

SRV213-R

# AWS re:Invent

## Thirty Serverless Architectures in ~~50~~ Minutes

Chris Munns – Senior Developer Advocate – Serverless

November 27, 2017

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Don't blink because in this session, we quickly show you thirty different architectural patterns that you can use with **AWS Lambda** to solve everything from **basic infrastructure automation tasks** to **building chatbots**. We cover the services that connect to AWS Lambda and help you **create serverless applications** that can respond to requests from many AWS services today. We will also discuss **how to secure these serverless applications, deploy them, monitor them, and profile them** for issues. By the end of this session, expect to have ideas on **how serverless architecture can improve your life**.

### About me:

Chris Munns - [munns@amazon.com](mailto:munns@amazon.com), [@chrismunns](https://twitter.com/chrismunns)

- Senior Developer Advocate—Serverless
- New Yorker
- Previously:
  - AWS Business Development Manager – DevOps, July '15 - Feb '17
  - AWS Solutions Architect Nov, 2011 - Dec 2014
  - Formerly on operations teams @Etsy and @Meetup
  - Little time at a hedge fund, Xerox, and a few other startups
- Rochester Institute of Technology: Applied Networking and Systems Administration '05
- Internet infrastructure geek



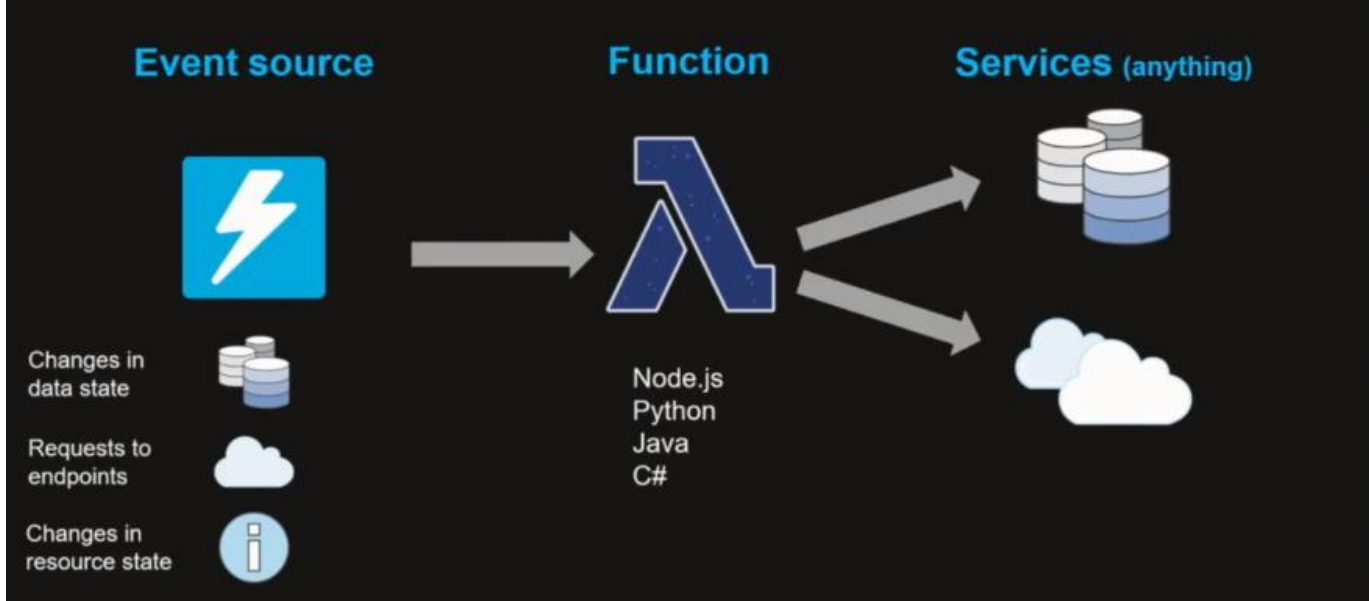
AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



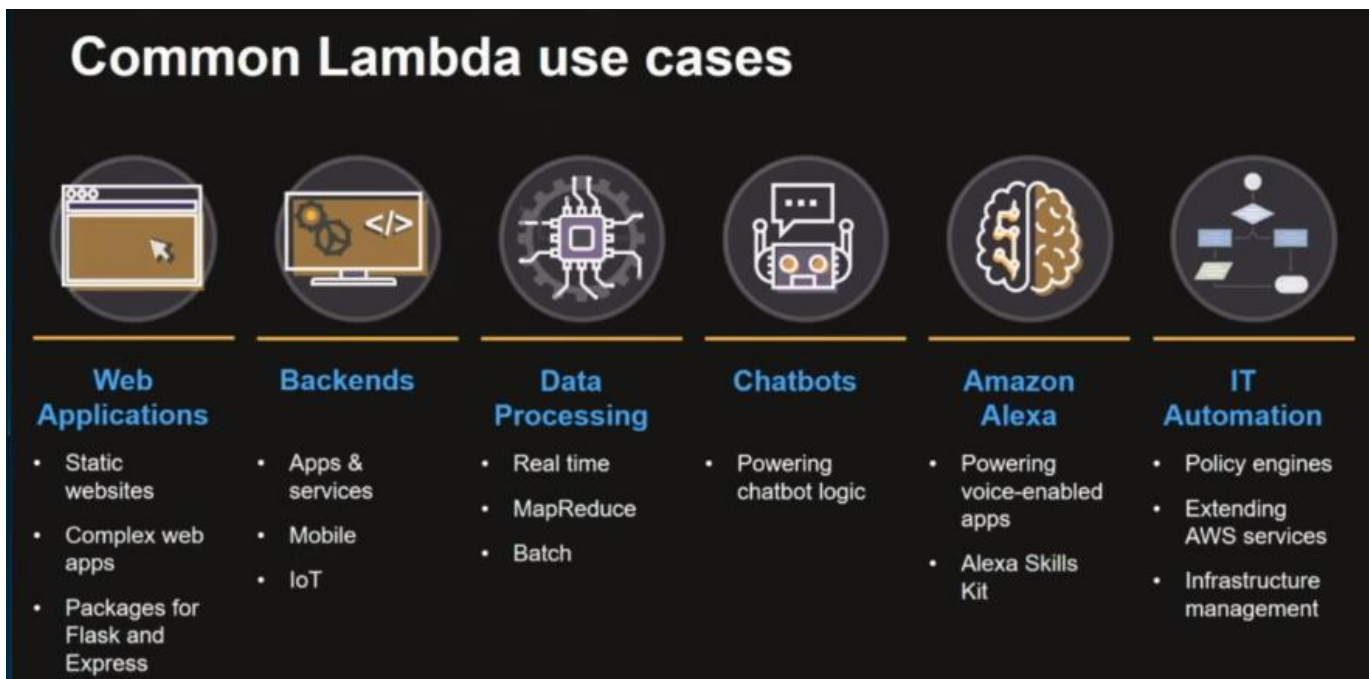
# Why are we here today?

## Serverless applications



The event invocation source triggers the lambda function

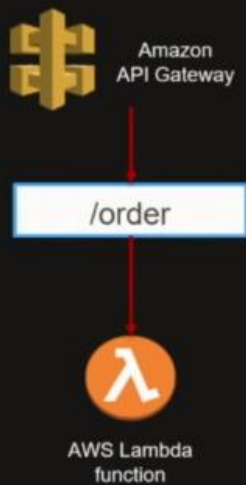
## Common Lambda use cases





# Lambda execution model

## Synchronous (push)



## Asynchronous (event)



## Stream-based



There are 3 different kinds of invocation models for using Lambda today, Asynchronous is a fire and forget approach.

# Lambda permissions model

## Fine grained security controls for both execution and invocation:

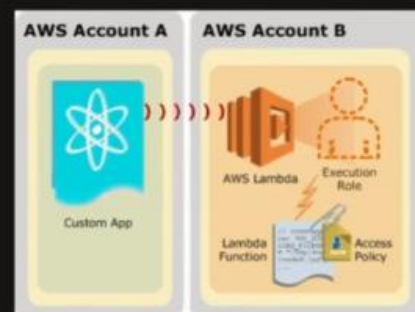
### Execution policies:

