한 최대 페이로드 크기가 128KB에서 256KB로 향상되었습니다. 자세한 내용은 <a href="#">AWS Lambda 제한</a> 을 참조하십시오.	November 16, 2018
<a href="#">AWS GovCloud(미국 동부) 리전</a>	이제 AWS GovCloud(미국 동부) 리전에서 AWS Lambda를 사용할 수 있습니다. 자세한 내용은 AWS 블로그에서 <a href="#">AWS GovCloud (US-East) Now Open</a> 을 참조하십시오.	November 12, 2018
<a href="#">별도의 개발자 안내서로 AWS SAM 항목 이전</a>	많은 항목이 AWS Serverless Application Model(AWS SAM)을 사용하여 서비스 애플리케이션을 빌드하는 내용을 중점적으로 다루고 있습니다. 이러한 항목이 <a href="#">AWS Serverless Application Model 개발자 안내서</a> 로 이전되었습니다.	October 25, 2018
<a href="#">콘솔에서 Lambda 애플리케이션 보기</a>	Lambda 콘솔의 <a href="#">애플리케이션</a> 페이지에서 Lambda 애플리케이션의 상태를 볼 수 있습니다. 이 페이지에는 AWS CloudFormation 스택의 상태가 표시되며, 스택의 리소스에 대한 자세한 정보를 확인할 수 있는 페이지 링크도 포함되어 있습니다. 또한 애플리케이션에 대한 집계 지표를 보고 사용자 지정 모니터링 대시보드를 생성할 수도 있습니다.	October 11, 2018

함수 실행 제한 시간	함수가 오래 실행될 수 있도록 구성 가능한 최대 실행 제한 시간이 5분에서 15분으로 늘어났습니다. 자세한 내용은 <a href="#">AWS Lambda 제한</a> 을 참조하십시오.	October 10, 2018
AWS Lambda에서 PowerShell Core에 대한 지원	이제 AWS Lambda에서 PowerShell Core 언어를 지원합니다. 자세한 내용은 <a href="#">PowerShell에서 Lambda 함수를 작성하기 위한 프로그래밍 모델</a> 을 참조하십시오.	September 11, 2018
AWS Lambda에서 .NET Core 2.1.0 런타임 지원	이제 AWS Lambda에서 .NET Core 2.1.0 런타임을 지원합니다. 자세한 내용은 <a href="#">.NET Core CLI</a> 를 참조하십시오.	July 9, 2018
RSS에서 현재 사용 가능한 업데이트	AWS Lambda 개발자 안내서에 대한 알림을 받으려면 RSS 피드를 구독하면 됩니다.	July 5, 2018
중국(닝샤) 리전	이제 중국(닝샤) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 엔드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 엔드포인트</a> 단원을 참조하십시오.	June 28, 2018
이벤트 소스로 Amazon SQS 지원	이제 AWS Lambda가 이벤트 소스로서 Amazon Simple Queue Service(Amazon SQS)를 지원합니다. 자세한 내용은 <a href="#">Lambda 함수 호출</a> 을 참조하십시오.	June 28, 2018

## 이전 업데이트

다음 표에서는 2018년 6월 이전 AWS Lambda 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

변경 사항	설명	날짜
Node.js 런타임 8.10에 대한 런타임 지원	이제 AWS Lambda에서 Node.js 런타임 버전 8.10을 지원합니다. 자세한 내용은 <a href="#">Node.js를 사용하여 Lambda 함수 빌드</a> (p. 235) 단원을 참조하십시오.	2018년 2월 4일
함수 및 별칭 개정 ID	이제 AWS Lambda에서 함수 버전 및 별칭에 대한 개정 ID를 지원합니다. 이들 ID를 사용하면 함수 버전 또는 별칭 리소스를 업데이트할 때 조건부 업데이트를 추적하고 적용할 수 있습니다.	2018년 1월 25일
Go 및 .NET 2.0에 대한 런타임 지원	AWS Lambda에서 Go 및 .NET 2.0에 대한 런타임 지원이 추가되었습니다. 자세한 내용은 <a href="#">Go를 사용하여 Lambda 함수 빌드</a> (p. 283) 및 <a href="#">C#을 사용하여 Lambda 함수 빌드</a> (p. 295) 단원을 참조하십시오.	2018년 1월 15일

