



Raj Saha  
[cloudwithraj.com](http://cloudwithraj.com)

▶ Cloud With Raj

---

Instructor Bio:

Sr. Solutions Architect @ 

Published Udemy/Pluralsight author

Public speaker

Author of multiple AWS official blogs

Previously - Distinguished Cloud Architect @Verizon

*Opinions are my own*

# How This Course is Structured

Getting Started

API Gateway

Lambda Advanced  
Concepts

Step Functions

Logging & Monitoring

AWS CLI

AWS Cloud9

Serverless Security

Storage For Serverless

Real World Project

DevOps For Serverless

AWS SAM

Serverless Frameworks

Serverless Vs Containers

Architectures &  
Optimization

# Traditional Server Cloud VM Serverless

**Scenario** - Everyday, dinner party at your home, number of guests could be between 1 to 20, no one RSVPs!

## Traditional Server in your Datacenter

- You own the kitchen
- You spend money buying every appliance
- You pay for the electricity used in the kitchen
- You make food for 20 people everyday
- **Lot of wasted food**

## Cloud VM (EC2)

- You do NOT own kitchen
- You get food from a takeout place who delivers instantly
- However, takeout place only delivers food exactly for 5 people at a time
- If 3 people show up, you waste food of 2 people
- If 7 people show up, you place 2 orders, each for 5 people, still some waste
- Better than Traditional Datacenter but still little wasteful

## Serverless (Lambda)

- You do NOT own kitchen
- You get food from a takeout place who delivers instantly
- However, this takeout place accepts order for any number of people
- You simply put order for the exact number of people showed up
- **Best cost optimized solution**

# Traditional Server Cloud VM Serverless

**Scenario** - Everyday, traffic hits your website, you don't know how much traffic going to hit each day

## Traditional Server in your Datacenter

- You own the datacenter building
- You spend money buying every server
- You pay for the cost of datacenter i.e. electricity, AC etc.
- You buy enough servers to accommodate for huge traffic
- **Lot of wasted \$\$\$**

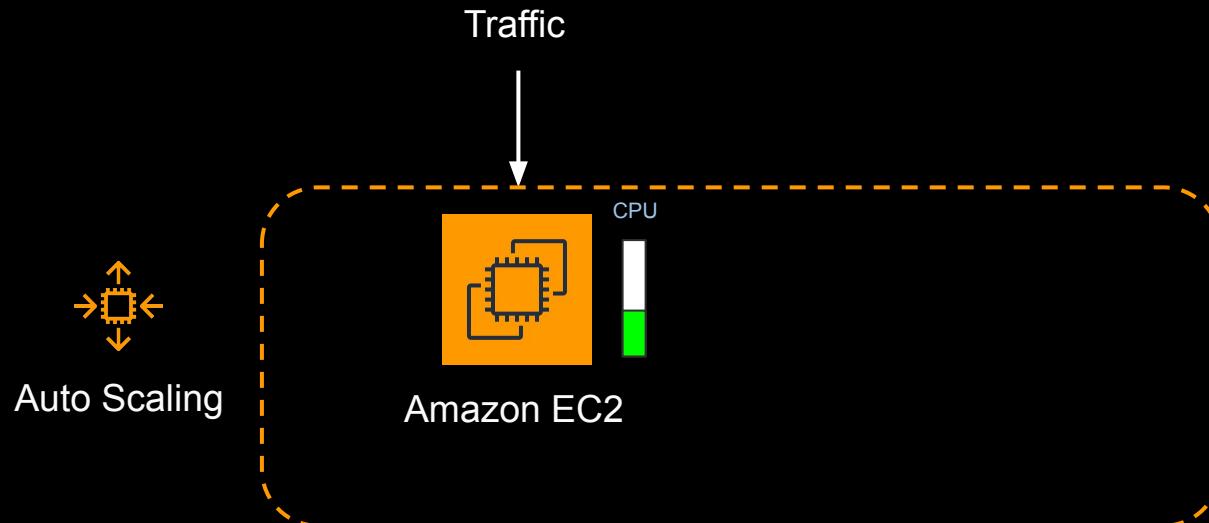
## Cloud VM (EC2)

- You do NOT own the datacenter
- You provision EC2
- However, each EC2 comes with fixed processing power and memory
- Sometimes traffic would be less than EC2 capacity
- If EC2 reaches capacity, add another EC2 via Auto Scaling Group, but with fixed predetermined capacity
- **Better than Traditional Datacenter but still little wasteful**

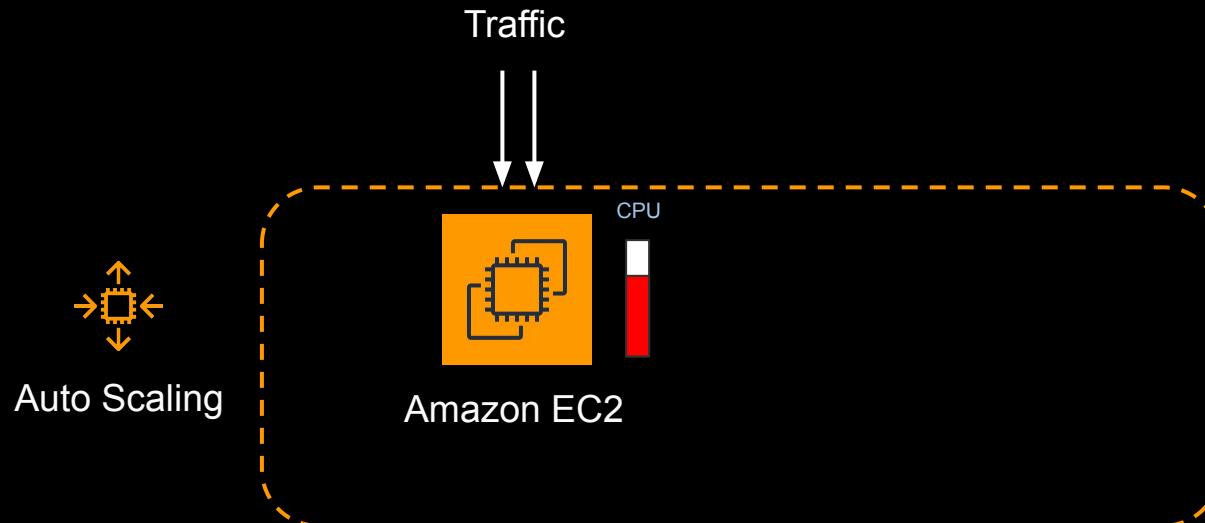
## Serverless (Lambda)

- You do NOT own the datacenter
- You utilize Serverless Services
- If more traffic hits, it auto scales automatically
- You pay for the number of executions rather than idle resources
- **Best cost optimized solution**

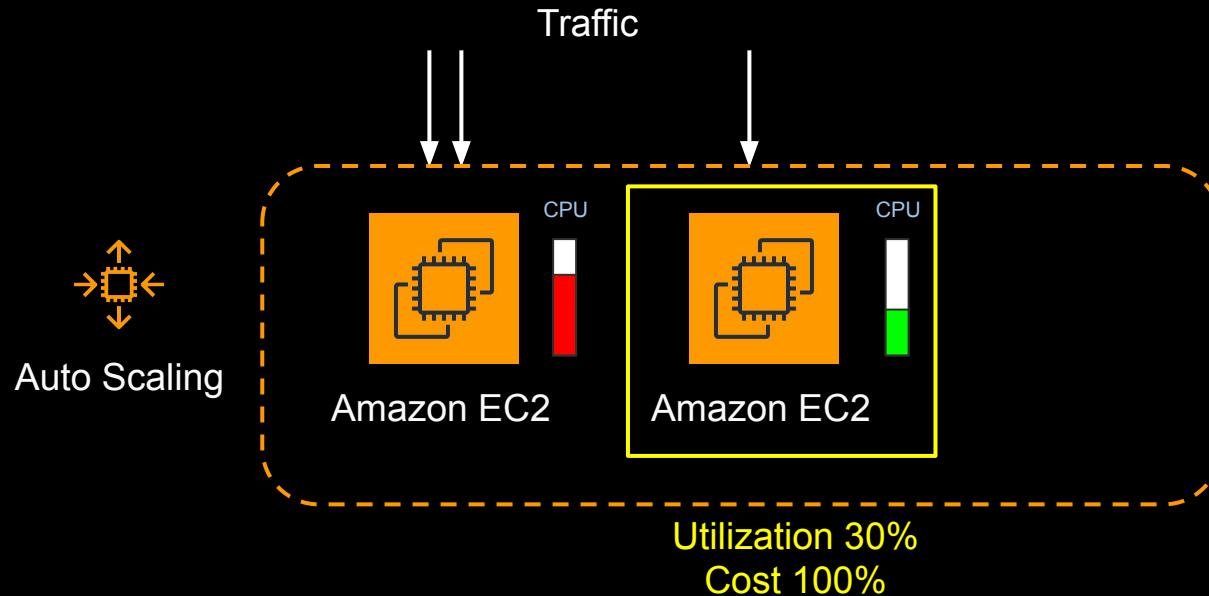
# Cloud VM (EC2)



# Cloud VM (EC2)



# Cloud VM (EC2)



# Serverless (Lambda)

Traffic



AWS Lambda

# Serverless (Lambda)

Traffic



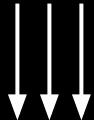
AWS Lambda



AWS Lambda

# Serverless (Lambda)

Traffic



AWS Lambda



AWS Lambda



AWS Lambda

Pay for what you use  
(3 Invocations)

# Lambda Logging & Monitoring



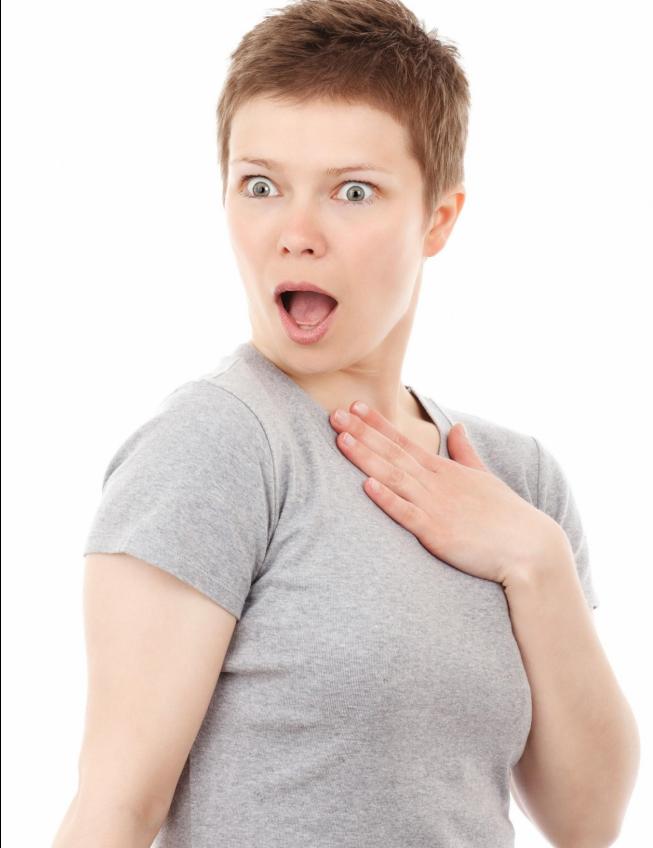
AWS Lambda



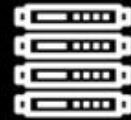
Amazon CloudWatch

- Logs
- Metrics
- CloudWatch Insights

# Serverless Is NOT Only Lambda



# What Defines Serverless?



No servers to provision or manage



Automatically scales with usage



Never pay for idle



Availability and fault tolerance built in

# Serverless means...



**No AMI Rehydration**

# Serverless Ecosystem - Beyond Lambda

## Compute



AWS Lambda



AWS Fargate

## Storage



Aurora Serverless



DynamoDB



Amazon S3

## Integration, Analytics



Amazon API  
Gateway



Amazon SQS



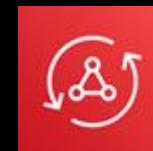
AWS Step  
Functions



AWS Glue



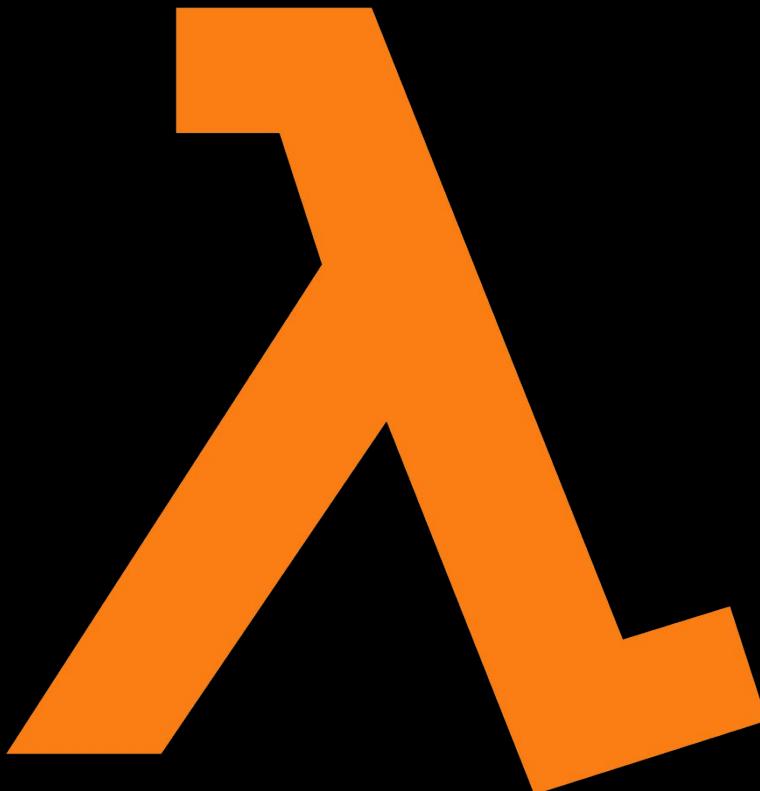
Amazon SNS



AWS AppSync

And More...