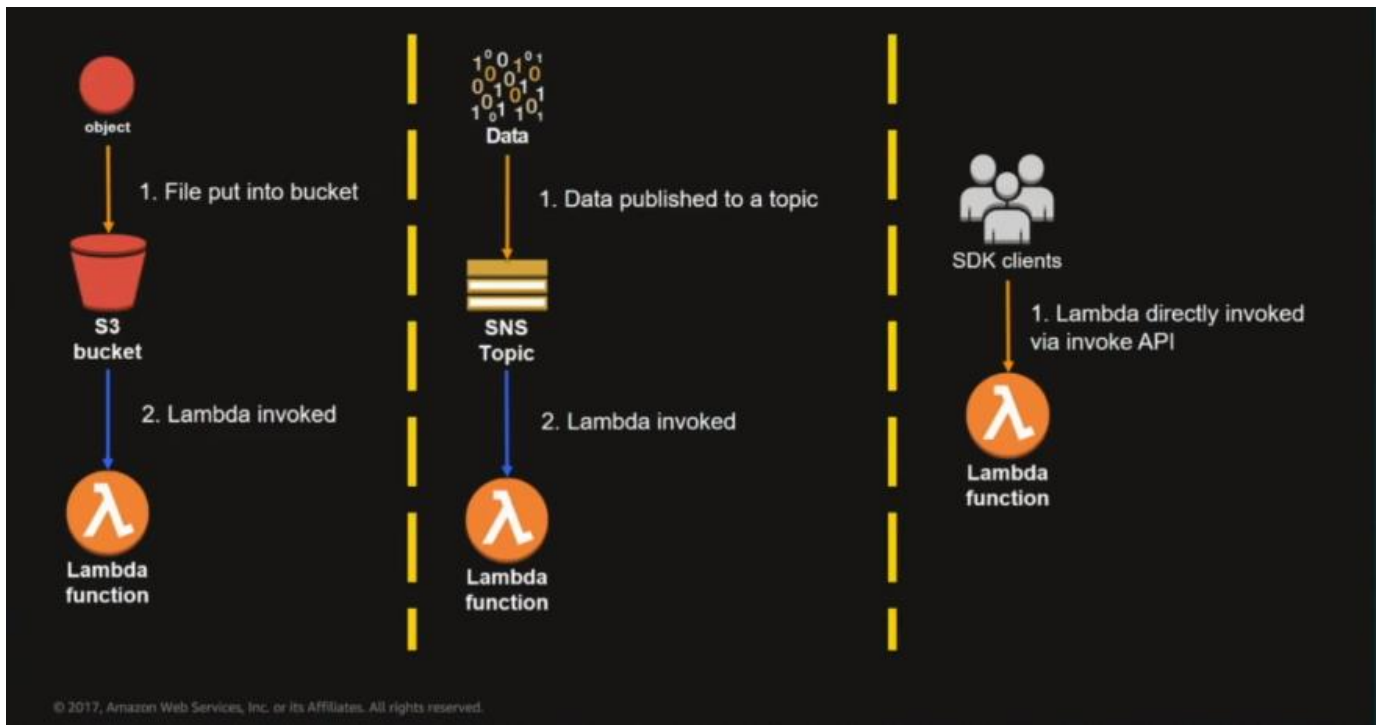
- Define what AWS resources/API calls can this function access via IAM
- Used in streaming invocations
- E.g. "Lambda function A can read from DynamoDB table users"

### Function policies:

- Used for sync and async invocations
- E.g. "Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross account access

```
1- {  
2-   "Version": "2012-10-17",  
3-   "Statement": [  
4-     {  
5-       "Effect": "Allow",  
6-       "Action": [  
7-         "logs:CreateLogGroup",  
8-         "logs:CreateLogStream",  
9-         "logs:PutLogEvents"  
10-      ],  
11-      "Resource": "*" }  
12-   ]  
13- }  
14- }
```

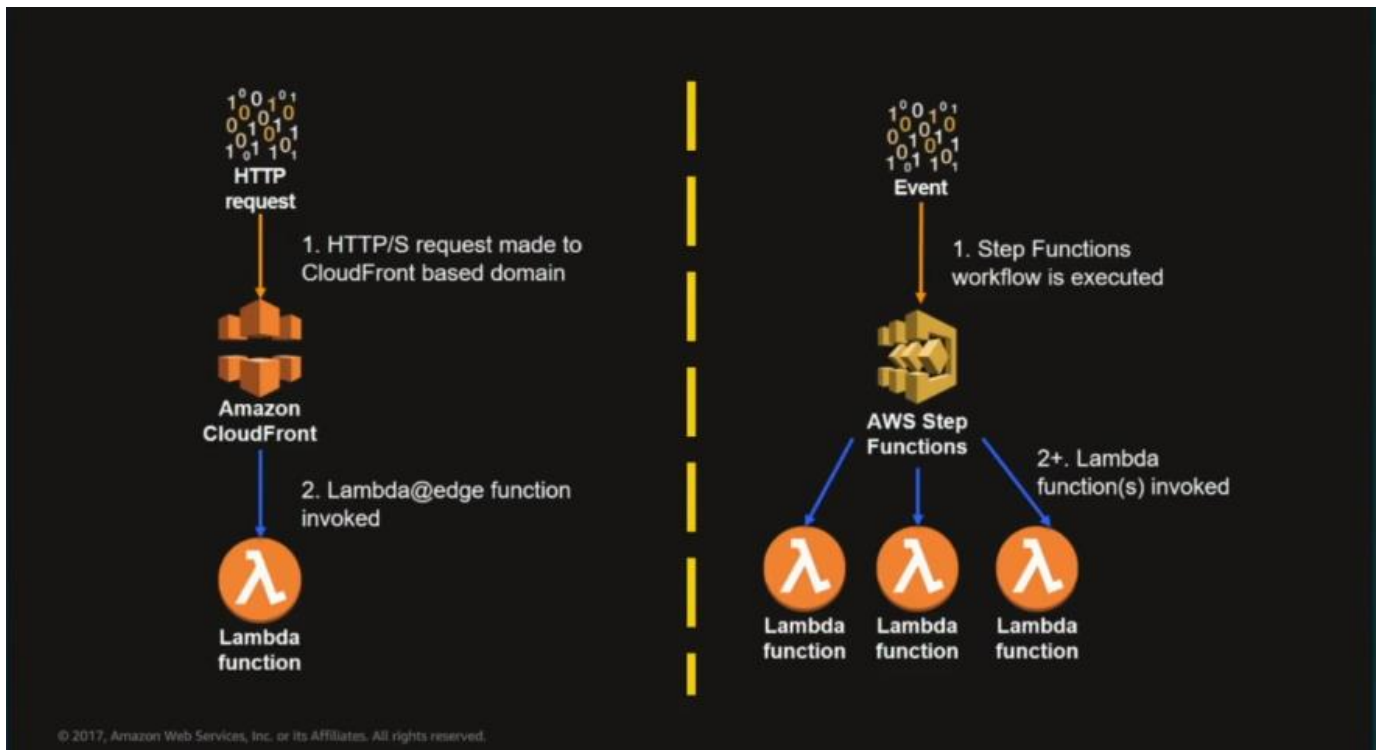




We can attach compute to storage in the form above. This allows some form of computation be done after data is put into a S3 bucket to do things like process a text based file, take an image and resize it, take an image and use AWS Rekognition to get metadata for that image before storing it, doing transcoding, etc. In this case, ***S3 is going to directly invoke Lambda with a payload containing information about the bucket and the object.***

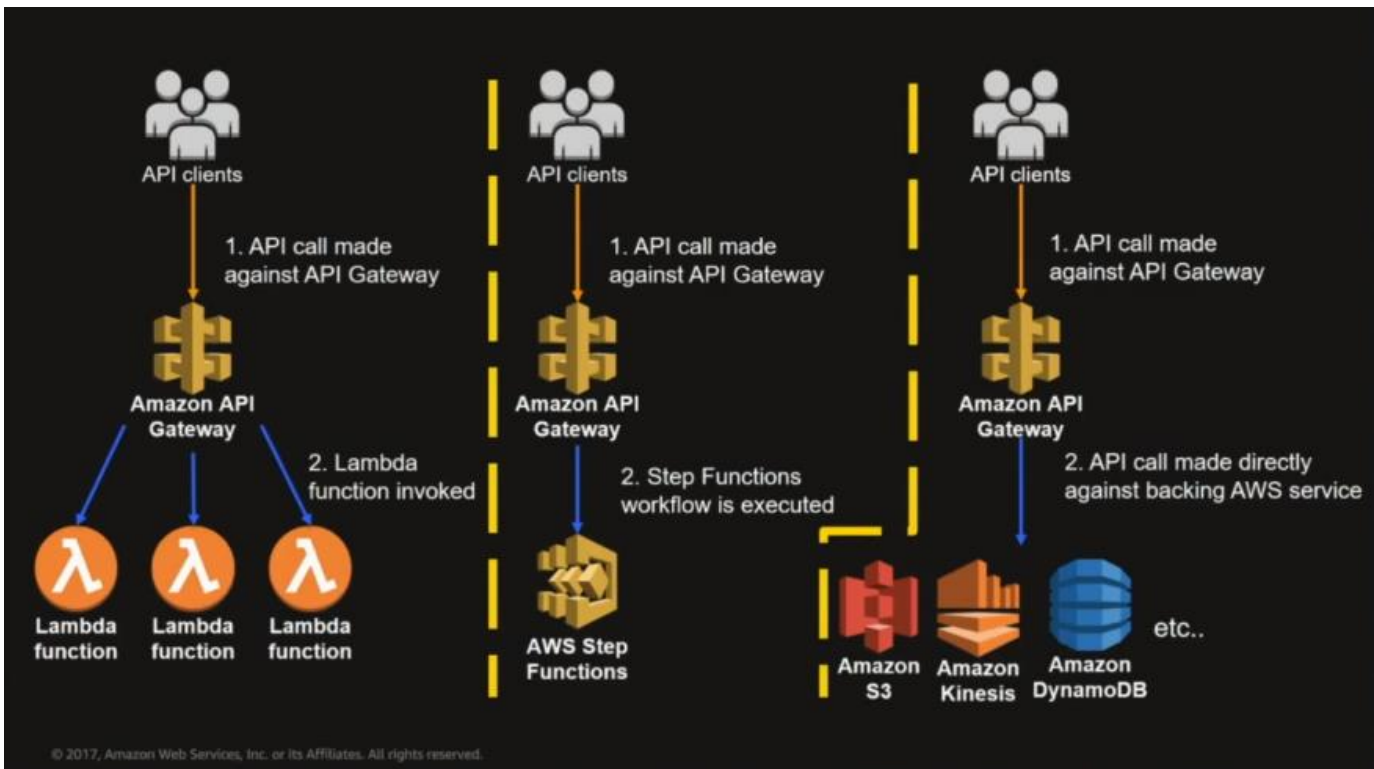
Another use case is having **SNS** go and trigger a lambda function, ***SNS is one of the core messaging services at AWS***, it used for ***PubSub*** and ***SMS push-based model architectures***. In this case we have a ***SNS Topic***, data is going to come into the topic from some other AWS service or directly via the SNS Topic API and those messages are going to be made available to a ***lambda*** function that will pull them in and execute on them.