변경 사항	설명	날짜
콘솔 재설계	AWS Lambda에는 사용자 환경을 단순화하고 간편한 디버깅 및 함수 코드 수정 작업을 지원하는 Cloud9 코드 편집기가 추가된 새 Lambda 콘솔이 도입되었습니다. 자세한 내용은 <a href="#">AWS Lambda 콘솔 편집기를 사용하여 함수 생성 (p. 24)</a> 단원을 참조하십시오.	2017년 11 월 30일
개별 함수에 대한 동시성 한도 설정	AWS Lambda에서는 이제 개별 함수에 대한 동시성 한도 설정을 지원합니다. 자세한 내용은 <a href="#">동시성 관리 (p. 37)</a> 단원을 참조하십시오.	2017년 11 월 30일
별칭을 사용한 트래픽 이동	AWS Lambda에서는 이제 별칭을 사용한 트래픽 이동을 지원합니다. 자세한 내용은 <a href="#">별칭을 사용한 트래픽 이동 (p. 60)</a> 단원을 참조하십시오.	2017년 11 월 28일
점진적 코드 배포	AWS Lambda에서는 이제 코드 배포를 사용하여 Lambda 함수의 새 버전을 안전하게 배포할 수 있도록 지원합니다. 자세한 내용은 <a href="#">점진적 코드 배포</a> 단원을 참조하십시오.	2017년 11 월 28일
중국(북경) 지역	이제 중국(북경) 지역에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 앤드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 앤드포인트</a> 단원을 참조하십시오.	2017년 11 월 9일
SAM Local 소개	AWS Lambda는 Lambda 런타임으로 업로드를 하기 전에 서비스 애플리케이션을 로컬로 개발, 테스트 및 분석할 수 있는 환경을 제공하는 AWS CLI 도구인 SAM Local(지금은 SAM CLI라고도 함)을 도입했습니다. 자세한 내용은 <a href="#">서비스 애플리케이션 테스트 및 디버그</a> 단원을 참조하십시오.	2017년 8월 11일
캐나다(중부) 리전	이제 캐나다(중부) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 앤드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 앤드포인트</a> 단원을 참조하십시오.	2017년 6월 22일
남아메리카(상파울루) 리전	이제 남아메리카(상파울루) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 앤드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 앤드포인트</a> 단원을 참조하십시오.	2017년 6월 6일
AWS X-Ray에 대한 AWS Lambda 지원	Lambda는 Lambda 애플리케이션에서 성능 문제를 감지, 분석 및 최적화할 수 있도록 허용하는 X-Ray에 대한 지원을 도입했습니다. 자세한 내용은 <a href="#">AWS X-Ray 사용 (p. 226)</a> 단원을 참조하십시오.	2017년 4월 19일
아시아 태평양(뭄바이) 리전	이제 아시아 태평양(뭄바이) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 앤드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 앤드포인트</a> 단원을 참조하십시오.	2017년 3월 28일
AWS Lambda는 Node.js 런타임 v6.10을 지원합니다.	AWS Lambda에서 Node.js 런타임 v6.10에 대한 지원이 추가되었습니다. 자세한 내용은 <a href="#">Node.js를 사용하여 Lambda 함수 빌드 (p. 235)</a> 단원을 참조하십시오.	2017년 3월 22일
EU(런던) 리전	이제 EU(런던) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 앤드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 앤드포인트</a> 단원을 참조하십시오.	2017년 2월 1일