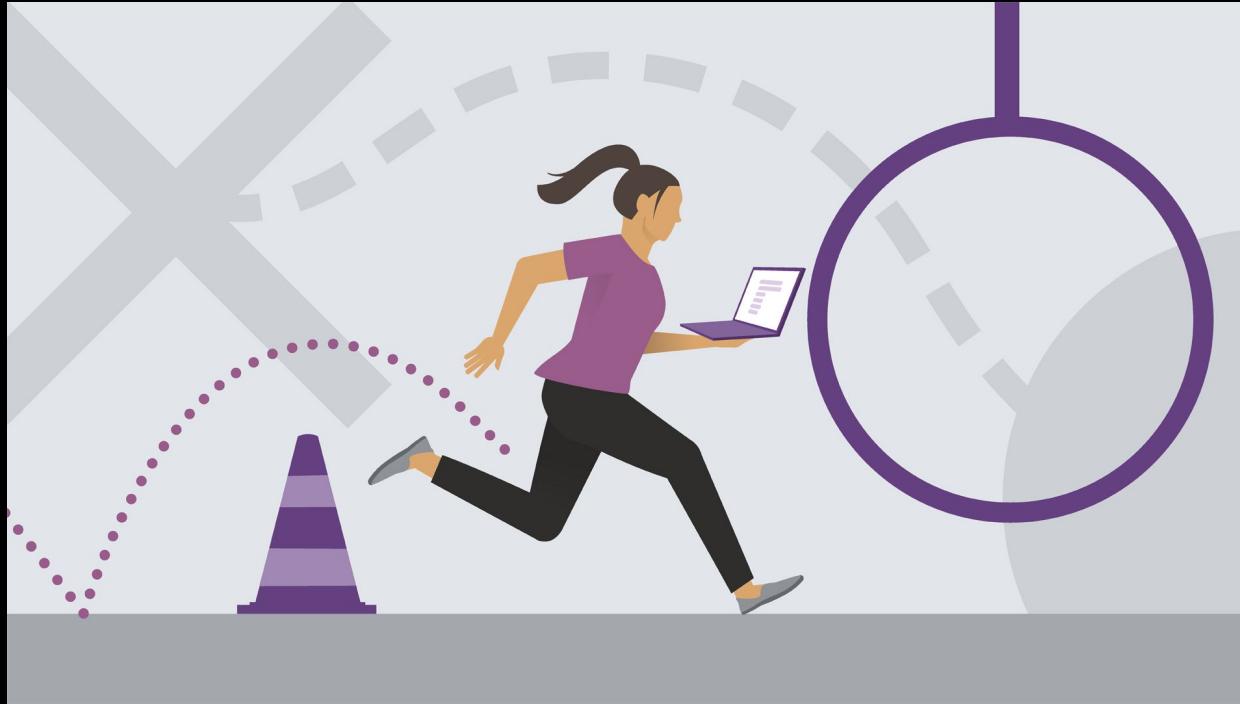
# Crown Jewel of Serverless



# What is AWS Lambda?

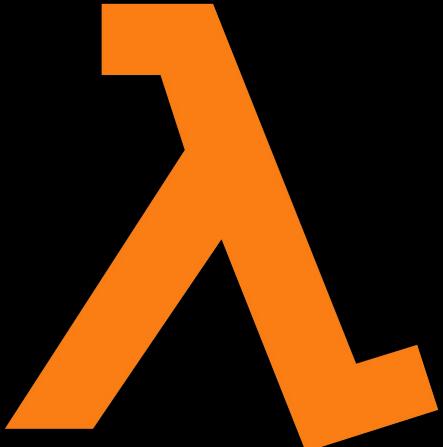
- AWS Lambda lets you run code without provisioning or managing servers
- With Lambda, you can run code for virtually any type of application or backend service - all with zero administration
- Just upload your code and Lambda takes care of everything required to run and scale your code with high availability
- You pay only for the compute time you consume

# What Does This Buy You?



Speed, Agility, and Innovation

# Some More Details About Lambda



Configuration

- Select Memory from 128 MB to 10 GB  
(reInvent 2020)
- CPU and Network allocated proportionally
- Max 15 Mins Runtime



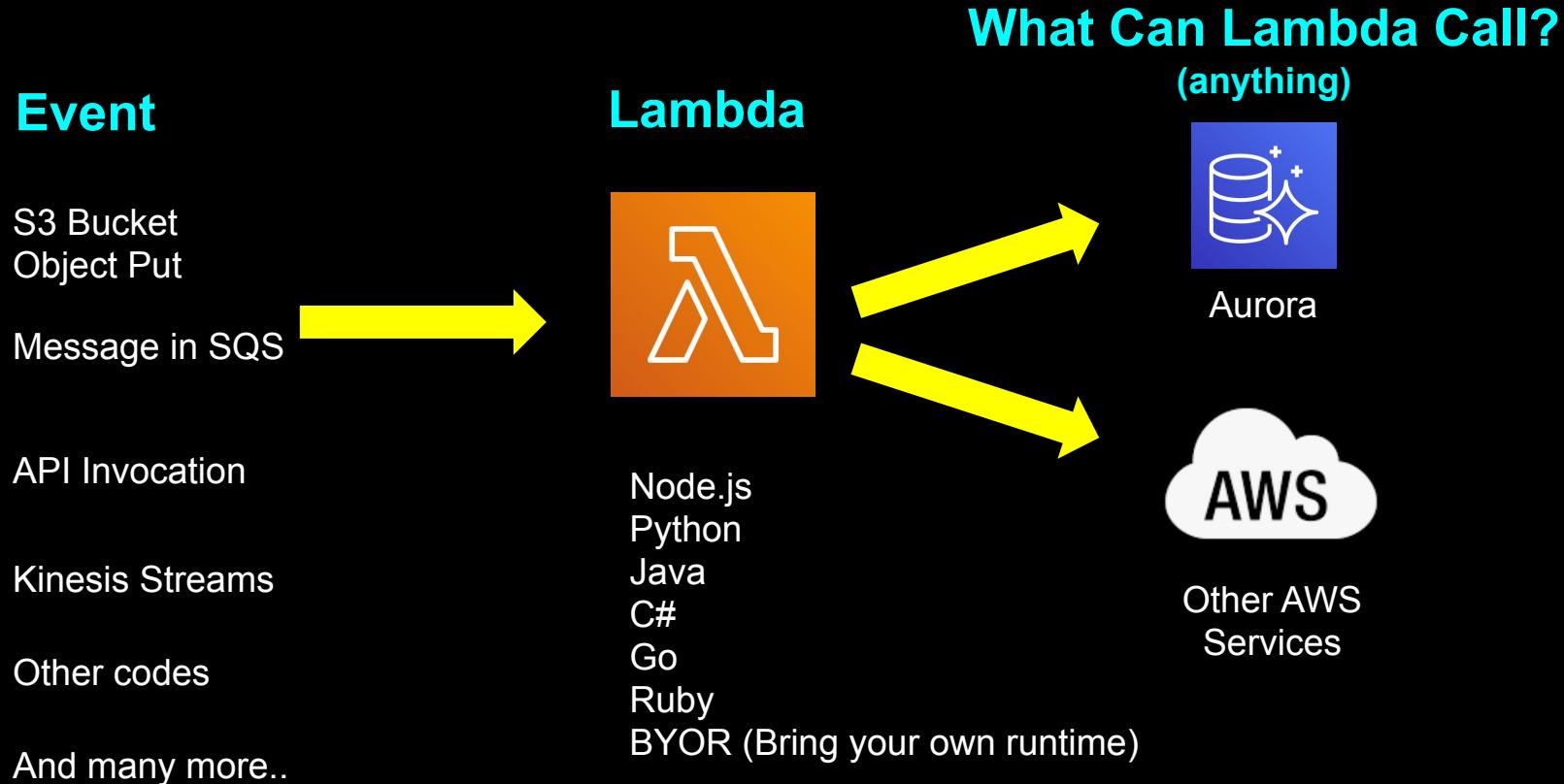
Architectural  
Flexibility

- Can be invoked synchronous (e.g. API) or asynchronous (e.g. SQS, S3)
- Inherent integration with other AWS services
- Wide range of use cases

# You Use Lambda All the Time!



# Serverless applications



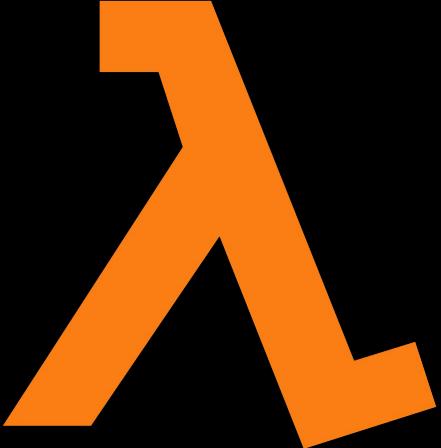


**“We don’t believe in one tool to rule the world. We want you to use the right tool for the right job.”—Andy Jassy, CEO of AWS**

# Cost Of Lambda



# Non Expiring Free Tier!!



- Every month you get 1M invocations and 400,000 GBs of compute
- Charged in 1 ms increments (reInvent 2020)
- No commitment required
- Never pay for idle

All demos covered under Free-Tier\*

\*Except Advanced API Logging

# How Do You Calculate for Real Projects!

AWS Lambda Pricing Calculator

**Number of Executions**   
Enter the number of times your Lambda function will be called per month

**Allocated Memory (MB)**  ▼  
Enter the allocated memory for your function

**Estimated Execution Time (ms)**   
Enter how long you expect the average execution will take in milliseconds

**Include Free Tier**  Yes  No

**TOTAL COSTS**  
Request Costs: \$0.40  
Execution Costs: \$11.67  

---

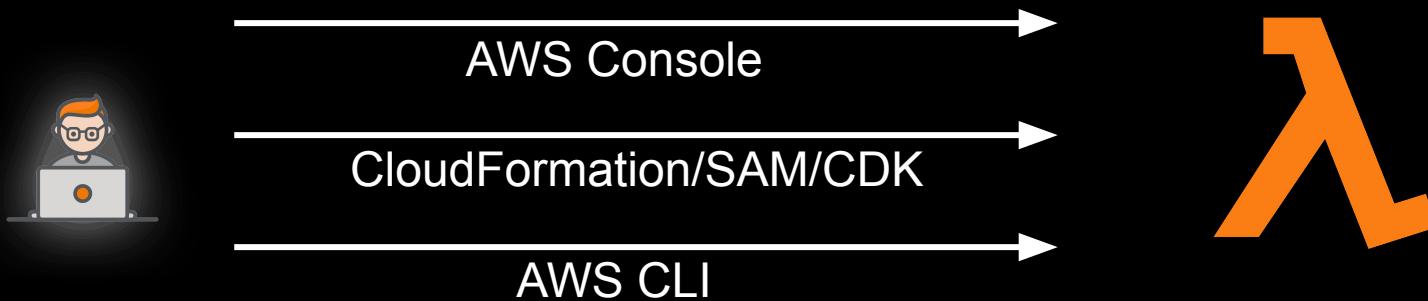
\$12.07/month

# Let's Calculate!

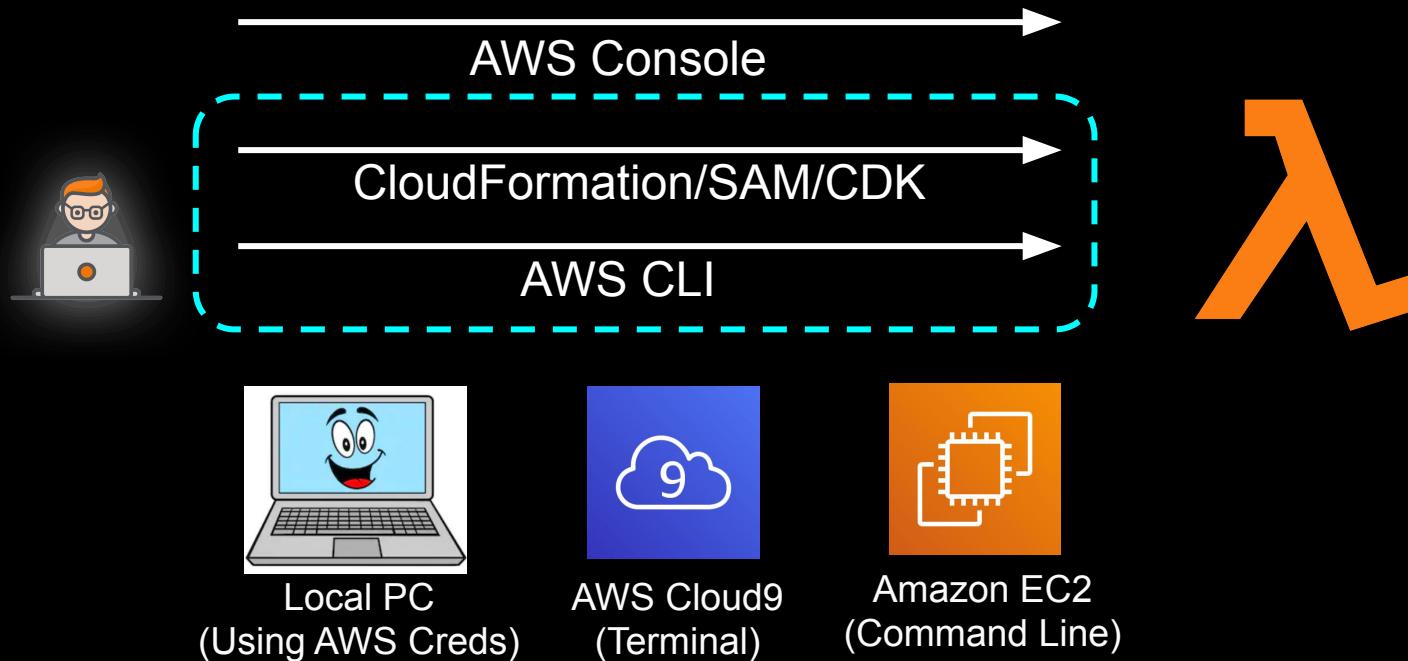
<https://s3.amazonaws.com/lambda-tools/pricing-calculator.html>