Lastly, Lambda functions can be invoked directly in the API. You don't have to have another invocation service in front of a lambda function if you are controlling the calling.



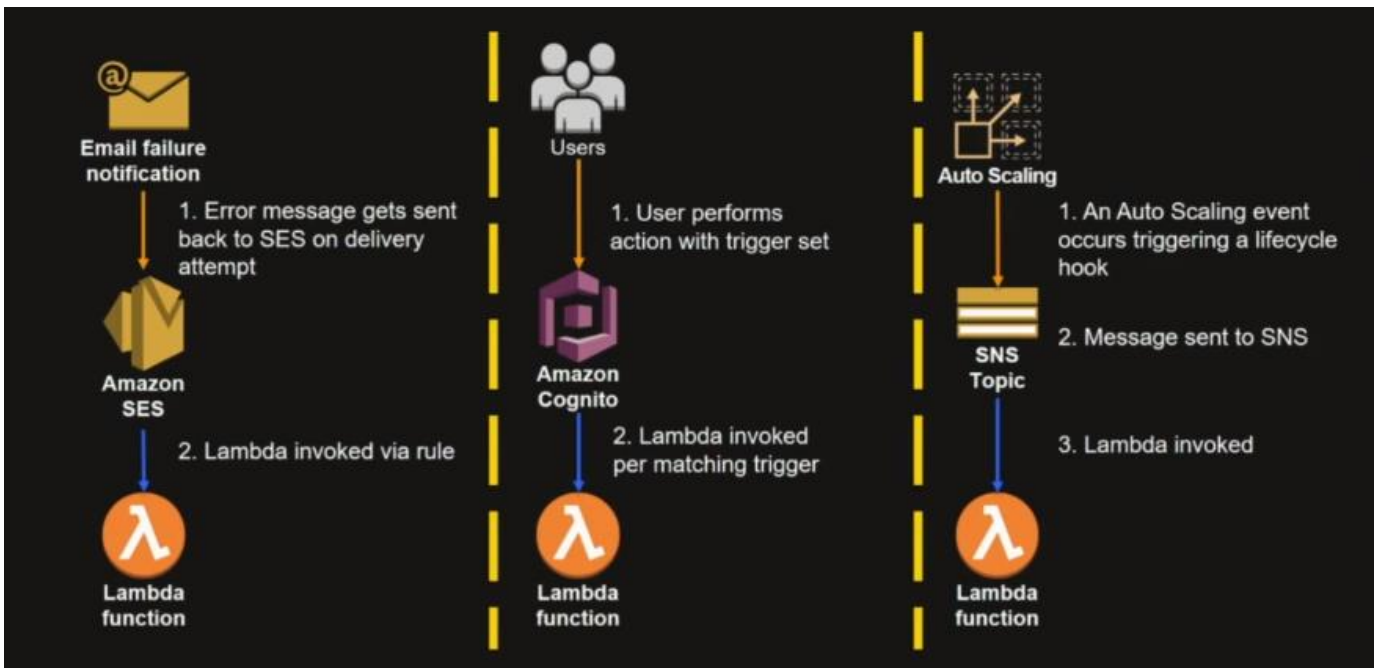
Next, we look at **Lambda at the Edge** which is technically part of the AWS CloudFront/CDN product. Lambda at the Edge allows you to invoke a lambda function directly in those edge nodes for doing things like processing headers and cookies in the incoming requests, taking the requests and figuring where it should be routed to, mutating or transforming the requests, have it talk directly to services like DynamoDB to move the compute all the way to the edge for faster response time.

**AWS Step Functions** is a workflow execution management service, primarily used for managing **state machines**. An event comes into a step function, step functions can then process the payload of that event, do things like decision trees, parallelization, retry, catch and fail, and other capabilities. **This allows you to bring out the orchestration logic in your business logic out into the step function level** so that your individual lambda code becomes more streamlined faster.



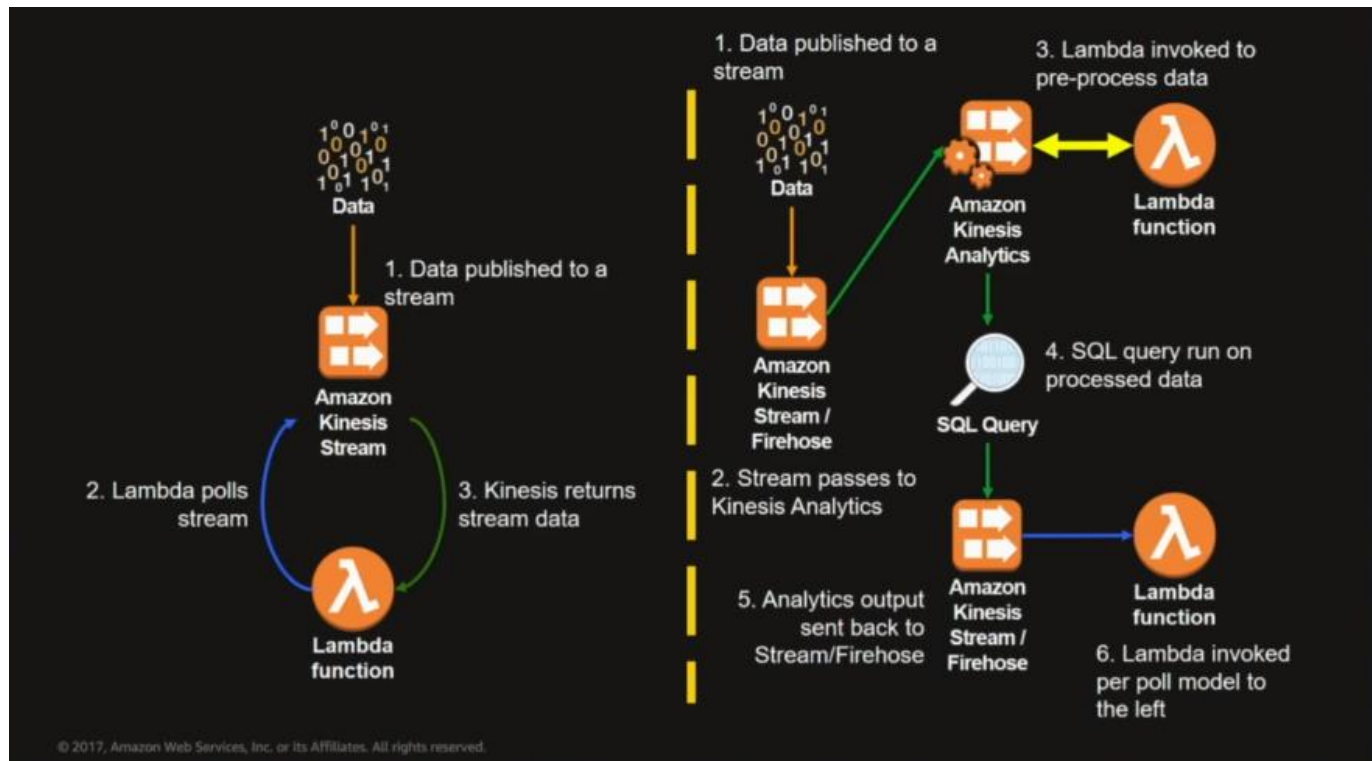
**API Gateway** allows you to *create entirely serverless backends* by using different lambdas to serve the endpoints. You can also invoke step functions directly from an API Gateway request, this can be used when you have a much larger request payload that you need to do some complex process with it, you can directly invoke the step function from the API Gateway instead of invoking a lambda that will in turn invoke the step function. This is more useful in an asynchronous model where you are not expecting a return value back as part of the request.

You can also put API Gateway directly in front of some services like DynamoDB, Kinesis, S3, etc. this is good for mobile use cases where you need to give customers access to data that they have in DynamoDB, allow them to pull down data from S3, etc.



These are some of the *messaging services use cases* available.