변경 사항	설명	날짜
AWS Lambda는 .NET 런타임, Lambda@Edge(프리뷰), 배달 못한 편지 대기열 및 서비스 애플리케이션의 자동 배포를 지원합니다.	<p>AWS Lambda에서 C#에 대한 지원이 추가되었습니다. 자세한 내용은 <a href="#">C#을 사용하여 Lambda 함수 빌드 (p. 295)</a> 단원을 참조하십시오.</p> <p>Lambda@Edge를 사용하면 CloudFront 이벤트에 대한 응답으로 AWS 엣지 로케이션에서 Lambda 함수를 실행할 수 있습니다. 자세한 내용은 <a href="#">CloudFront Lambda@Edge와 함께 AWS Lambda (p. 161)</a> 단원을 참조하십시오.</p>	2016년 3월 12일
AWS Lambda가 지원되는 이벤트 소스로서 Amazon Lex를 추가합니다.	<p>Lambda 및 Amazon Lex을 사용하여 Slack 및 Facebook 같은 다양한 서비스에서 채팅 봇을 신속하게 빌드할 수 있습니다. 자세한 내용은 <a href="#">Using AWS Lambda with Amazon Lex (p. 188)</a> 단원을 참조하십시오.</p>	2016년 11월 30일
미국 서부(캘리포니아 북부) 리전	<p>이제 미국 서부(캘리포니아 북부) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 엔드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 엔드포인트</a> 단원을 참조하십시오.</p>	2016년 11월 21일
Lambda 기반 애플리케이션을 생성 및 배포하고 Lambda 함수 구성 설정에서 환경 설정을 사용할 수 있도록 AWS Serverless Application Model을 도입했습니다.	<p><b>AWS Serverless Application Model:</b> 이제 AWS SAM을 사용하여 서비스 애플리케이션 내에 리소스를 표현하기 위한 구문을 정의할 수 있습니다. 애플리케이션을 배포하려면 애플리케이션의 요소로서 필요한 리소스와 AWS CloudFormation 템플릿 파일(JSON 또는 YAML로 작성)에 연결된 권한 정책을 지정하고 배포 아티팩트를 패키징한 다음, 템플릿을 배포하기만 하면 됩니다. 자세한 내용은 <a href="#">AWS Lambda 애플리케이션 (p. 113)</a> 단원을 참조하십시오.</p> <p><b>환경 변수:</b> 환경 변수를 사용하여 함수 코드 밖에서 Lambda 함수에 대한 구성 설정을 지정할 수 있습니다. 자세한 내용은 <a href="#">AWS Lambda 환경 변수 (p. 39)</a> 단원을 참조하십시오.</p>	2016년 11월 18일
아시아 태평양(서울) 리전	<p>이제 아시아 태평양(서울) 리전에서 AWS Lambda를 사용할 수 있습니다. Lambda 리전 및 엔드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 엔드포인트</a> 단원을 참조하십시오.</p>	2016년 8월 29일
아시아 태평양(시드니) 리전	<p>이제 아시아 태평양(시드니) 리전에서 Lambda를 사용할 수 있습니다. Lambda 리전 및 엔드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 엔드포인트</a> 단원을 참조하십시오.</p>	2016년 6월 23일
Lambda 콘솔 업데이트	<p>Lambda 콘솔은 역할 생성 프로세스를 간소화하도록 업데이트되었습니다. 자세한 내용은 <a href="#">콘솔로 Lambda 함수 만들기 (p. 3)</a> 단원을 참조하십시오.</p>	2016년 6월 23일
이제 AWS Lambda에서 Node.js 런타임 v4.3에 대한 지원이 추가되었습니다. 자세한 내용은 <a href="#">Node.js를 사용하여 Lambda 함수 빌드 (p. 235)</a> 단원을 참조하십시오.	<p>AWS Lambda에서 Node.js 런타임 v4.3에 대한 지원이 추가되었습니다. 자세한 내용은 <a href="#">Node.js를 사용하여 Lambda 함수 빌드 (p. 235)</a> 단원을 참조하십시오.</p>	2016년 4월 07일
EU(프랑크푸르트) 리전	<p>이제 EU(프랑크푸르트) 리전에서 Lambda를 사용할 수 있습니다. Lambda 리전 및 엔드포인트에 대한 자세한 내용은 AWS General Reference의 <a href="#">리전 및 엔드포인트</a> 단원을 참조하십시오.</p>	2016년 3월 14일
VPC 지원	<p>이제 VPC의 리소스에 액세스하도록 Lambda 함수를 구성할 수 있습니다. 자세한 내용은 <a href="#">Amazon VPC에서 리소스에 액세스하도록 Lambda 함수 구성 (p. 66)</a> 단원을 참조하십시오.</p>	2016년 2월 11일