# Ways To Create Lambda

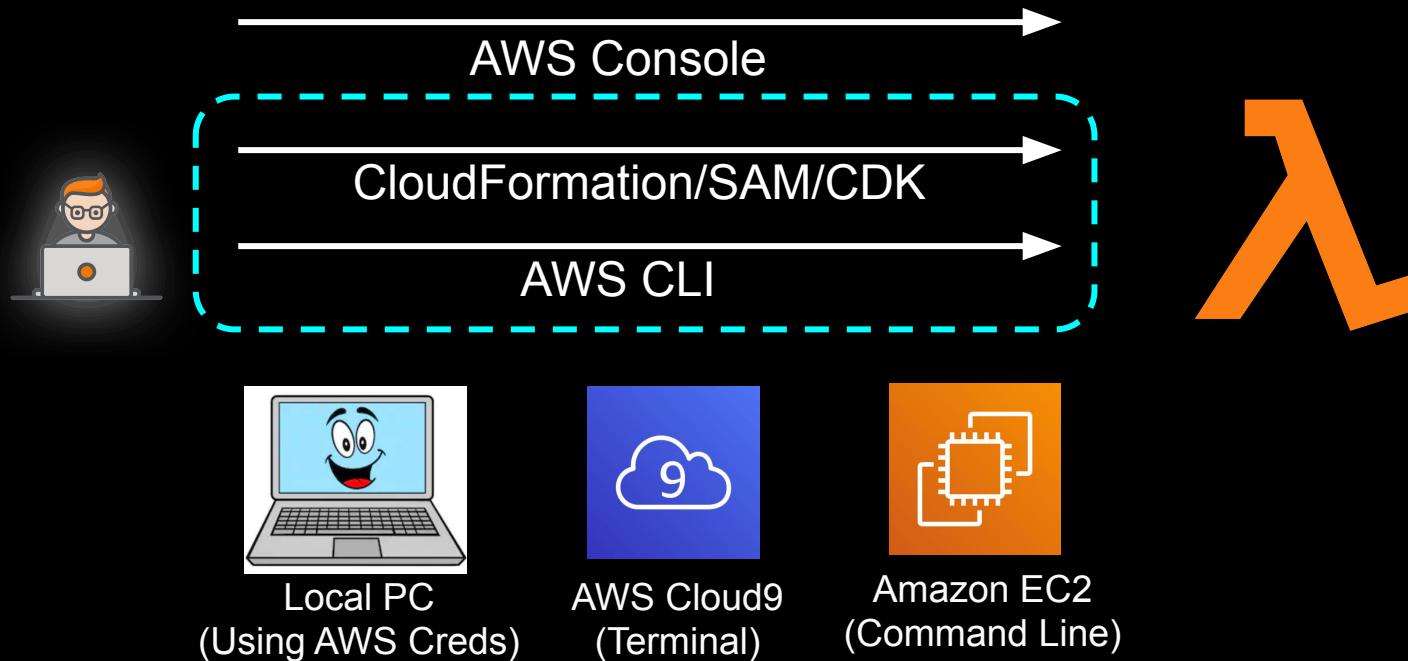
# Ways To Create Serverless



# Learning Medium



# Learning Medium



# IAM Roles

# Role Based Access

AWS Account #1234

Admin

Have full access to any service

Developer

Read, write, execute selected services



Lambda



Dynamo



S3

Tester

Execute selected services



Lambda

# Role Based Access

AWS Account #1234

Admin (Tina)

Have full access to any service

Developer (Bob)

Read, write, execute selected services



Lambda



Dynamo



S3

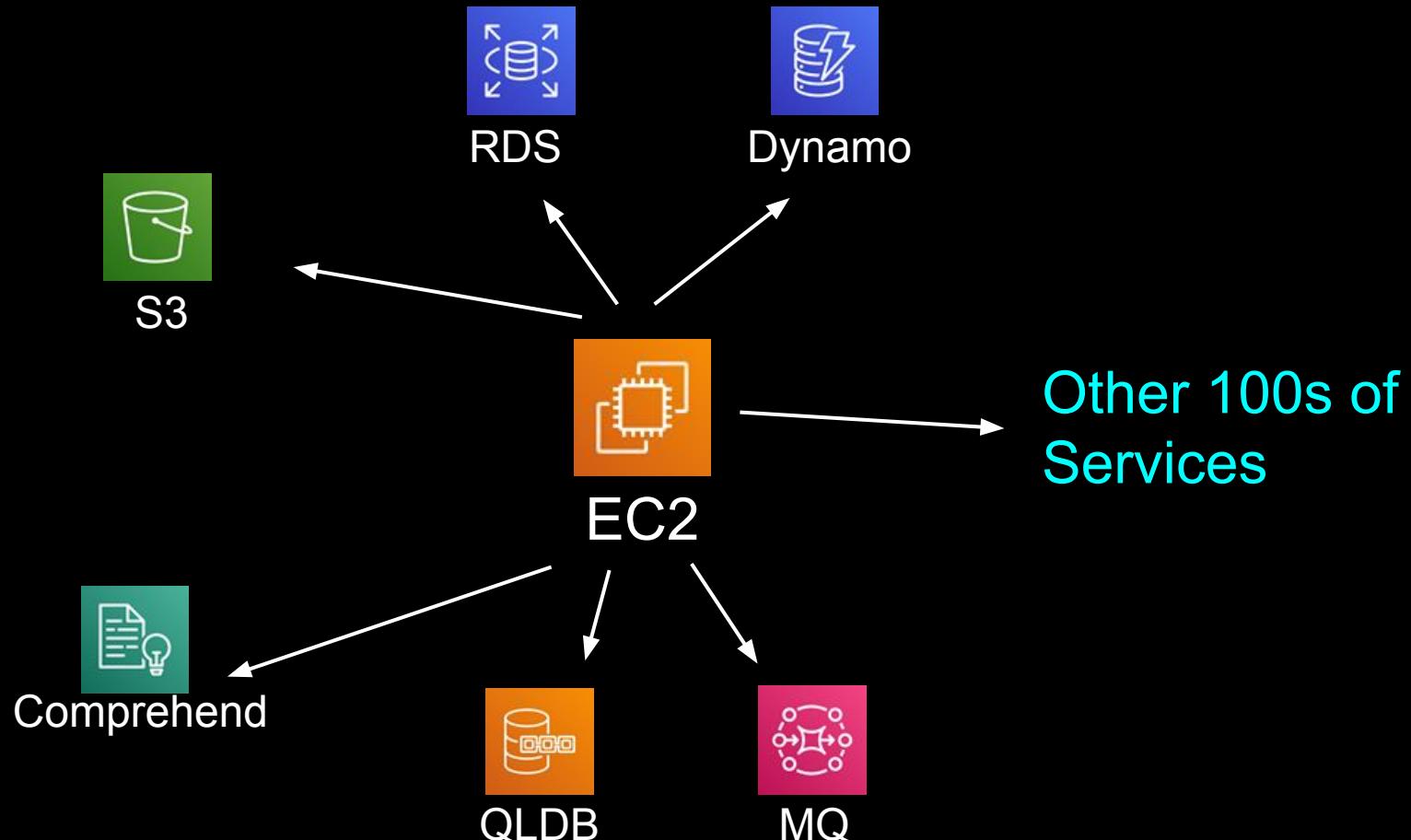
Tester (Susan)

Execute selected services

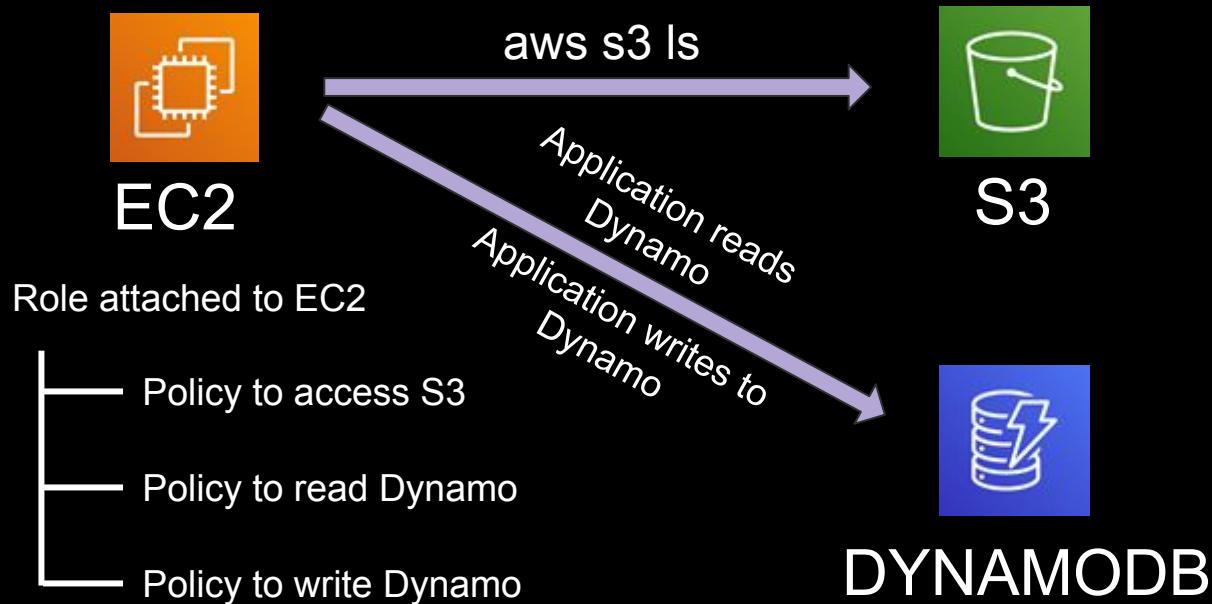


Lambda

# Why IAM Role?

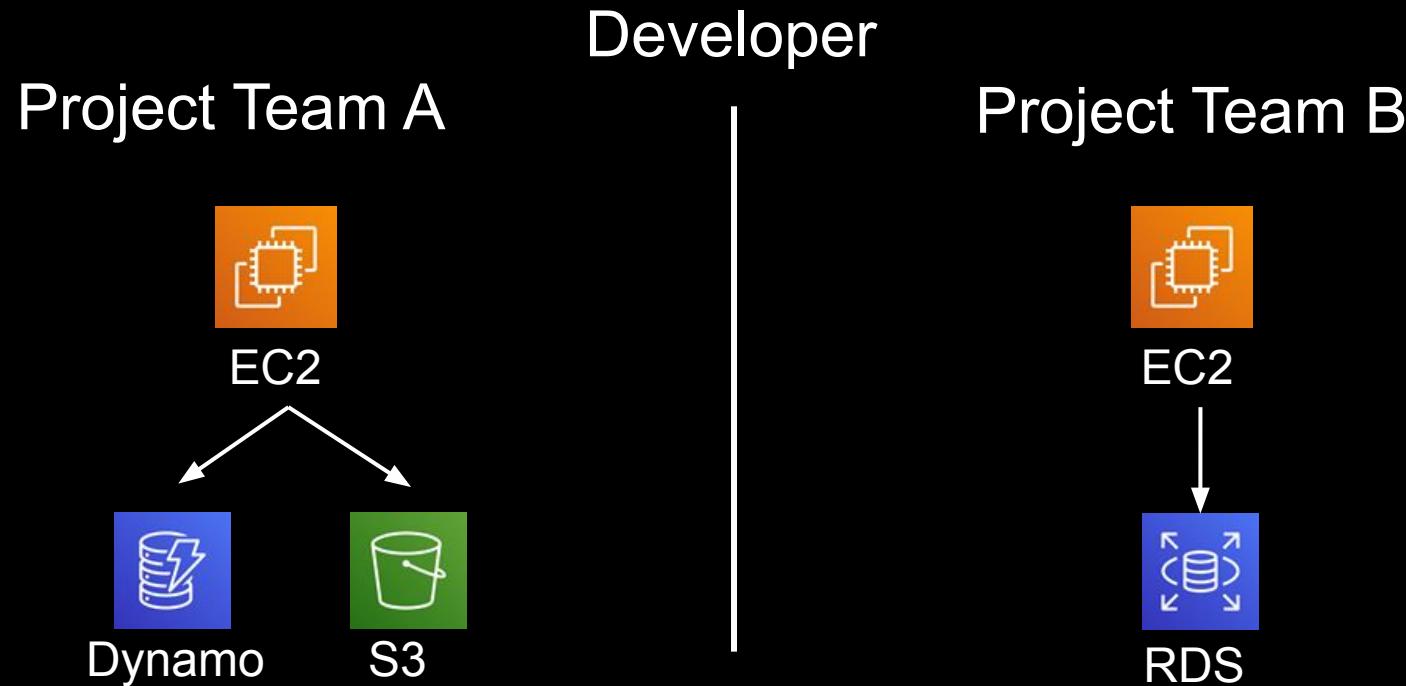


# IAM Roles



# Role Based Access

AWS Account #1234



# Is This Service Serverless

# Let's Test The 4 Rules

- ✓ No servers to provision or manage
- ✓ Automatically scales with usage
- ✓ Never pay for idle
- ✓ Availability and fault tolerance built in



Amazon SQS

Serverless

# Let's Test The 4 Rules



No servers to provision or manage



Automatically scales with usage



Never pay for idle



Availability and fault tolerance built in



Amazon Kinesis

Not  
Serverless

# Let's Test The 4 Rules

- ✗ No servers to provision or manage
- ✗ Automatically scales with usage
- ✗ Never pay for idle
- ✗ Availability and fault tolerance built in