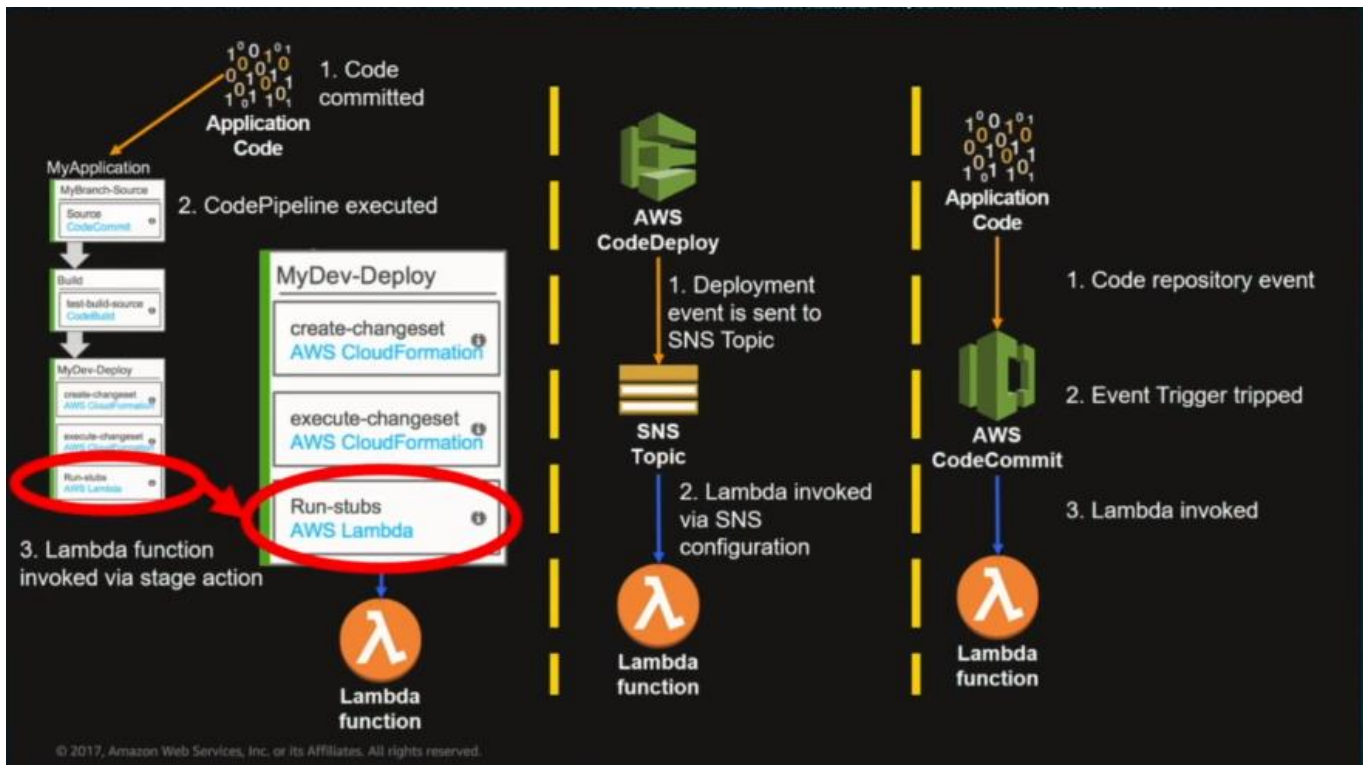
You can send out emails using **SES** and then have SES send back error responses from the recipient's email server to a **lambda** function that will do something with the responses like flagging that customer emails to not send them emails anymore until they re-subscribe or log in. **Amazon Cognito** has the ability to trigger a lambda as part of different lifecycle events like resetting password, updating profiles, etc. you can invoke a lambda function to react to events like that.



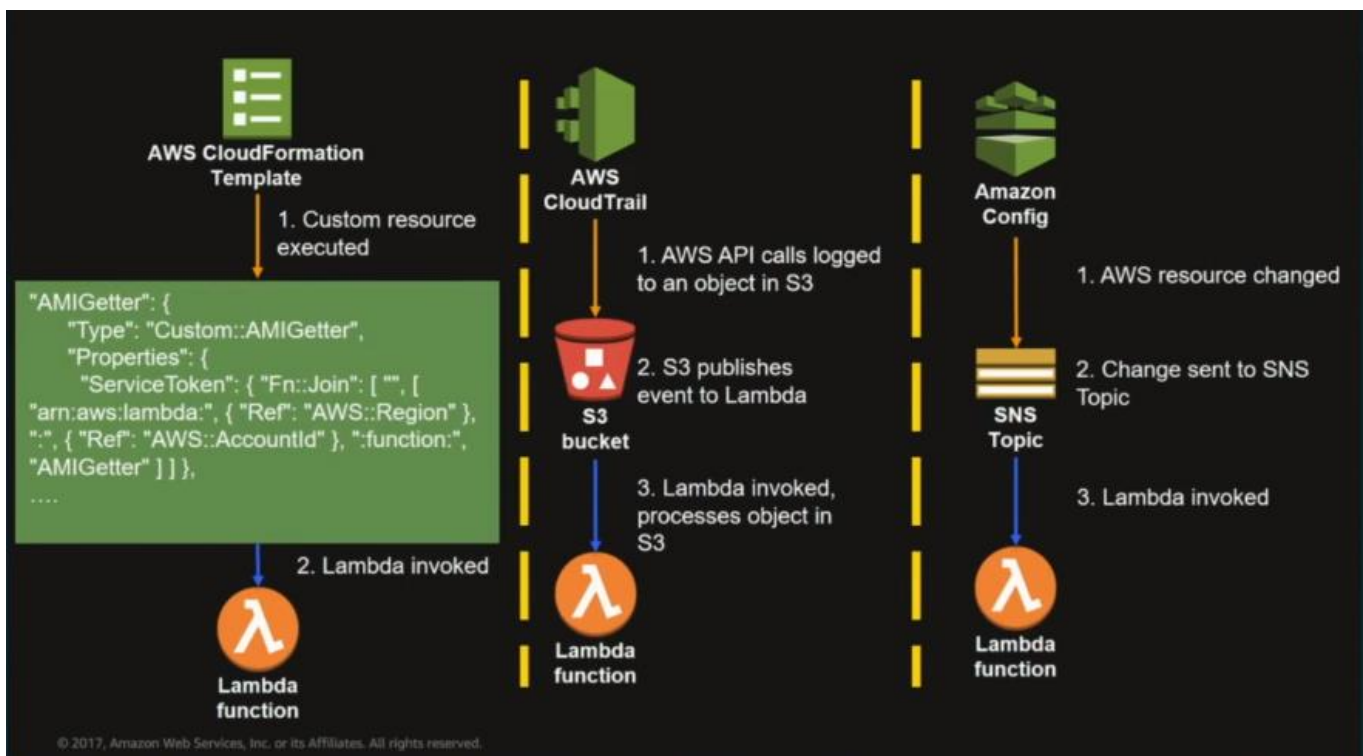
These are some of the Streaming Analytics capabilities use cases with Lambda. Amazon Kinesis has Streams, Firehose, and Analytics can be used when you need to do very high throughput ingesting of data like in the ads tech space where you want to do beacon collection or click tracking on a website, ingesting logs data from devices. As data comes into Kinesis, you have the ability to invoke a lambda on a shard by shard basis.

We can also work with Kinesis Analytics that allows you to take in data through a Firehose, pre-process the data using a lambda, process it in near real-time with normal SQL queries, and output the result data to whatever source you want.



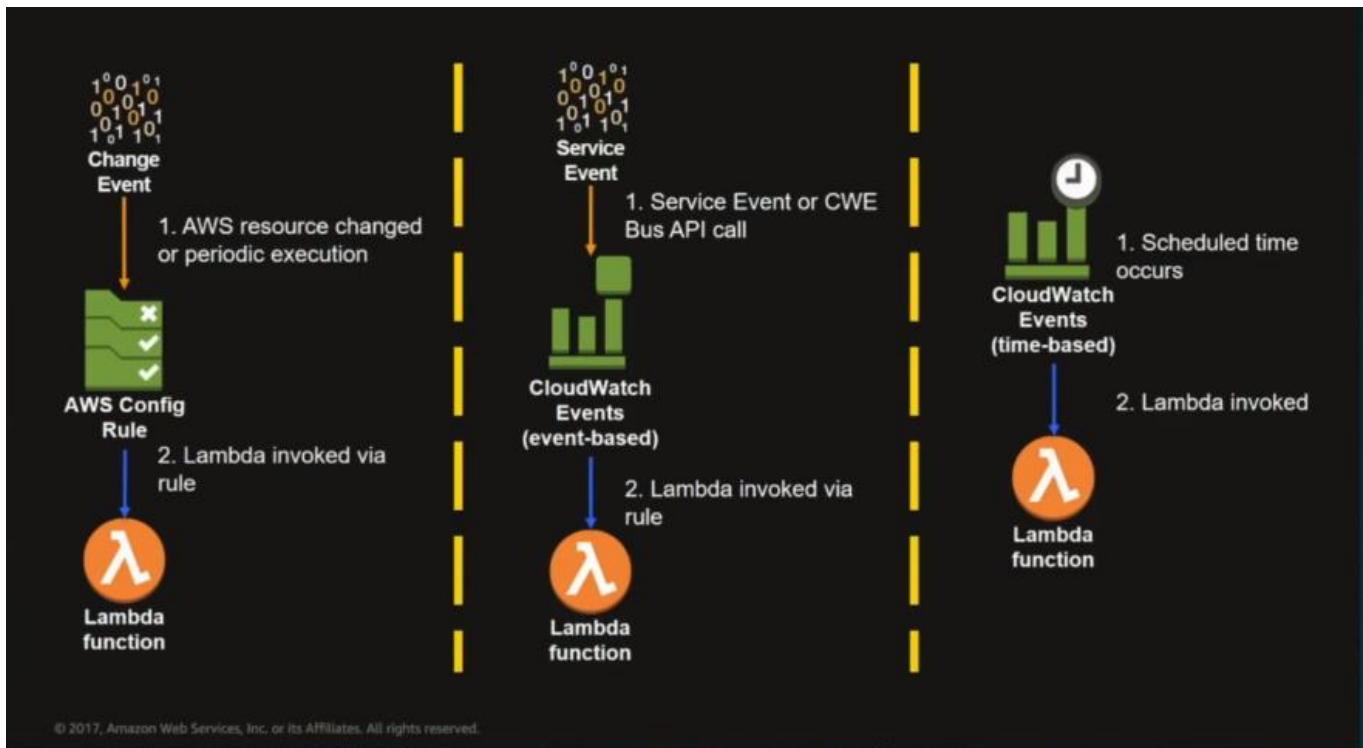


These are some of the development and management use cases.