변경 사항	설명	날짜
AWS Lambda 런타임이 업데이트되었습니다.	<p><a href="#">실행 환경 (p. 99)</a>이 업데이트되었습니다.</p>	2015년 11 월 4일
버전 관리 지원, Lambda 함수의 코드를 개발하기 위한 Python, 예약된 이벤트, 실행 시간 증가	<p>이제 Python을 사용하여 Lambda 함수 코드를 배포할 수 있습니다. 자세한 내용은 <a href="#">Python을 사용하여 Lambda 함수 빌드 (p. 245)</a> 단원을 참조하십시오.</p> <p>버전 관리: Lambda 함수에 대한 하나 이상의 버전을 유지할 수 있습니다. 버전 관리는 다른 환경(예를 들어 개발, 테스트 또는 프로덕션)에서 실행되는 Lambda 함수 버전을 관리할 수 있게 해줍니다. 자세한 내용은 <a href="#">AWS Lambda 함수 버전 관리 및 별칭 (p. 45)</a> 단원을 참조하십시오.</p> <p>예정된 이벤트: AWS Lambda 콘솔을 사용하여 코드를 정기적으로 호출할 수 있도록 AWS Lambda를 설정할 수도 있습니다. 고정 비율(시간, 일 또는 주)을 지정하거나 cron 식을 지정할 수 있습니다. 예는 <a href="#">Using AWS Lambda with Amazon CloudWatch Events (p. 153)</a> 단원을 참조하십시오.</p> <p>실행 시간 증가: 대용량 데이터 수집 및 처리 작업 같은 함수를 더 오래 실행할 수 있도록 최대 5분 동안 Lambda 함수가 실행되도록 설정할 수 있습니다.</p>	2015년 10 월 08일
DynamoDB Streams 지원	<p>이제 DynamoDB Streams는 상용 버전으로 사용할 수 있으며, DynamoDB가 출시된 모든 리전에서 사용이 가능합니다. 테이블에서 DynamoDB Streams를 활성화하고 테이블에 대한 트리거로서 Lambda 함수를 사용할 수 있습니다. 트리거는 DynamoDB 테이블에 대해 수행된 업데이트에 대한 응답에서 취할 수 있는 사용자 지정 작업입니다. 예제 연습을 보려면 <a href="#">자습서: Amazon DynamoDB 스트림에 AWS Lambda 사용 (p. 167)</a> 단원을 참조하십시오.</p>	2015년 7월 14일
이제 AWS Lambda는 REST와 호환 가능한 클라이언트에서의 Lambda 함수 호출을 지원합니다.	<p>지금까지는 웹, 모바일 또는 IoT 애플리케이션에서 Lambda 함수를 호출하려면 AWS SDK(예를 들어 Java용 AWS SDK, Android용 AWS SDK, iOS용 AWS SDK)가 필요했습니다. 이제는 AWS Lambda가 Amazon API Gateway를 사용하여 생성할 수 있는 사용자 지정이 가능한 API를 통해 REST 호환 클라이언트에서 Lambda 함수를 호출할 수 있도록 지원합니다. Lambda 함수 엔드포인트 URL에 요청을 전송할 수 있습니다. 개방형 액세스를 허용하거나 AWS Identity and Access Management(IAM)을 활용하여 액세스를 인증하거나 API 키를 사용하여 Lambda 함수에 대한 다른 사람의 액세스를 측정할 수 있도록 엔드포인트에서 보안을 구성할 수 있습니다.</p> <p>시작하기 연습의 예제를 보려면 <a href="#">Using AWS Lambda with Amazon API Gateway (p. 131)</a> 단원을 참조하십시오.</p> <p>Amazon API Gateway에 대한 자세한 내용은 <a href="https://aws.amazon.com/api-gateway/">https://aws.amazon.com/api-gateway/</a>를 참조하십시오.</p>	2015년 7월 09일
이제 AWS Lambda 콘솔은 Lambda 함수를 손쉽게 생성하고 테스트할 수 있도록 블루프린트를 제공합니다.	<p>AWS Lambda 콘솔은 블루프린트 집합을 제공합니다. 각 블루프린트는 Lambda 기반 애플리케이션을 손쉽게 생성하는데 사용할 수 있도록 Lambda 함수에 대한 샘플 이벤트 소스 구성과 샘플 코드를 제공합니다. 이제 AWS Lambda 시작하기 연습은 블루프린트를 사용합니다. 자세한 내용은 <a href="#">AWS Lambda 시작하기 (p. 3)</a> 단원을 참조하십시오.</p>	이 릴리스

변경 사항	설명	날짜
이제 AWS Lambda는 Lambda 함수를 작성할 수 있도록 Java를 지원합니다.	이제 Java로 Lambda 코드를 작성할 수 있습니다. 자세한 내용은 <a href="#">Java를 사용하여 Lambda 함수 빌드 (p. 257)</a> 단원을 참조하십시오.	2015년 6월 15일
이제 AWS Lambda에서 Lambda 함수를 생성 또는 업데이트할 때 함수.zip으로 Amazon S3 객체를 지정할 수 있습니다.	Lambda 함수 배포 패키지(.zip 파일)를 Lambda 함수를 생성하고 싶은 동일한 리전의 Amazon S3 버킷에 업로드할 수 있습니다. 그런 다음, Lambda 함수를 생성 또는 업데이트할 때 버킷 이름과 객체 키 이름을 지정할 수 있습니다.	2015년 5월 28일
이제 AWS Lambda는 모바일 백엔드 지원을 추가한 상용 버전으로 사용할 수 있습니다.	<p>AWS Lambda는 프로덕션 용도의 상용 버전으로 사용할 수 있습니다. 이 릴리스에는 AWS Lambda를 사용하여 모바일, 태블릿 및 사물인터넷(IoT) 백엔드를 손쉽게 빌드함으로써 인프라 프로비저닝 또는 관리 없이도 자동으로 확장이 가능하게 해주는 기능들이 새로 추가되었습니다. AWS Lambda는 실시간(동기식) 이벤트와 비동기식 이벤트를 모두 지원합니다. 추가 기능에는 보다 쉬워진 이벤트 소스 구성 및 관리 기능이 포함되어 있습니다. Lambda 함수에 대한 리소스 정책을 도입하여 권한 모델 및 프로그래밍 모델을 간소화했습니다.</p> <p>이에 따라 설명서가 업데이트되었습니다. 자세한 내용은 다음 주제를 참조하십시오.</p> <p><a href="#">AWS Lambda 시작하기 (p. 3)</a>  <a href="#">AWS Lambda</a></p>	2015년 4월 9일
미리 보기 릴리스	AWS Lambda 개발자 안내서 미리 보기 릴리스	2014년 11월 13일

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.