Amazon EC2

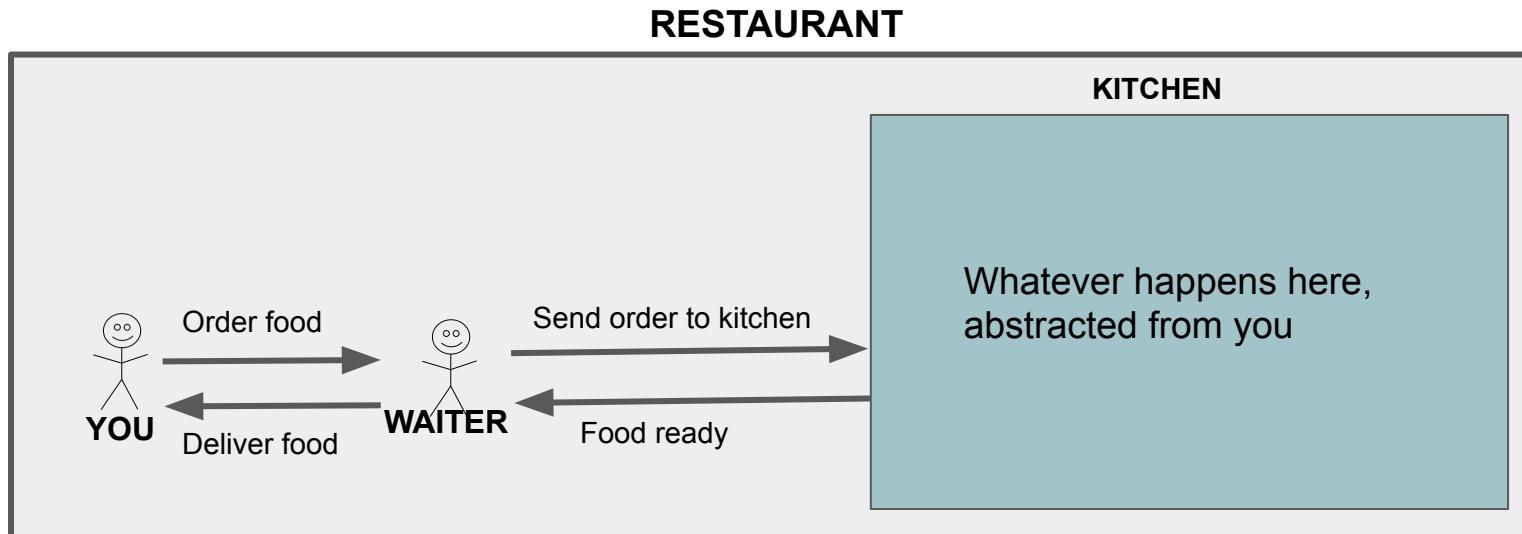
Not  
Serverless

# Understanding API with Real World Example

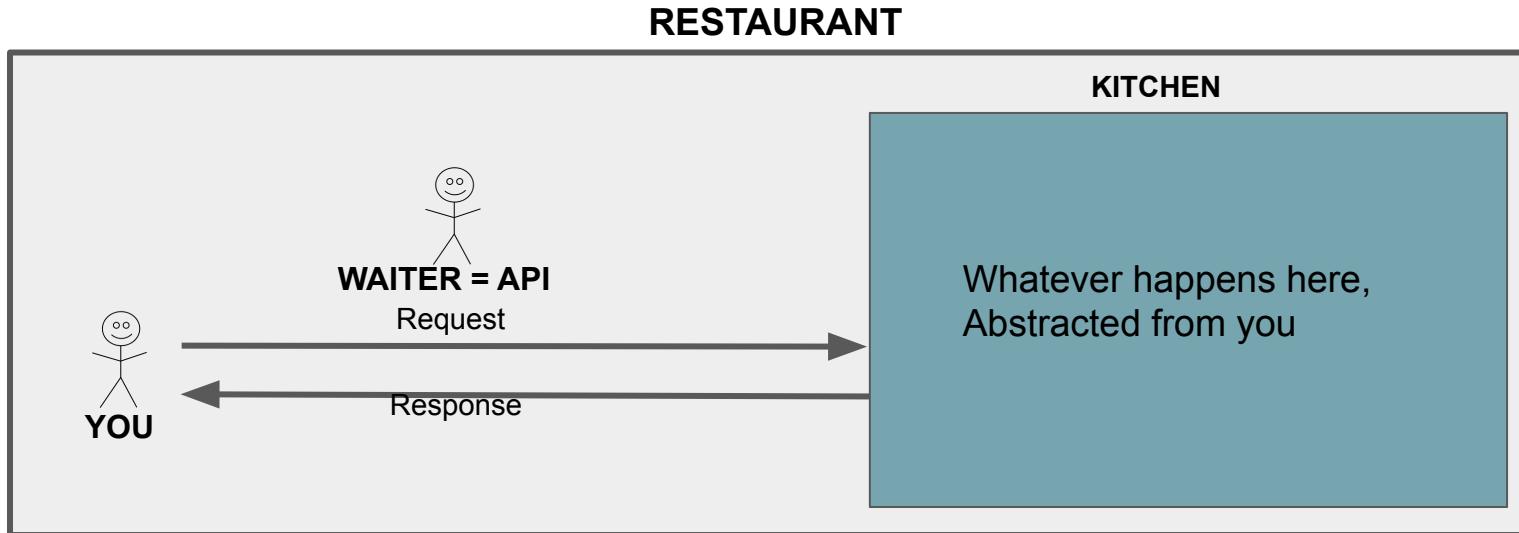
# What is API?

## Wikipedia Definition

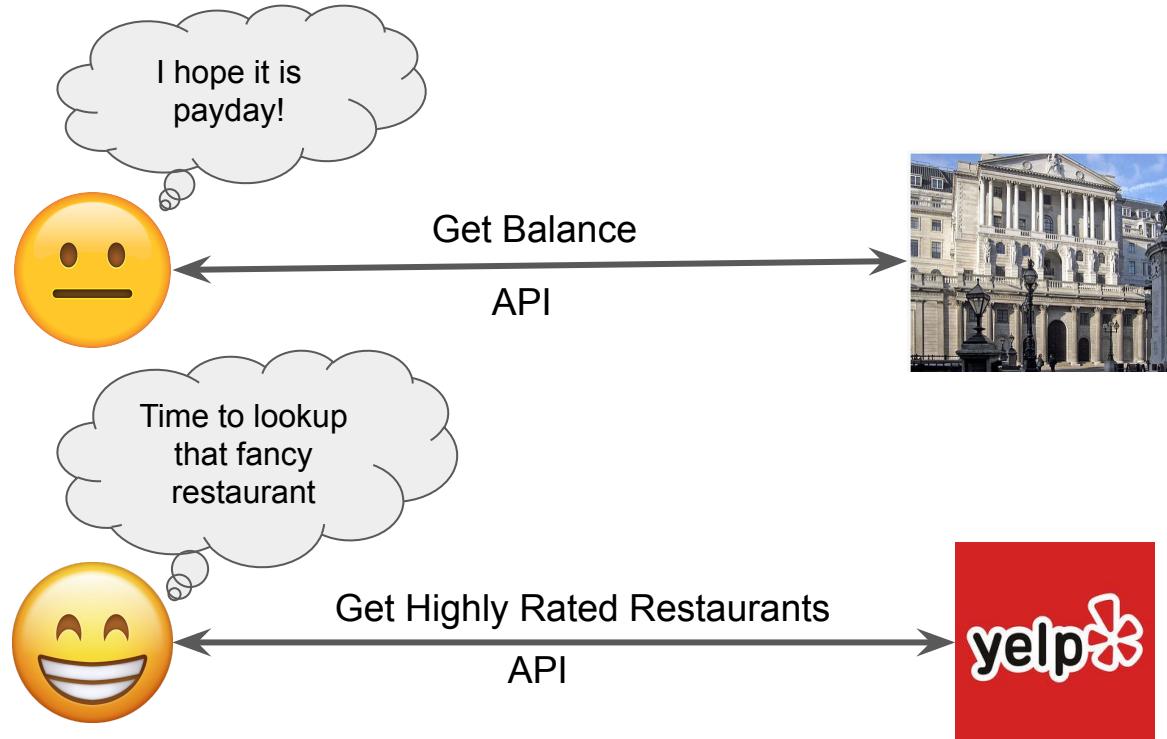
In [computer programming](#), an **application programming interface (API)** is a set of subroutine definitions, [communication protocols](#), and tools for building software. In general terms, it is a set of clearly defined methods of **communication** between various components.



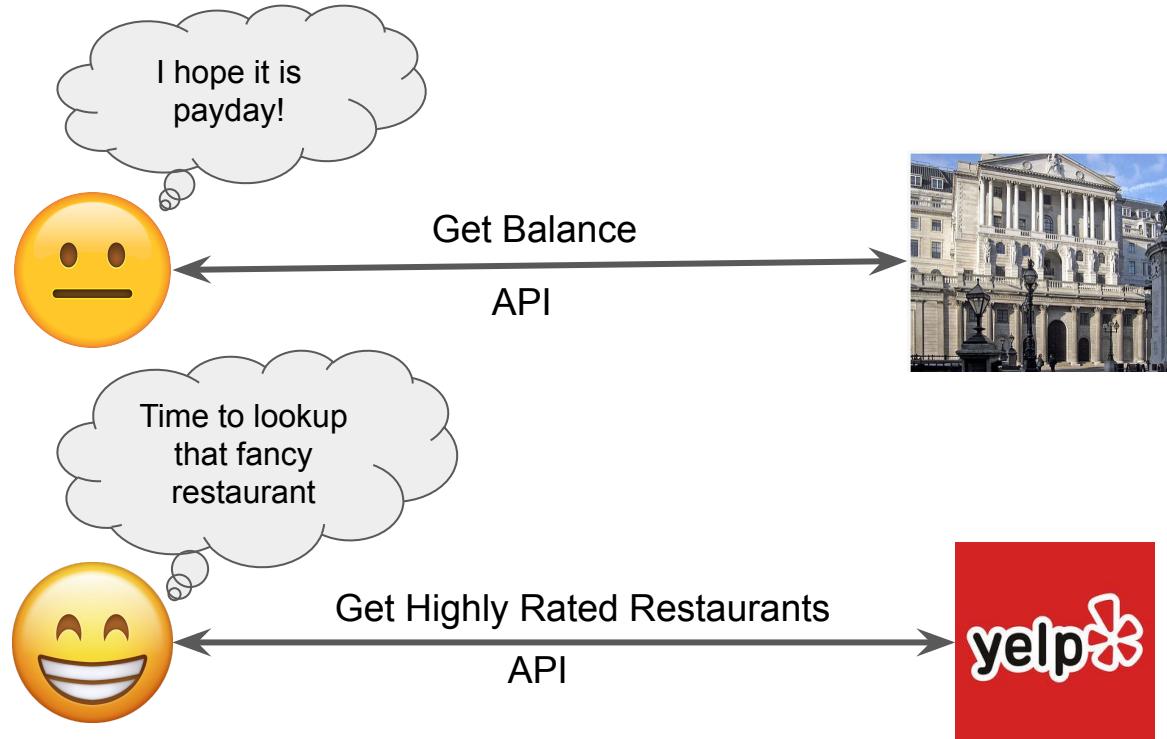
# What is API?



# Real life example of APIs



# Real life example of APIs



# Real life example of APIs

Book Taxi

Look Up Flight Deals

Browse Items in Online Store

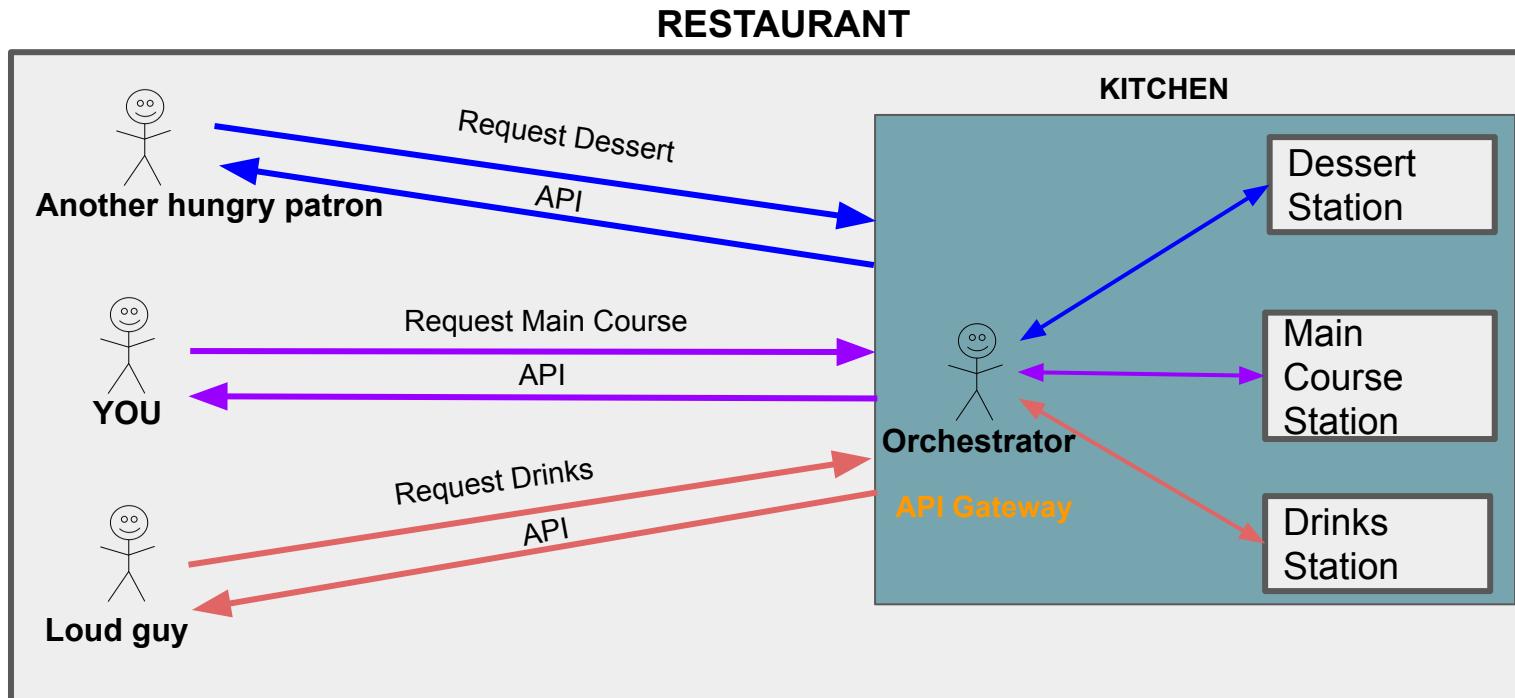
## **APIs are everywhere!**

### **Going back to wikipedia definition**

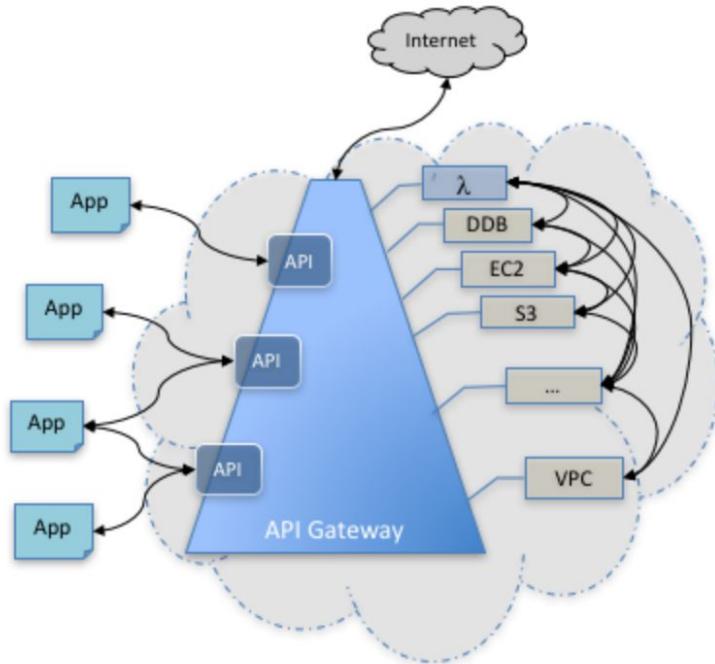
In general terms, it is a set of clearly defined methods of **communication** between various components.