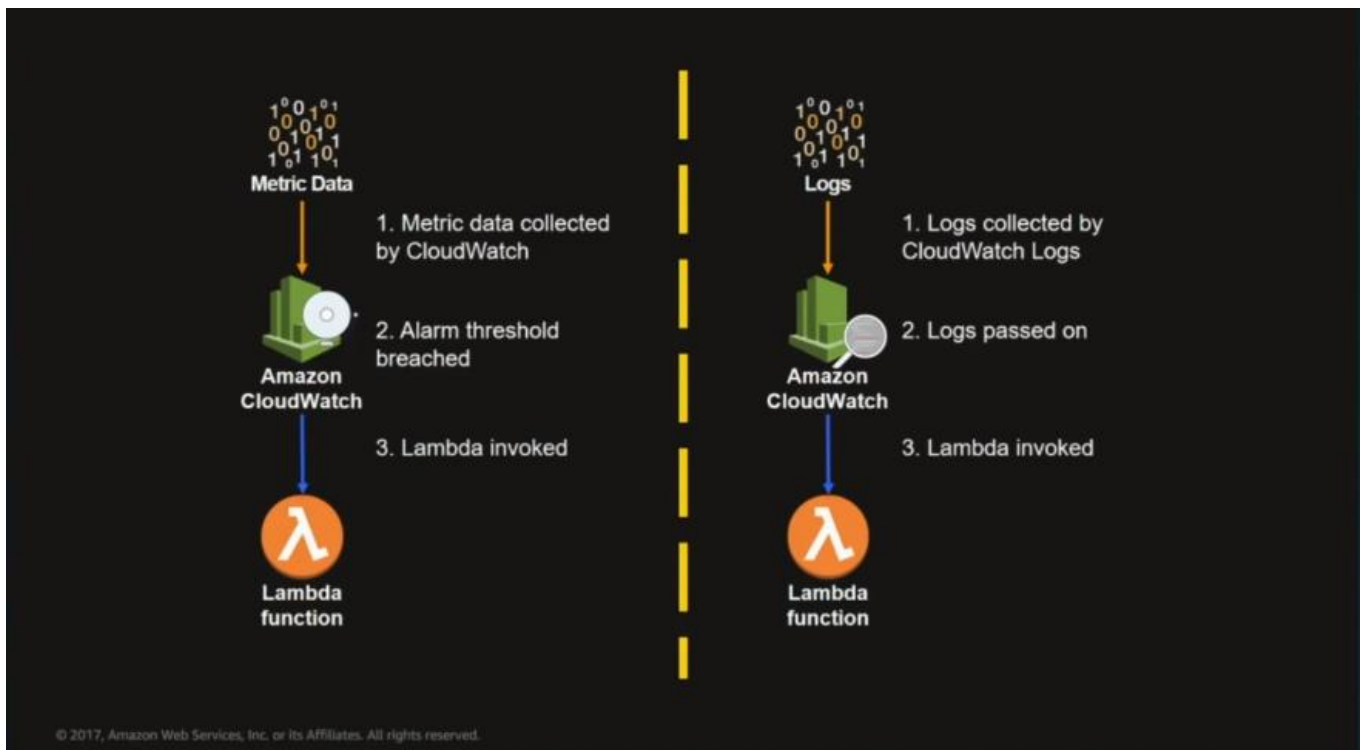


A **custom resource** in **CloudFormation** is a resource that you define using a lambda function, this is useful for things like reaching out to other services like EC2 for the latest AMI deployed, a VPC subnet that is assigned to your Dev environment using their tags, etc.

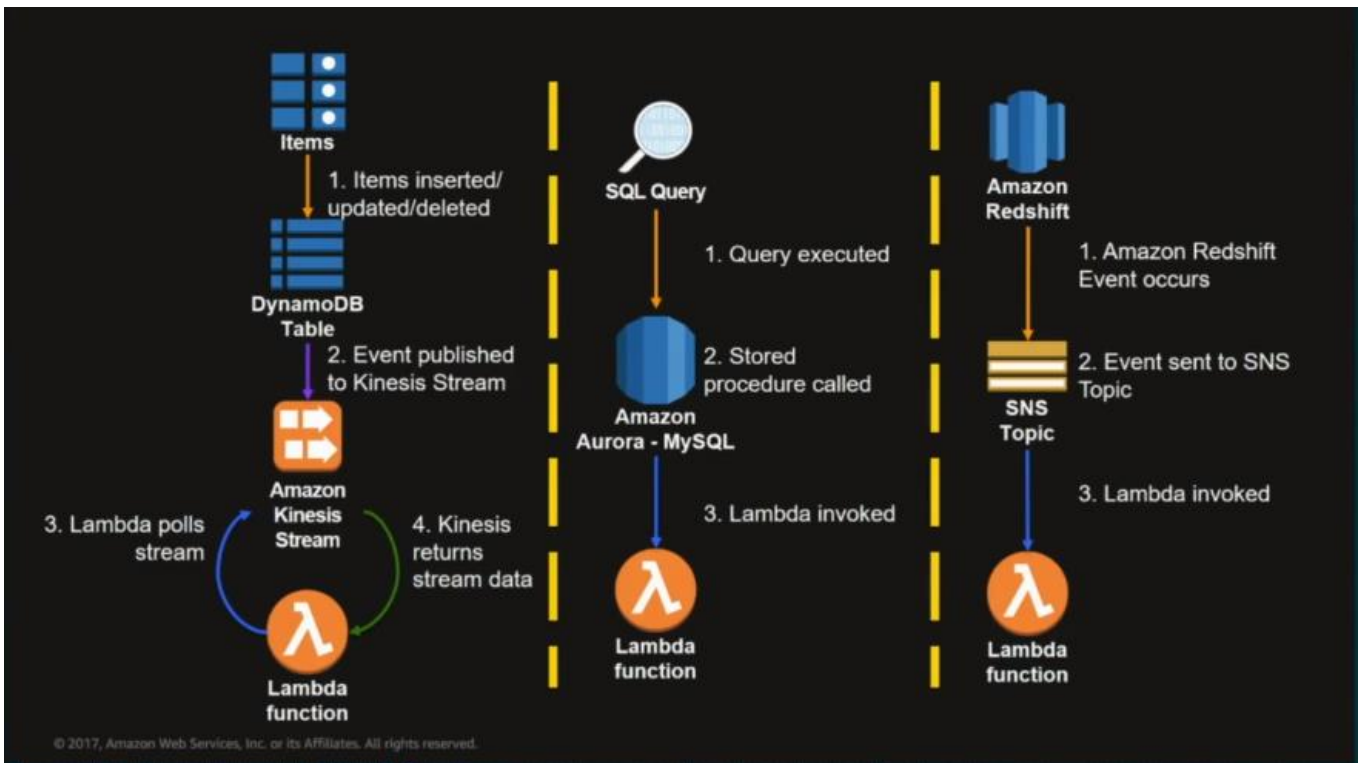




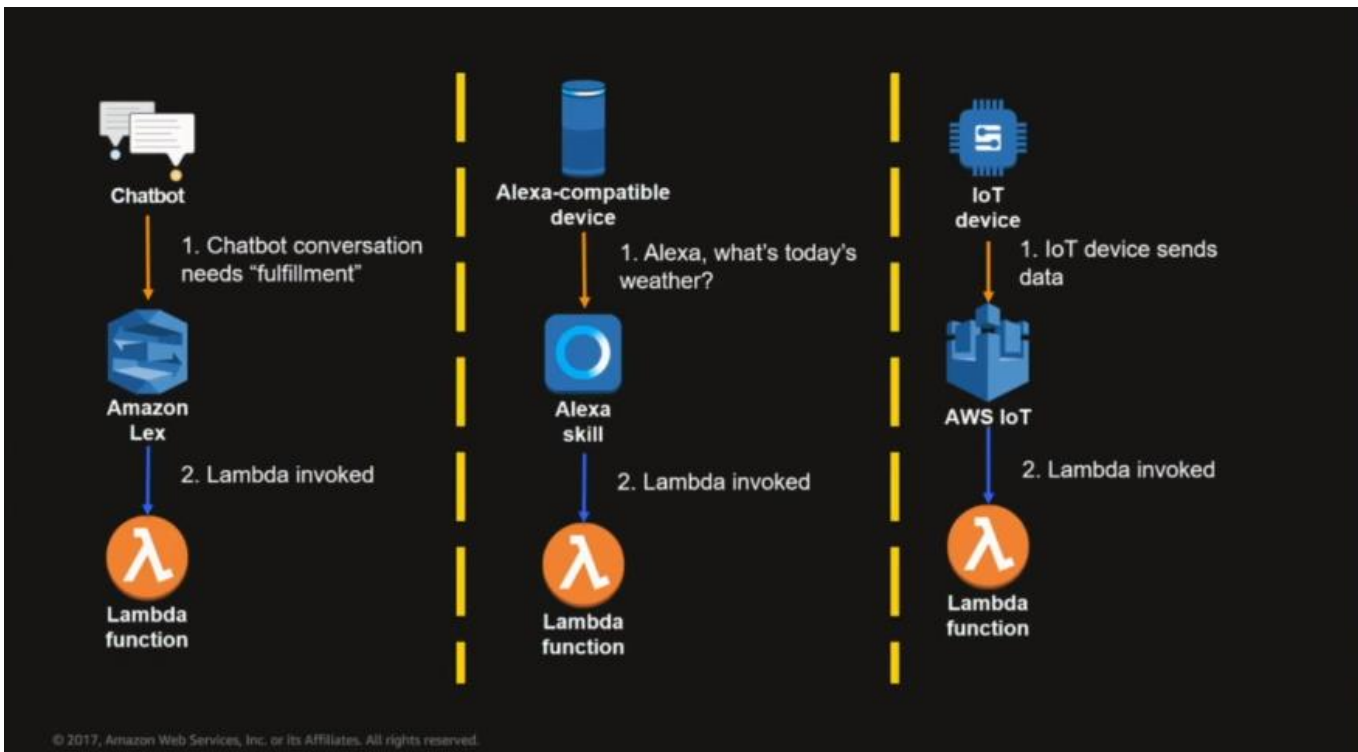
**Scheduled CloudWatch events are like managed Cron jobs.** You simply schedule a lambda function to execute at a specific time to do some work.



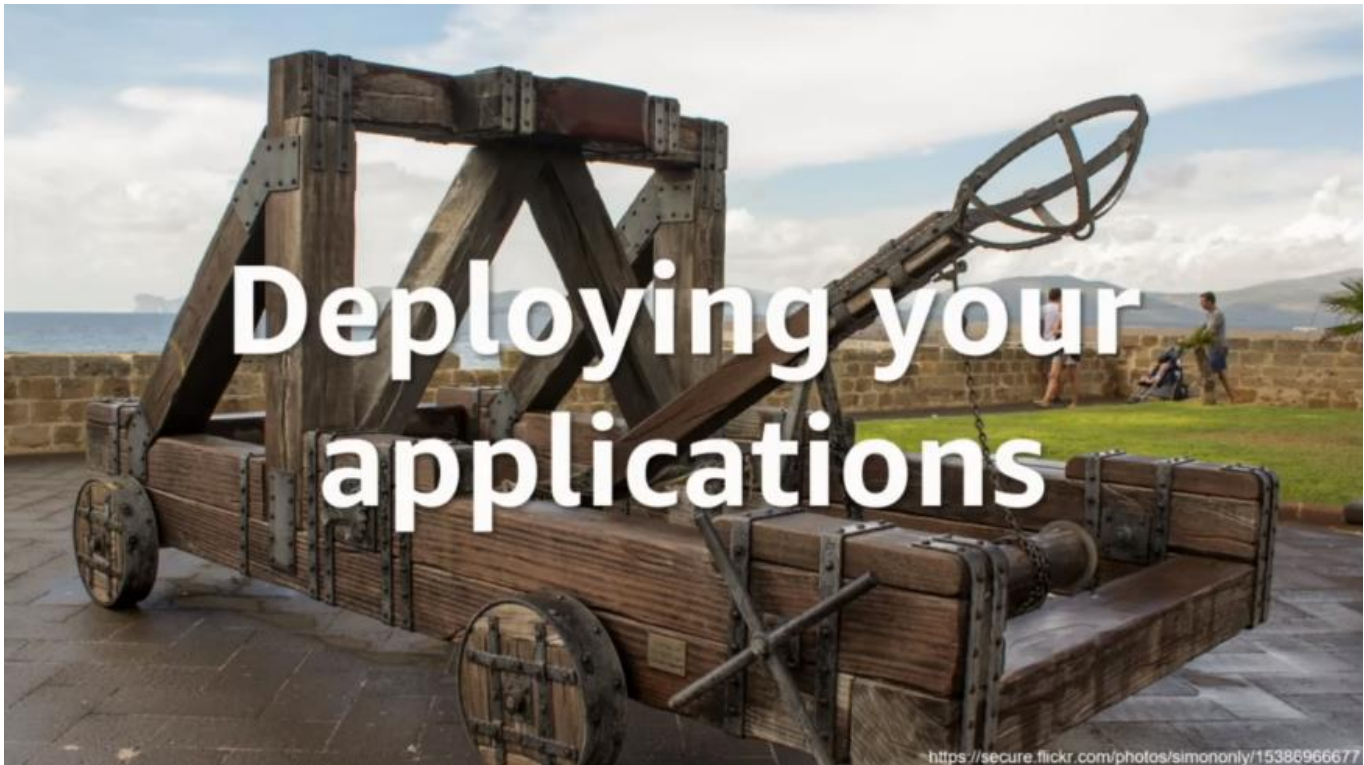
You can send logs coming into CloudWatch to a lambda to check for some values for real-time analysis.



These are some database use cases. The first can be a way to create backups of a DynamoDB table in another DynamoDB table in a different region.



These are some of the AI functionalities using lambda for chatbots, alexa, lex and IoT device implementations.



# AWS Serverless Application Model (SAM)



CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything CloudFormation supports

Open specification (Apache 2.0)

<https://github.com/awslabs/serverless-application-model>

## SAM template

```
{
  AWSTemplateFormatVersion: '2010-09-09'
  Transform: AWS::Serverless-2016-10-31
  Resources:
    GetHtmlFunction:
      Type: AWS::Serverless::Function
      Properties:
        CodeUri: s3://sam-demo-bucket/todo_list.zip
        Handler: index.gethtml
        Runtime: nodejs4.3
        Policies: AmazonDynamoDBReadOnlyAccess
        Events:
          GetHtml:
            Type: Api
            Properties:
              Path: /{proxy+}
              Method: ANY
    ListTable:
      Type: AWS::Serverless::SimpleTable
```

Tells CloudFormation this is a SAM template it needs to “transform”

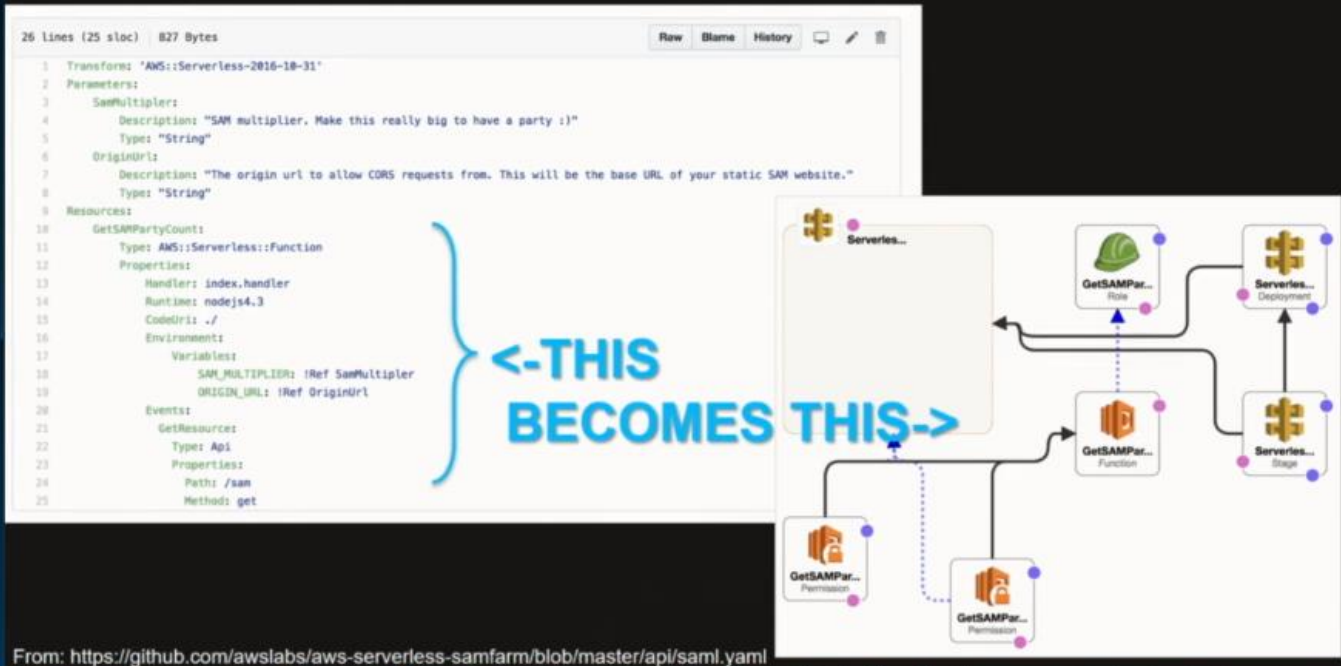
Creates a Lambda function with the referenced managed IAM policy, runtime, code at the referenced zip location, and handler as defined. Also creates an API Gateway and takes care of all mapping/permissions necessary

Creates a DynamoDB table with 5 Read & Write units

This is an example of a SAM template, there are 3 main parts to a SAM template shown above. Transforms are extension models in CloudFormation and the above transform is for Serverless that tells CF that this is a SAM template and it should look out for SAM resources.