# What is API Gateway

# What is API Gateway?



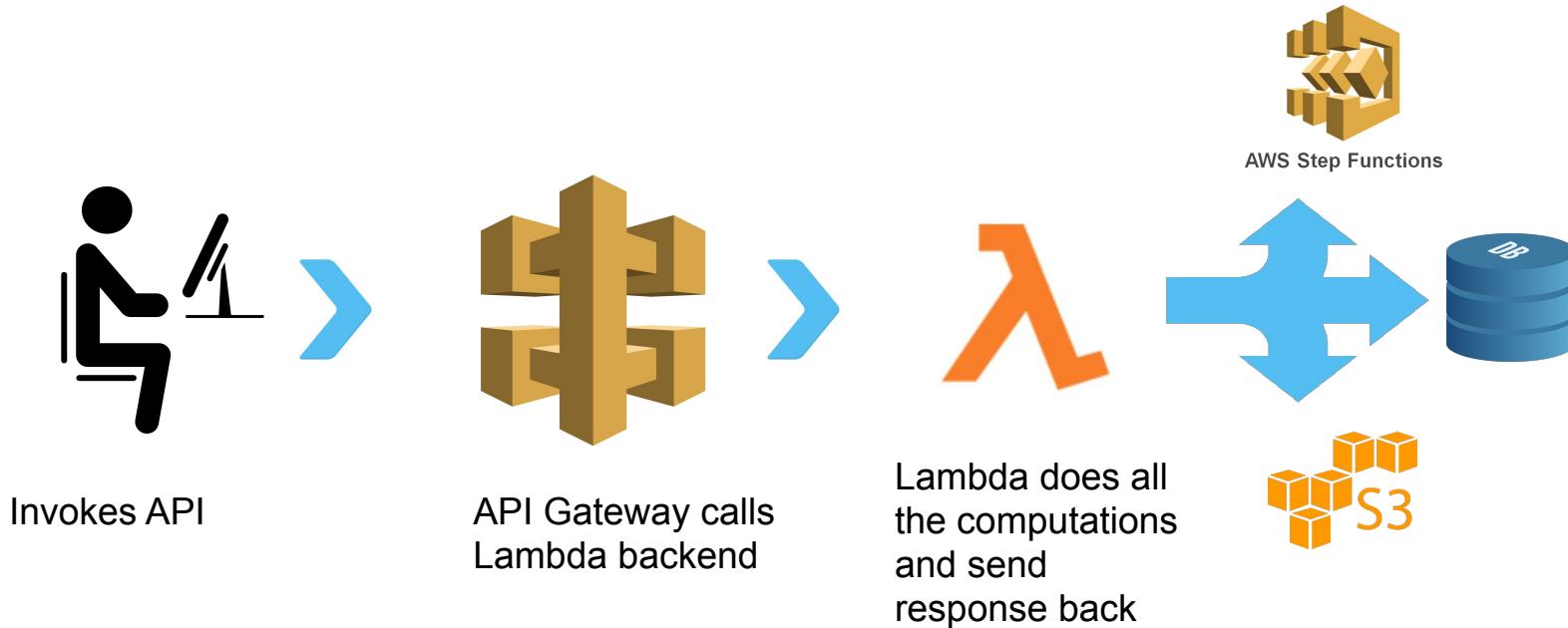
# API Gateway continued...



## Functions of API Gateway

- Lets you create, configure, and host a API
- Authentication and Authorization of your API
- Tracing, Caching and Throttling of API Requests
- Staged deployments, Canary release
- And much more..

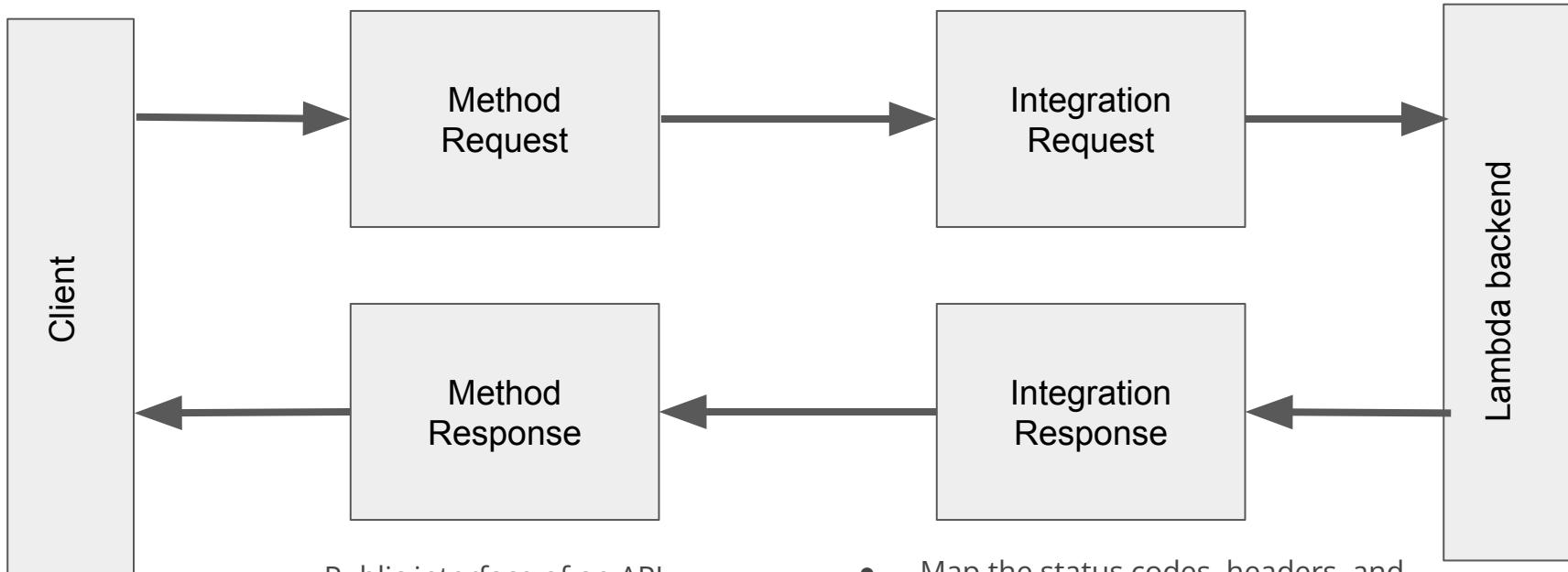
# How does Lambda fit into API Gateway?



# API Gateway Components

# API Gateway Components

- The public interface of an API method
  - Defines the parameters and body
- Interfaces with backend
  - Map the parameters and body of a method request to the formats required by the backend



- Public interface of an API
  - Defines the status codes, headers, and body models
- Map the status codes, headers, and payload that are received from the backend to the response format that is returned to a client app

# API Gateway Components - Continued

**Invoke URL** - URL to invoke the API

**Usage Plan** - A [usage plan](#) provides selected API clients with access to one or more deployed APIs. You can use a usage plan to configure throttling and quota limits, which are enforced on individual client API keys

**API Developer** - Your AWS account that owns an API Gateway deployment (for example, a service provider that also supports programmatic access.)

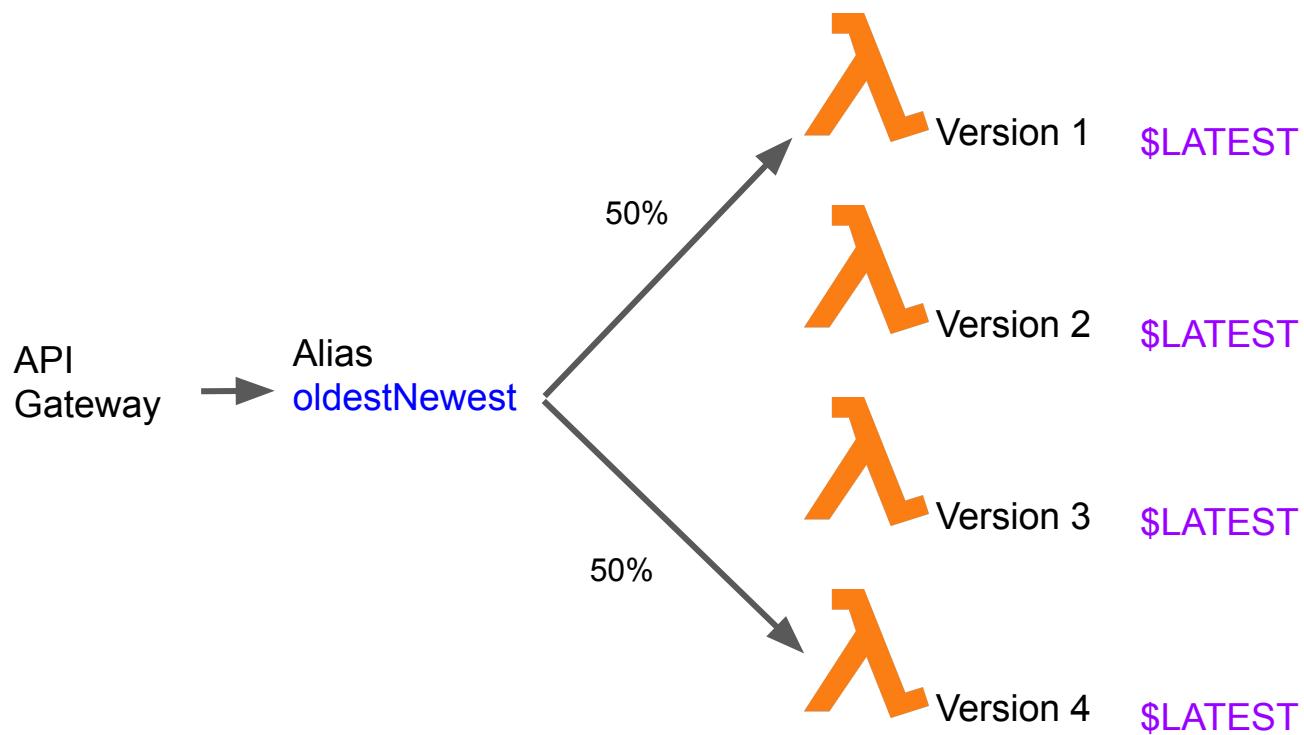
**App Developer** - An app creator who may or may not have an AWS account and interacts with the API that you, the API developer, have deployed. App developers are your customers. An app developer is typically identified by an [API key](#).

**Resources** (could be different projects/business areas)

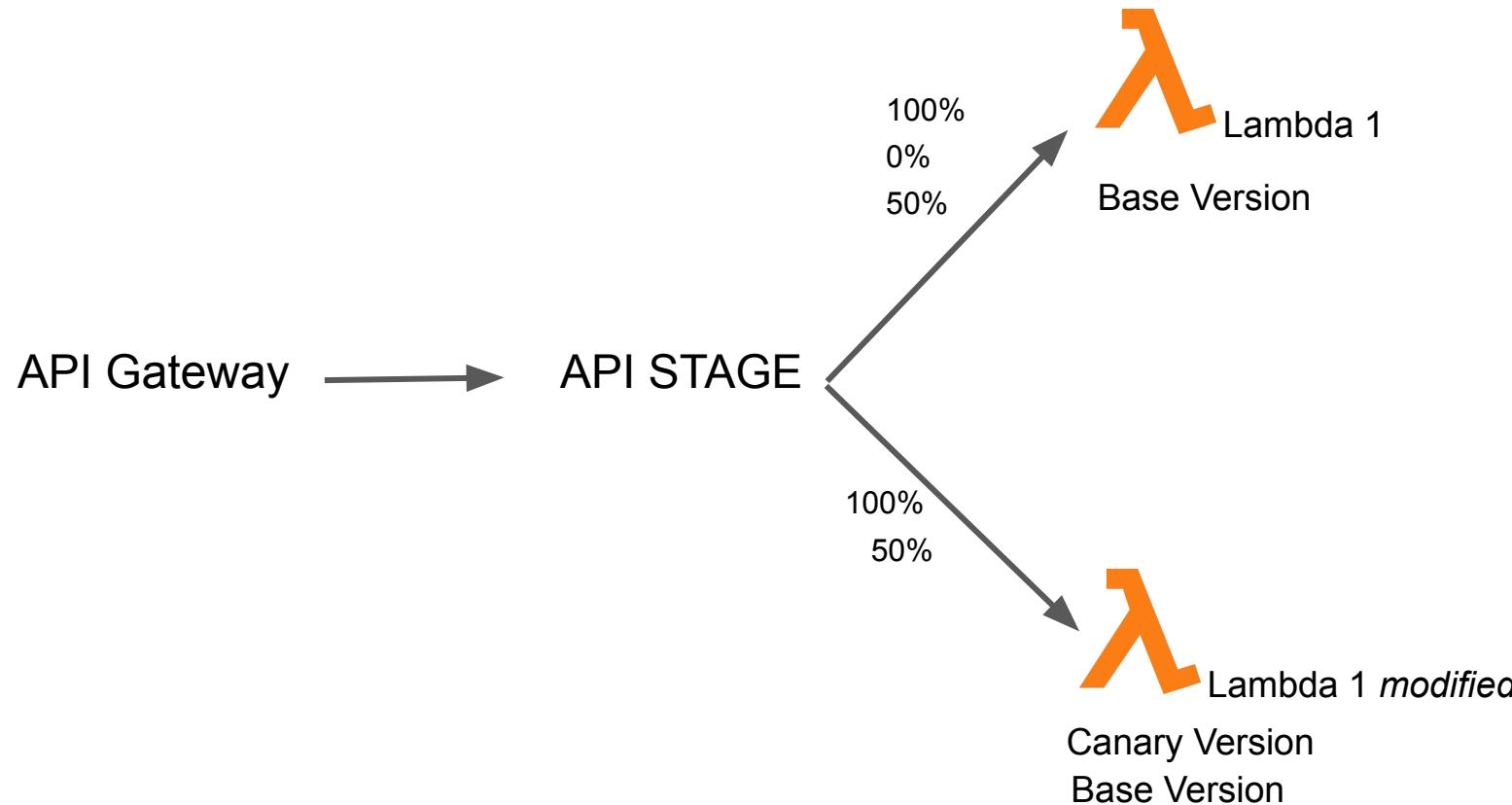
 **Methods** (GET, POST etc.) - Each method along with resources, are deployed to stages, with invoke url for each method under each resource in each stage