# SAM template



## SAM Template Properties

**AWS::Serverless::Function**

**AWS::Serverless::Api**

**AWS::Serverless::SimpleTable**

Handler: index.js  
Runtime: nodejs4.3  
CodeUri: 's3://my-code-bucket/my-function.zip'  
Description: Creates thumbnails of uploaded images  
MemorySize: 1024  
Timeout: 15  
Policies: AmazonS3FullAccess  
Environment:  
 Variables:  
 TABLE\_NAME: my-table  
Events:  
 PhotoUpload:  
 Type: S3  
 Properties:  
 Bucket: my-photo-bucket  
Tracing: Active|PassThrough  
Tags:  
 AppNameTag: ThumbnailApp  
 DepartmentNameTag: ThumbnailDepartment

From SAM Version 2016-10-31

## AWS::Serverless::Function Event source types

S3

SNS

Kinesis | DynamoDB

Api

Schedule

CloudWatchEvent

IoTRule

AlexaSkill

Note: Events are a map of string to Event Source Object

Event Source Objects have the following structure:

Type:

Properties:

For Example:

Events:

MyEventName:

Type: S3

Properties:

Bucket: my-photo-bucket

From SAM Version 2016-10-31

## SAM Template Properties

AWS::Serverless::Function

**AWS::Serverless::Api**

AWS::Serverless::SimpleTable

StageName: prod

DefinitionUri: swagger.yml

CacheClusterEnabled: true

CacheClusterSize: 28.4

Variables:

VarName: varvalue

From SAM Version 2016-10-31

# SAM Template Properties

AWS::Serverless::Function

AWS::Serverless::Api

**AWS::Serverless::SimpleTable**

PrimaryKey:

Name: id

Type: String

ProvisionedThroughput:

ReadCapacityUnits: 5

WriteCapacityUnits: 5

From SAM Version 2016-10-31

## SAM commands – Package & Deploy

### Package

- Creates a deployment package (.zip file)
- Uploads deployment package to an Amazon S3 Bucket
- Adds a CodeUri property with S3 URI

### Deploy

- Calls CloudFormation 'CreateChangeSet' API
- Calls CloudFormation 'ExecuteChangeSet' API

# Introducing SAM Local

CLI tool for local testing of serverless apps

Works with Lambda functions and “proxy-style” APIs

Response object and function logs available on your local machine

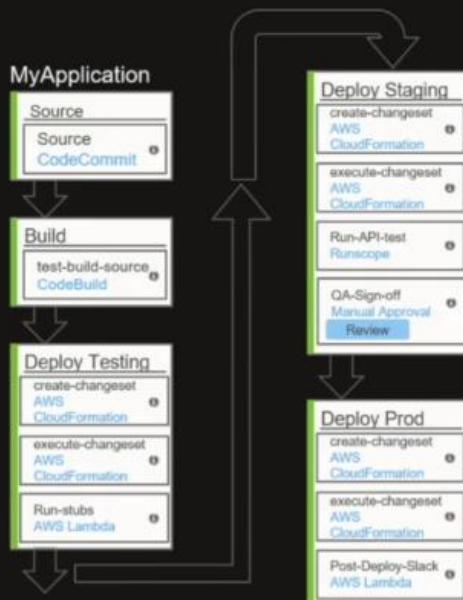
Uses open source docker-lambda images to mimic Lambda’s execution environment:

- Emulates timeout, memory limits, runtimes



<https://github.com/aws-labs/aws-sam-local>

## An example minimal production pipeline:

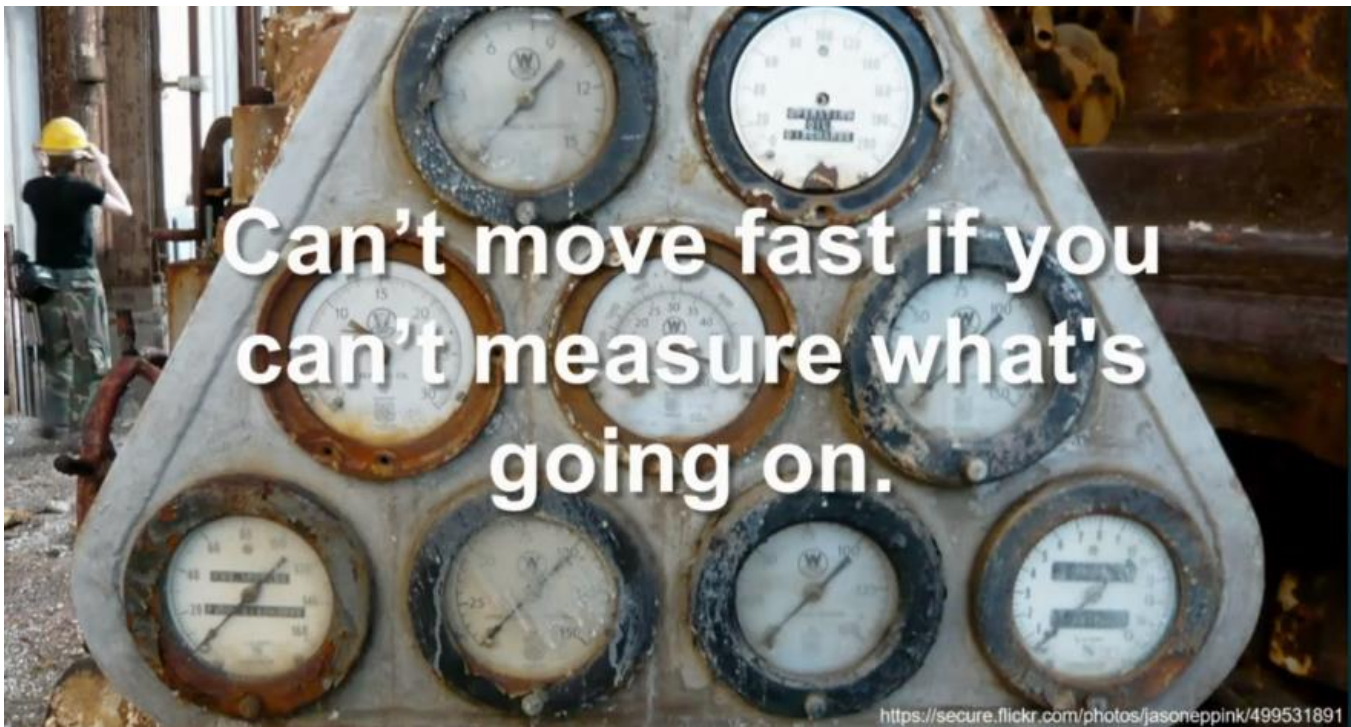


### This pipeline:

- Five Stages
- Builds code artifact
- Three deployed to “Environments”
- Uses SAM/CloudFormation to deploy artifact and other AWS resources
- Has Lambda custom actions for running my own testing functions
- Integrates with a 3<sup>rd</sup> party tool/service
- Has a manual approval before deploying to production







## Metrics and logging are a universal right!

### CloudWatch Metrics:

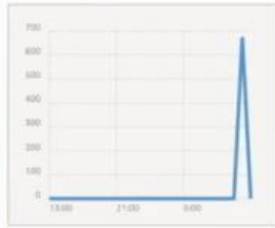
- 6 Built in metrics for Lambda
  - Invocation Count, Invocation duration, Invocation errors, Throttled Invocation, Iterator Age, DLQ Errors
  - Can call "[put-metric-data](#)" from your function code for custom metrics
- 7 Built in metrics for API-Gateway
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, Cache Hit Count, Cache Miss Count
  - Error and Cache metrics now support [averages](#) and [percentiles](#)



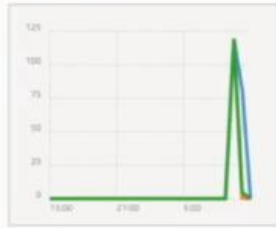
CloudWatch metrics at a glance (last 24 hours) 

[View logs in CloudWatch](#)