# Lambda Version and Alias

# Lambda Version and Alias



# API Gateway Canary Deployment



# API Gateway API Endpoint Types

# API Gateway API Endpoint Types

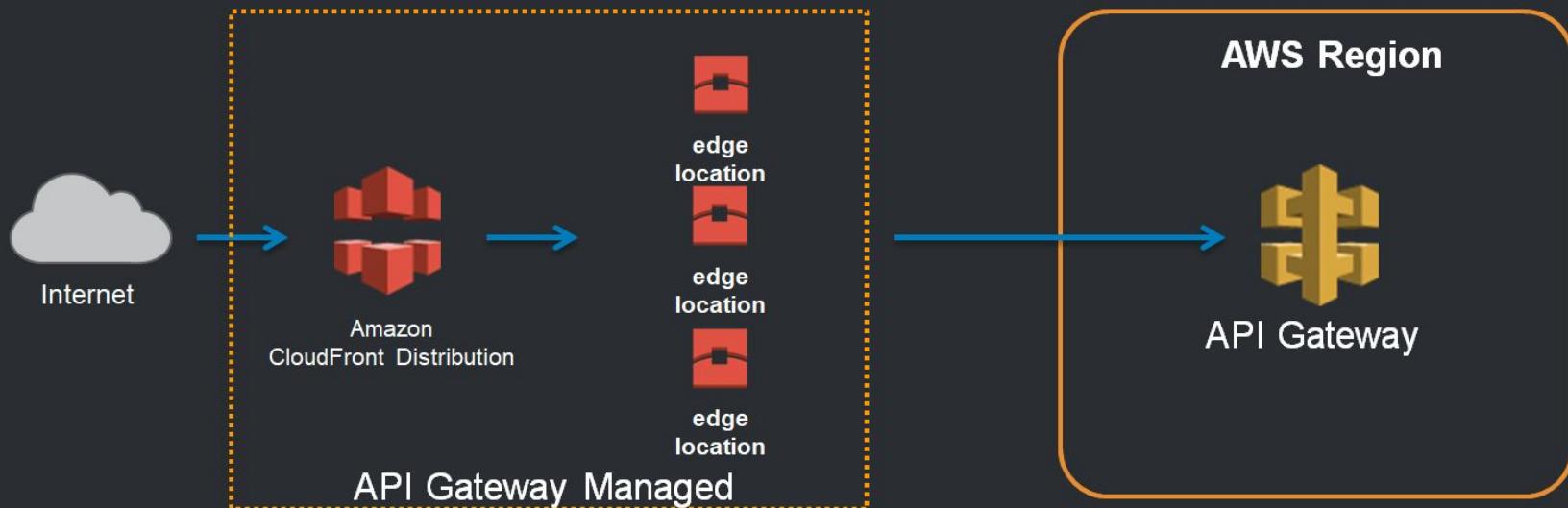
**Edge optimized:** Designed to help you reduce client latency from anywhere on the Internet

**Regional:** Designed to reduce latency when calls are made from the same region as the API

**Private:** Designed to expose APIs only inside your VPC

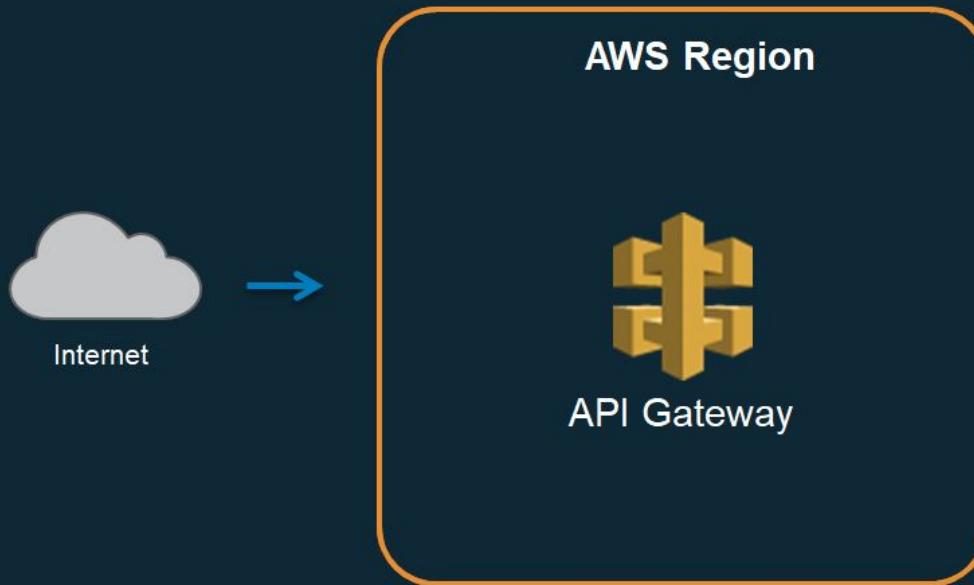
# Edge Optimized

**Edge optimized:** Designed to help you reduce client latency from anywhere on the Internet

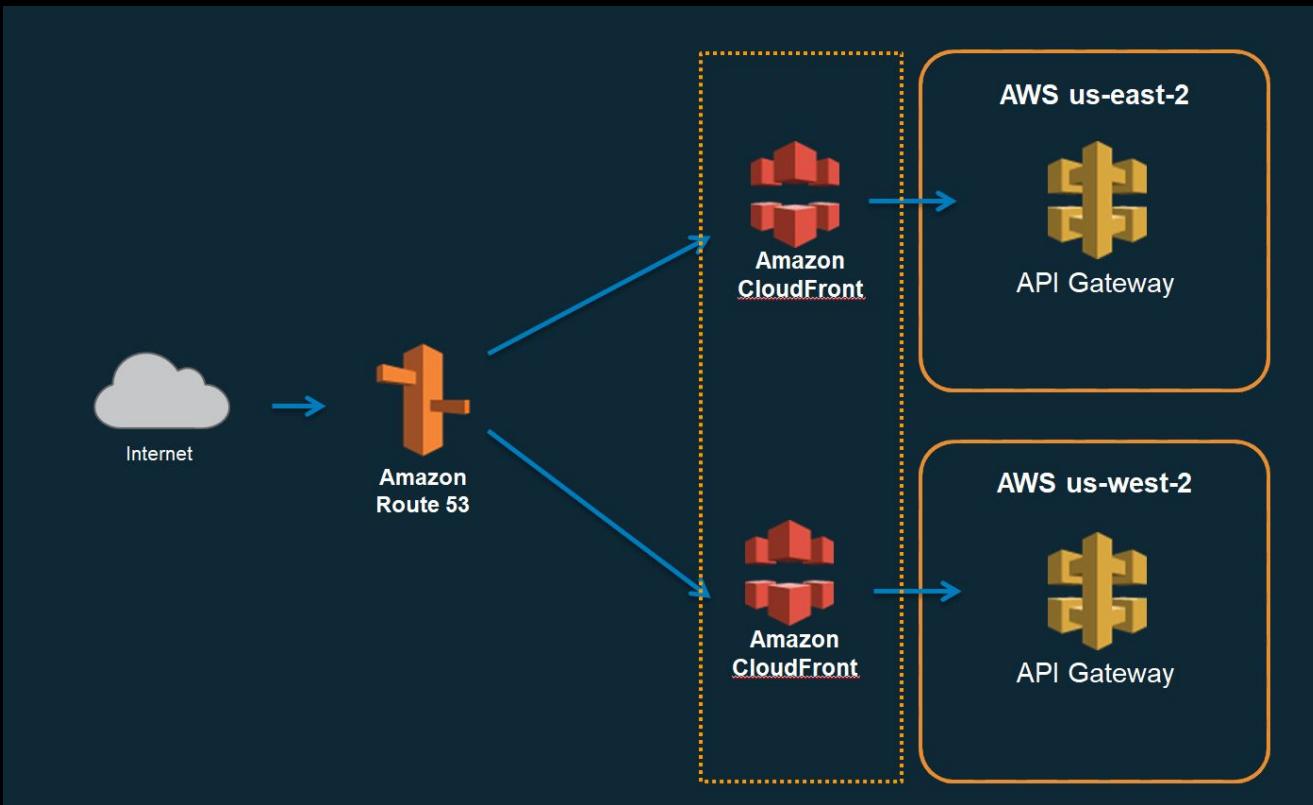


# Regional

**Regional:** Designed to reduce latency when calls are made from the same region as the API

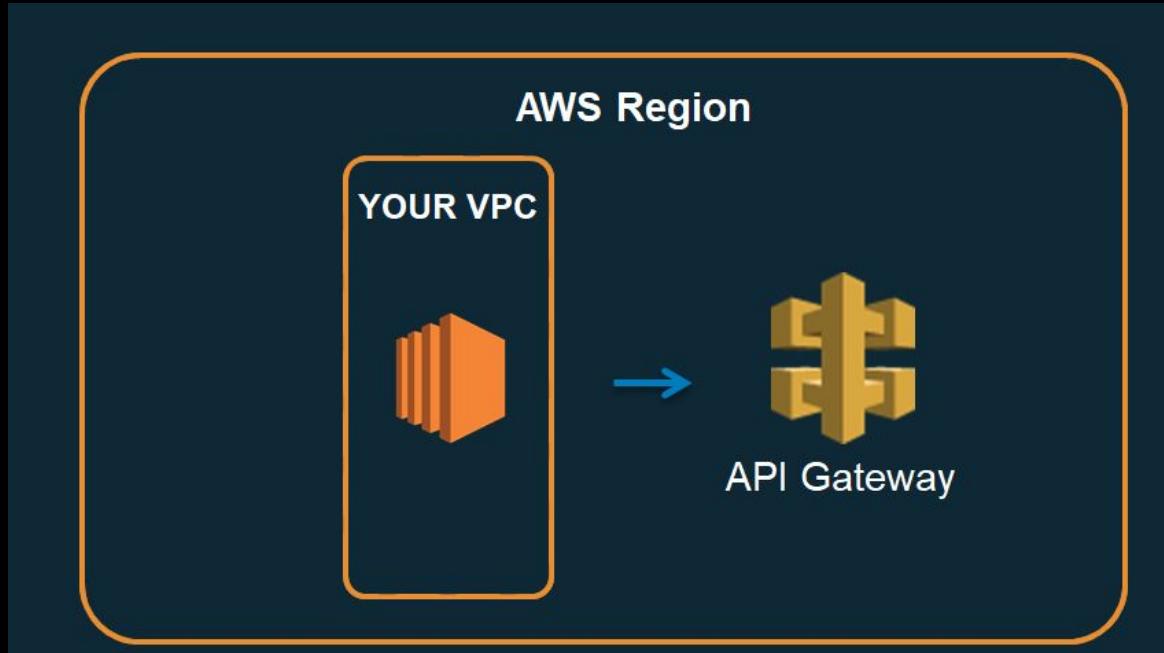


# Regional



# Private

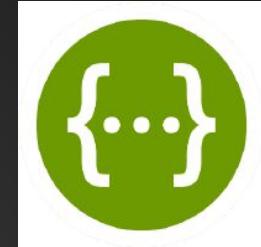
**Private:** Designed to expose APIs only inside your VPC



# DEMO

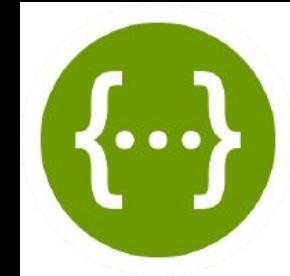
- Choosing Endpoint Type of your API
- Note - You can change your Endpoint Type!

# What is Swagger?



- API as Code
- Human and machine readable to create APIs
- Share APIs internally and externally
- Generates API documentation
- Supports YAML and JSON
- Now called “Open API Initiative”
- You can generate Swagger files for all your APIs in API Gateway, even if you created them in console!

# Why Swagger?



- Recreate your APIs
- Enables you to use any API Management tool (NOT Locked in)
- Create standards for the APIs in your enterprise
- Documentation helps future developers

# DEMO

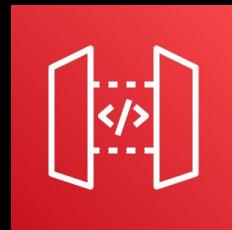
- Export API Gateway API as Swagger
- Create API with Swagger

# DEMO - *Under SAM*

- Defining API with SAM and Swagger

# CORS (Cross Origin Resource Sharing)

+



API Gateway

# What is CORS?

- Browser security feature that restricts cross-origin HTTP requests

## What Qualifies for Cross Origin HTTP Requests?

- A different domain (e.g. from cat.com to dog.com)
- A different subdomain (e.g. from cat.com to adopt.cat.com )
- A different port (e.g. from cat.com to cat.com:10700)
- A different protocol (e.g. from https://cat.com to http://cat.com)

# Cross-origin HTTP requests

## Simple (All below has to satisfy)

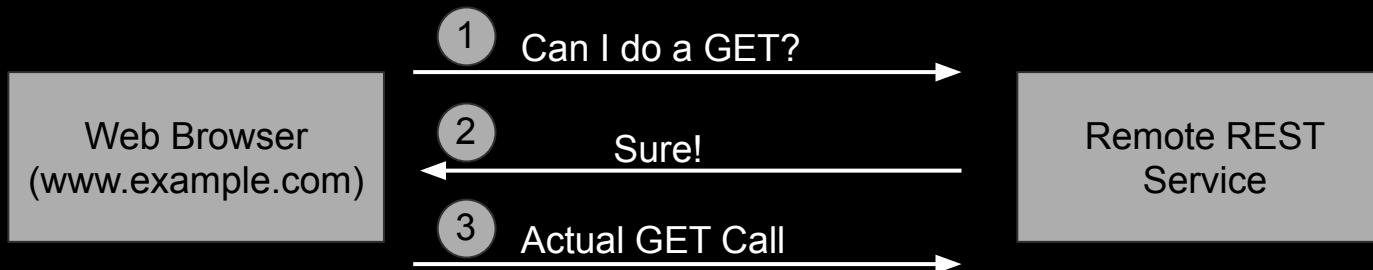
- Only for GET, HEAD, POST
- POST must include Origin header
- Request payload content type is text/plain, multipart/form-data, or application/x-www-form-urlencoded
- Request does not contain custom header
- Plus requirements in [https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Simple\\_requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Simple_requests)

## Non Simple (Almost All Real World APIs)

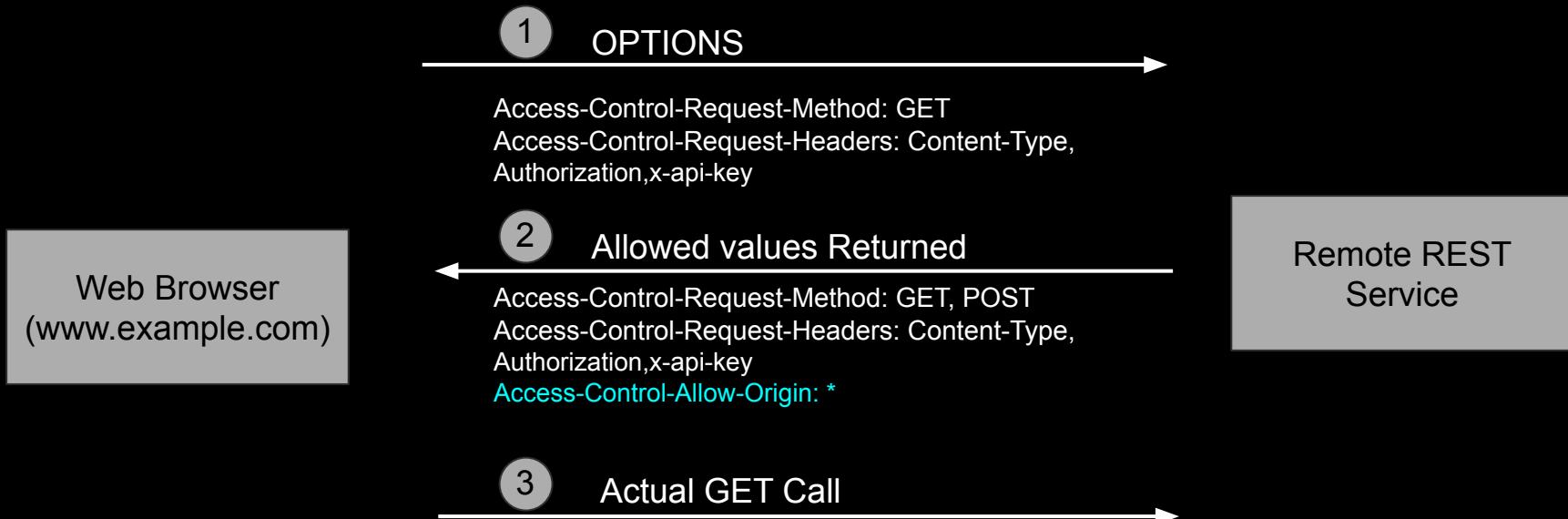
- All other cross origin HTTP Requests

# CORS Pre-flight

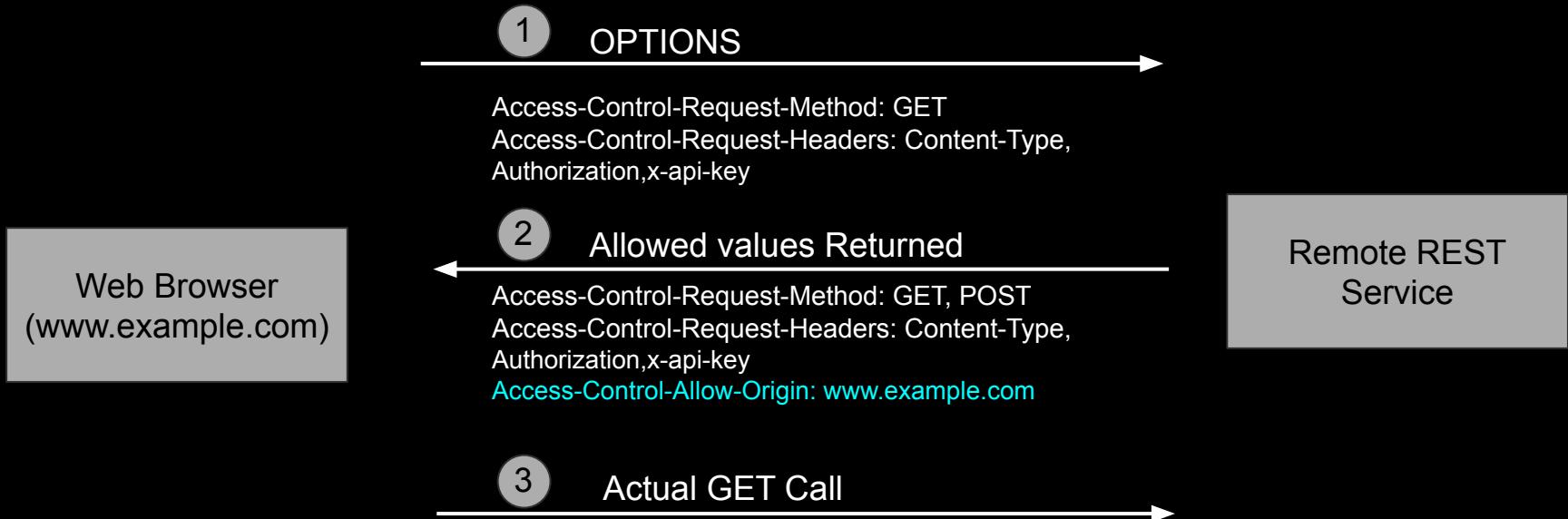
- First invoke OPTIONS Method to validate
- After getting valid response with allowed values, invoke actual Method (GET, POST etc.)



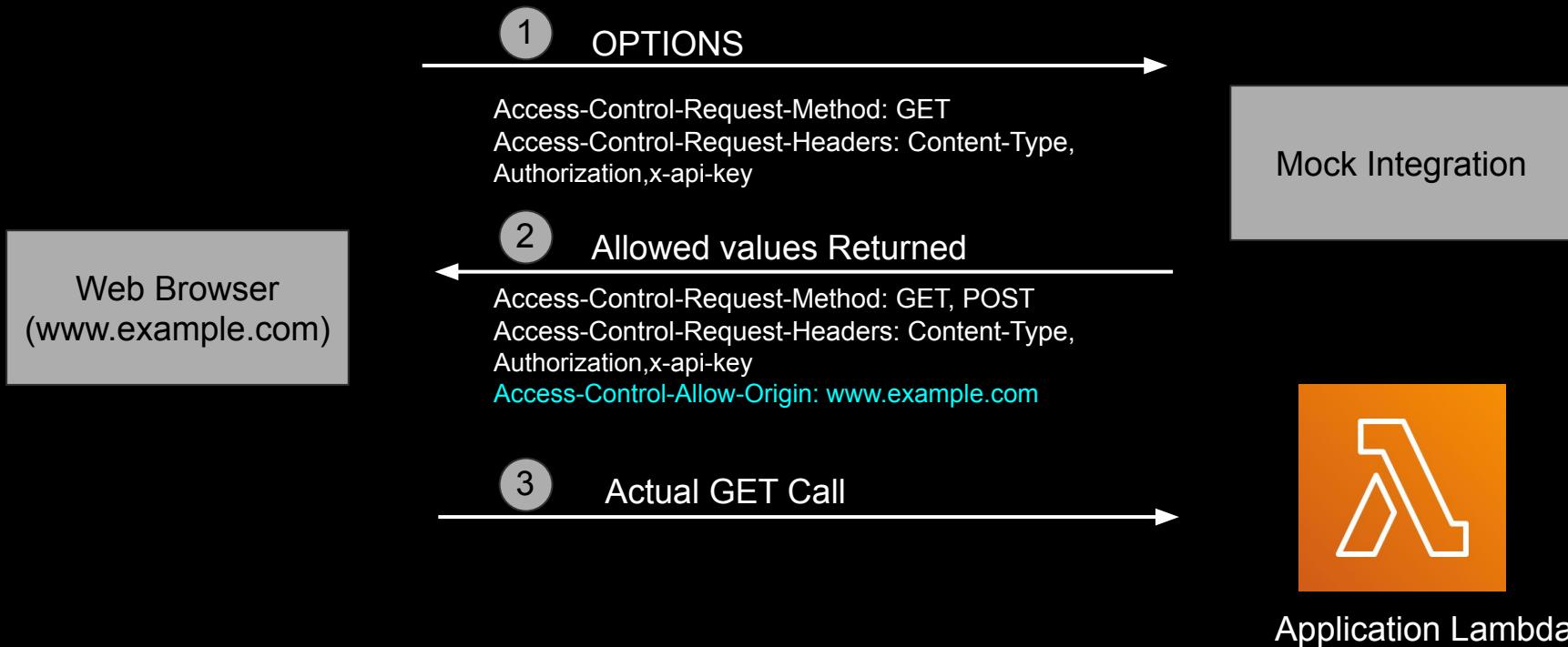
# CORS Flow



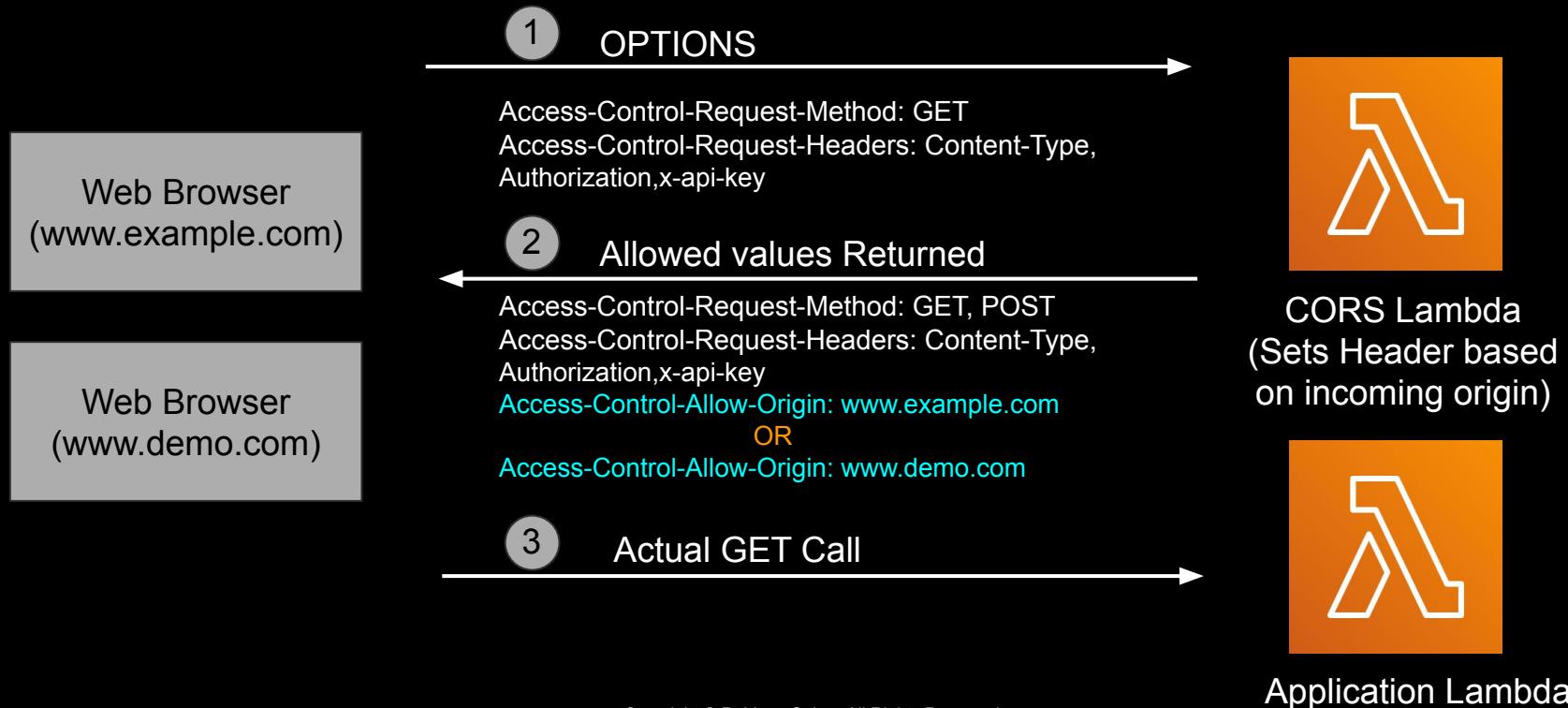
# CORS Flow



# CORS Flow - API Gateway



# Multi Origin CORS Flow



# REST API



# HTTP API

# HTTP APIs - What and Why?

- What if only API integration you need is Lambda and Url
- HTTP APIs announced Dec 2019
- Low-latency, cost effective integration with Lambda and HTTP URLs
- That's not the only difference!