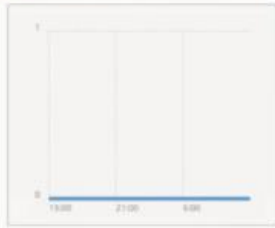
Invocation count 



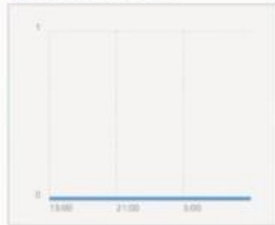
Invocation duration 



Invocation errors 



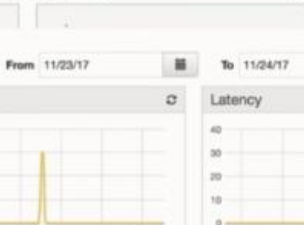
Throttled invocations 



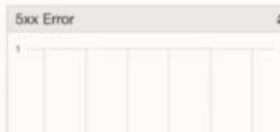
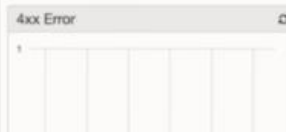
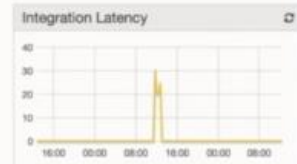
Iterator age 



DLQ errors 



Stage: Prod From: 11/23/17 To: 11/24/17



## Metrics and logging are a universal right!

### CloudWatch Logs:

- **API Gateway Logging**
  - 2 Levels of logging, ERROR and INFO
  - Optionally log method request/body content
  - Set globally in stage, or override per method
- **Lambda Logging**
  - Logging directly from your code with your language's equivalent of `console.log()`
  - Basic request information included
- **Log Pivots**
  - Build metrics based on log filters
  - Jump to logs that generated metrics
- **Export logs to AWS ElastiCache or S3**
  - Explore with Kibana or Athena/QuickSight



# Amazon API Gateway Custom Access Logging!

NEW!

Settings Logs Stage Variables SDK Generation Export Deployment History Documentation History

Configure the metering and caching settings for the stage.

## CloudWatch Settings

Enable CloudWatch Logs ☐ ⓘ

Enable Detailed CloudWatch Metrics ☐ ⓘ

## Custom Access Logging

Enable Access Logging ☒

CloudWatch Group

Log Format { "requestId": "\$context.requestId", "ip": "\$context.identity.sourceIp", "caller": "\$context.identity.caller", "user": "\$context.identity.user", "requestTime": "\$context.requestTime", "httpMethod": "\$context.httpMethod", "resourcePath": "\$context.resourcePath", "status": "\$context.status", "protocol": "\$context.protocol", "responseLength": "\$context.responseLength" }

Insert Example:

[List of Log Variables](#)

```
{ "requestId": "da82692e-ce4e-11e7-ada2-6d3cd4a95886",  
  "ip": "54.240.196.186",  
  "caller": "-",  
  "user": "-",  
  "requestTime": "20/Nov/2017:23:59:32 +0000",  
  "httpMethod": "GET",  
  "resourcePath": "/pets",  
  "status": "200",  
  "protocol": "HTTP/1.1",  
  "responseLength": "3"  
}
```

Log Example

Save Changes

SAMDemo01

Add widget

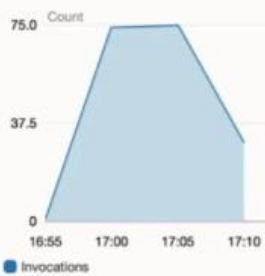
Actions

Save dashboard

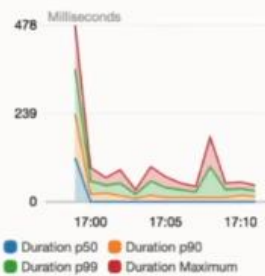
1h 3h 12h 1d 3d 1w custom (15m)

Refresh

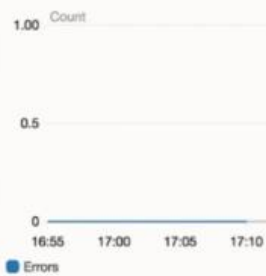
## Invocations



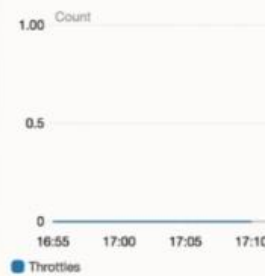
## Duration



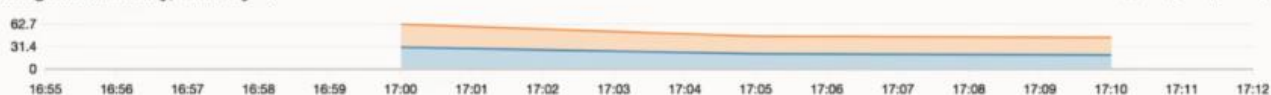
## Errors



## Throttles



## IntegrationLatency, Latency



## 4XXError, 5XXError



## Count



## SAM Lambda Alarm



# AWS X-Ray



This is a Profiling and Tracing service that captures all requests in your applications

## Application instrumentation (Node.js)

```
1 var AWSXRay = require('aws-xray-sdk-core');
2 var AWS = AWSXRay.captureAWS(require('aws-sdk'));
3 s3 = new AWS.S3({signatureVersion: 'v4'});
4
5 exports.handler = (event, context, callback) => {
6
7     var params = {Bucket: 'tim-example-bucket', Key: 'MyKey', Body: 'Hello!'};
8
9     s3.putObject(params, function(err, data) {});
10 };
```

**re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

**aws**

Within lambda, you can flag on the ability to capture X-Ray mode.



# X-Ray Service Map

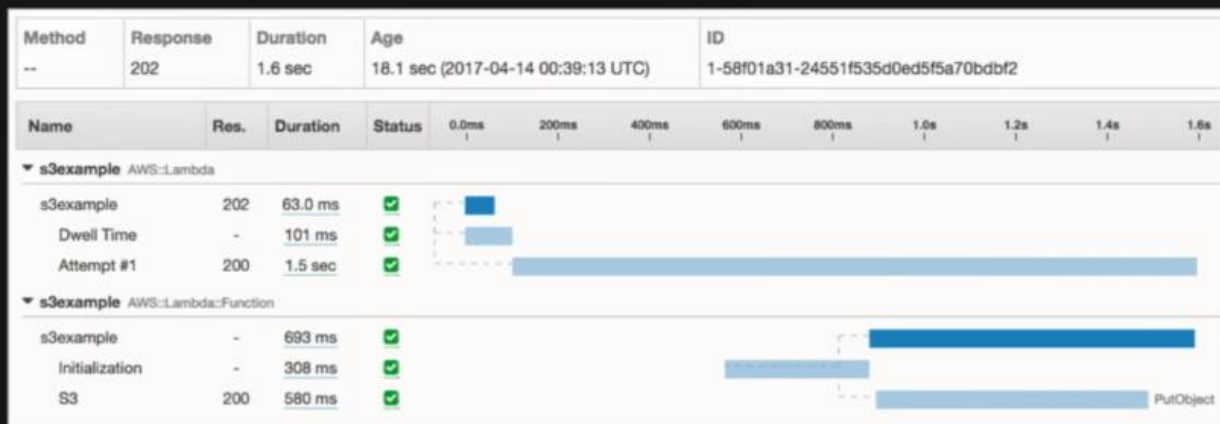


AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



# X-Ray Trace Timeline



AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



# Chris, how do I figure out what's wrong?

## These tools are here, so use them!

### 1. Turn on X-Ray now

1. look at wrapping your own calls with it via the X-Ray SDKs

### 2. Don't underestimate the power of logging in Lambda

1. Simple "debug: in functionX" statements work great and are easy to find in CloudWatch Logs

### 3. The most valuable metrics are the ones closest to your customer/use-case

1. How many gizmos did this function call/create/process/etc

## Lambda Dead Letter Queues

"By default, a failed Lambda function **invoked asynchronously is retried twice**, and then the event is discarded. Using Dead Letter Queues (DLQ), you can indicate to Lambda that unprocessed events should be sent to an Amazon SQS queue or Amazon SNS topic instead, where you can take further action." – <https://docs.aws.amazon.com/lambda/latest/dg/dlq.html>

- **Turn this on!** (for async use-cases)
- Monitor it via an **SQS Queue length metric/alarm**
- If you use SNS, send the messages to something durable and/or a trusted endpoint for processing
  - Can send to Lambda functions in other regions
- If and when things go **"boom"** DLQ can save your invocation event information



# FIN, ACK

## There are a lot of different use-cases for AWS Lambda!

- From full-fledged application backends to “glue” functions attached to operational tasks
- Removes the need to run hosts for small scripts such as cron jobs and small web services, with many benefits:
  - Reduced cost
  - Reduced maintenance overhead
  - No capacity planning needed for potential spikes in usage
  - Security model that allows for finely scoped access and permissions
- Use AWS SAM (Serverless Application Models) to deploy!
- Can get started right in the console or via AWS CodeStar!



## aws.amazon.com/serverless

 Menu



Products ▾

Solutions

Pricing

Software

Support

Customers

More ▾

English ▾

My Account ▾

Sign In to the Console



Serverless Computing and Applications

Build and run applications without thinking about servers

Get Started

AWS Lambda

Getting Started Resources

Use Cases

Developer Tools

Partner Solutions

Compute Blog

### Build Serverless Applications for Production

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for virtually any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great

SRV213R

**AWS**  
**re:Invent**

**Thank you!**

Remember to review this session!

**AWS**  
**re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