# HTTP API Vs Rest API - API Types

API type	HTTP API	REST API
Regional	✓	✓
Edge-optimized		✓
Private		✓

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# HTTP API Vs Rest API - Integration

Integration	HTTP API	REST API
HTTP proxy	✓	✓
Lambda proxy	✓	✓
HTTP		✓
AWS services		✓
Private integration	✓	✓
Mock		✓

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# HTTP API Vs Rest API - Security

Security	HTTP API	REST API
Client certificates		✓
AWS WAF		✓
Resource policies		✓

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# HTTP API Vs Rest API - Authorizers

Authorizers	HTTP API	REST API
AWS Lambda		✓
IAM		✓
Amazon Cognito	✓ *	✓
Native OpenID Connect / OAuth 2.0	✓	

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# HTTP API Vs Rest API - API Management

API management	HTTP API	REST API
Usage plans		✓
API keys		✓
Custom domain names	✓ *	✓

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# HTTP API Vs Rest API - Monitoring

Monitoring	HTTP API	REST API
Access logs to Amazon CloudWatch Logs	✓	✓
Access logs to Amazon Kinesis Data Firehose		✓
Execution logs		✓
Amazon CloudWatch metrics	✓	✓
AWS X-Ray		✓

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

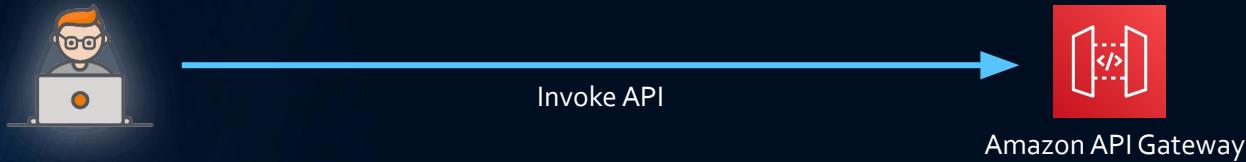
# HTTP API Vs Rest API - Development

Development	HTTP API	REST API
Cache		✓
Request transformation		✓
Request / response validation		✓
Test invocation		✓
CORS configuration	✓	✓
Automatic deployments	✓	
Default stage	✓	
Default route	✓	

Source: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>

# Custom Domain API Gateway

# Using API Gateway Url



**Invoke**  
**URL:** <https://p6xpf5z5m.execute-api.us-west-2.amazonaws.com/default/myfirstlambda>

# Using API Gateway Url



# Using API Gateway Url

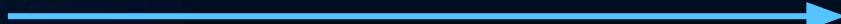


# Using Custom Domain



Amazon EC2

Application unchanged  
Server authentication using  
certificates



Invoke API via domain



Amazon API Gateway

**Invoke**  
**URL:** <https://lambda-api.com>

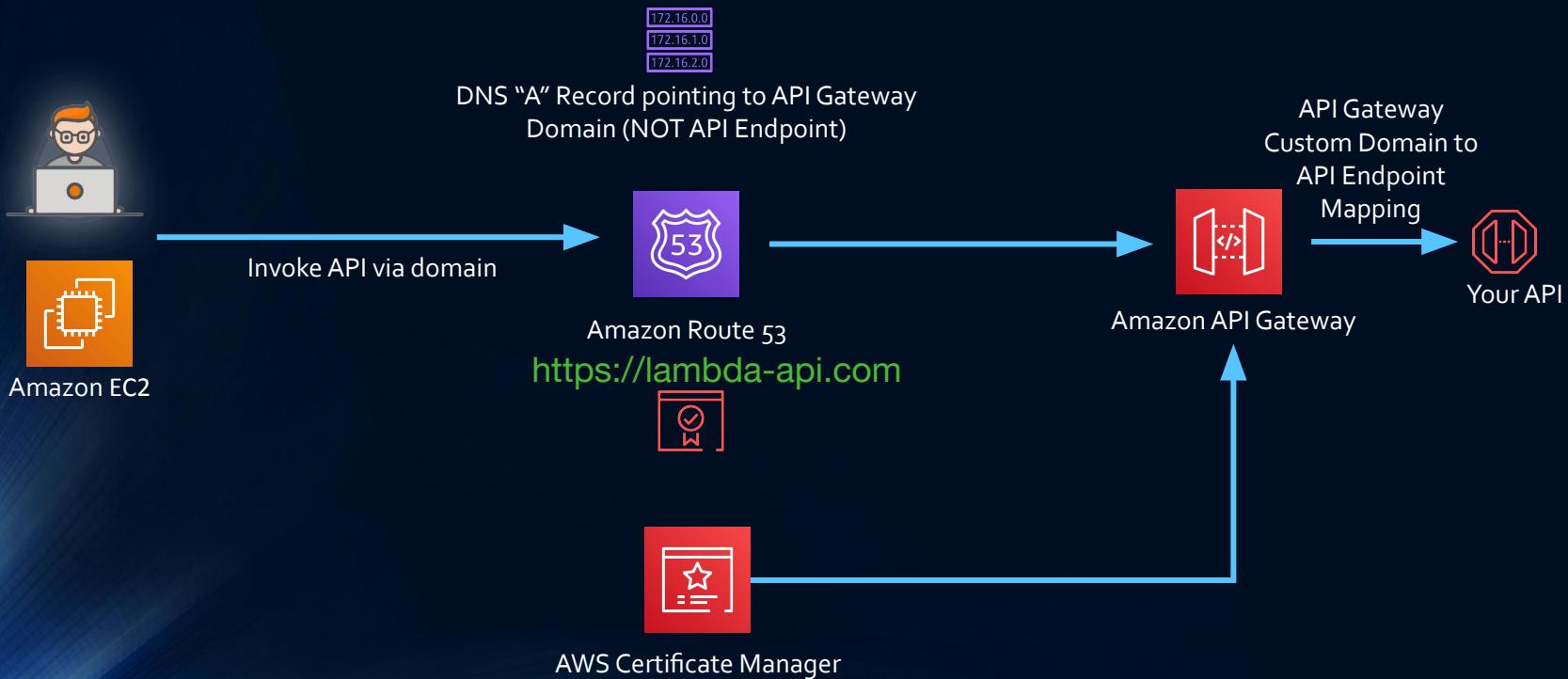
**Domain pointing to:** <https://p6xpfz5m.execute-api.us-west-2.amazonaws.com/default/myfirstlambda>

**Domain pointing to:** <https://p778bcx6.execute-api.us-west-2.amazonaws.com/default/mylatestlambda>

# Using API Gateway Custom Domain

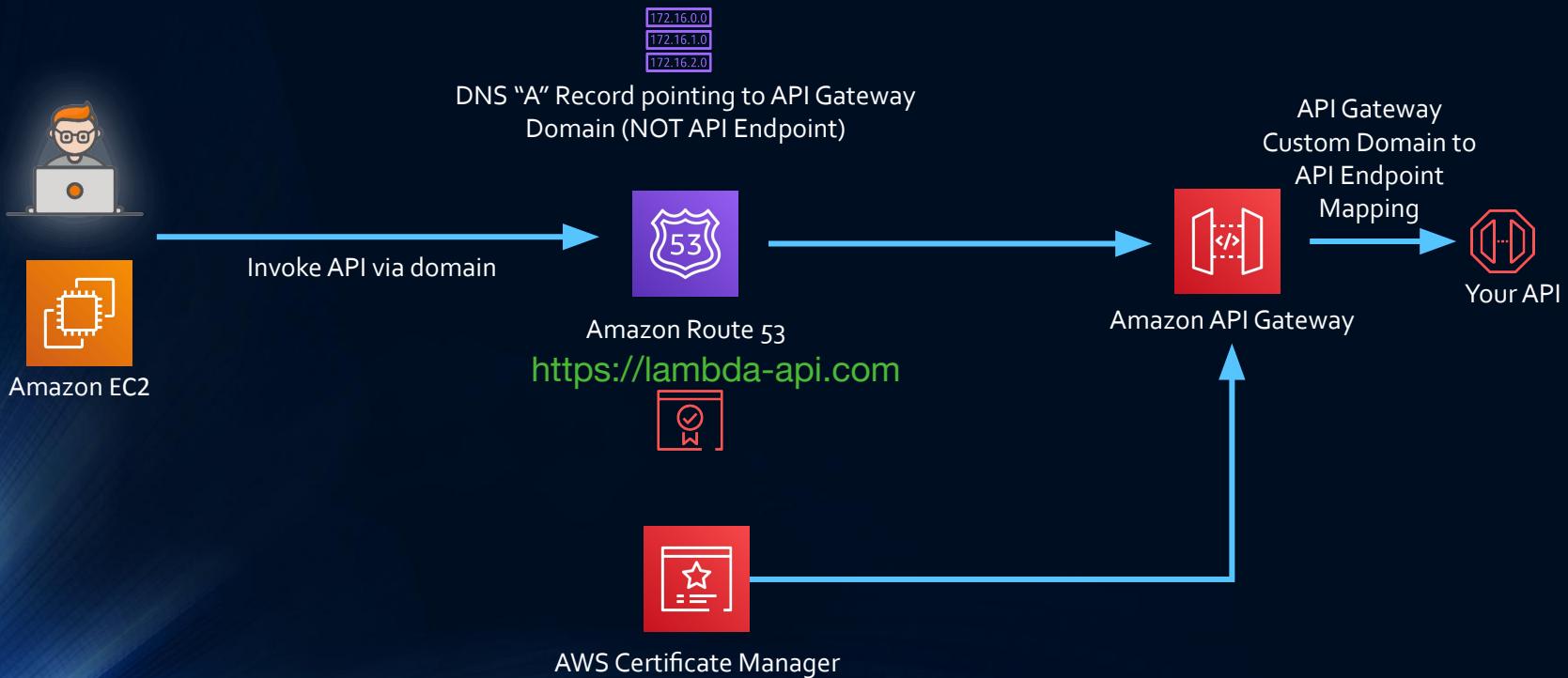
- ACM can be expensive (\$400 per certificate after free-tier expires)
- Lot of steps involved

# API Gateway Custom Domain Flow



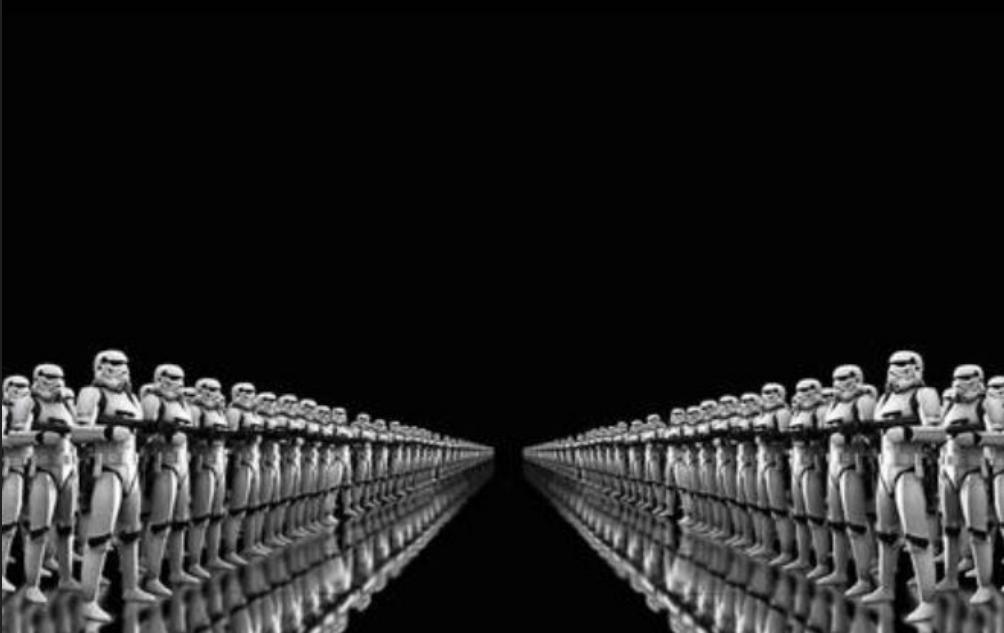
# Demo

# What about Client Authentication?



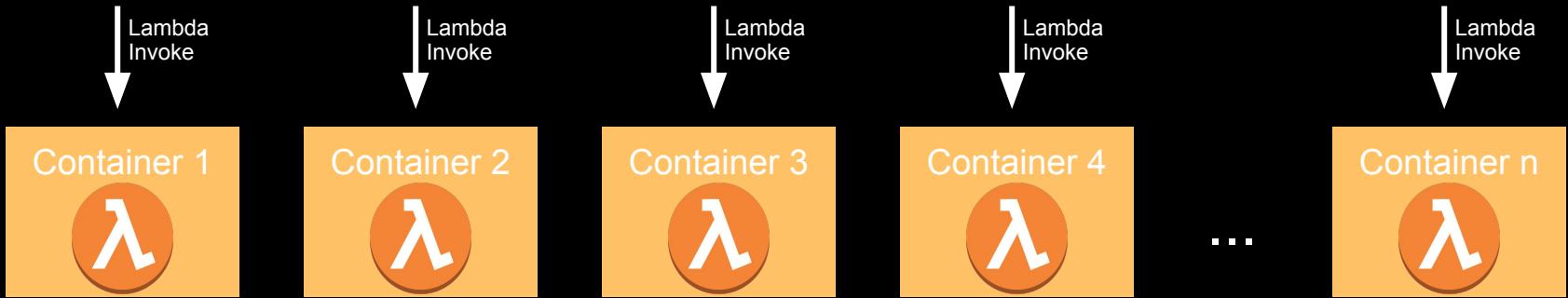
Lambda  
Scaling &  
Concurrency

# How Does Lambda Scale?



# Lambda Code Execution Under The Hood

## Concurrent Invocation

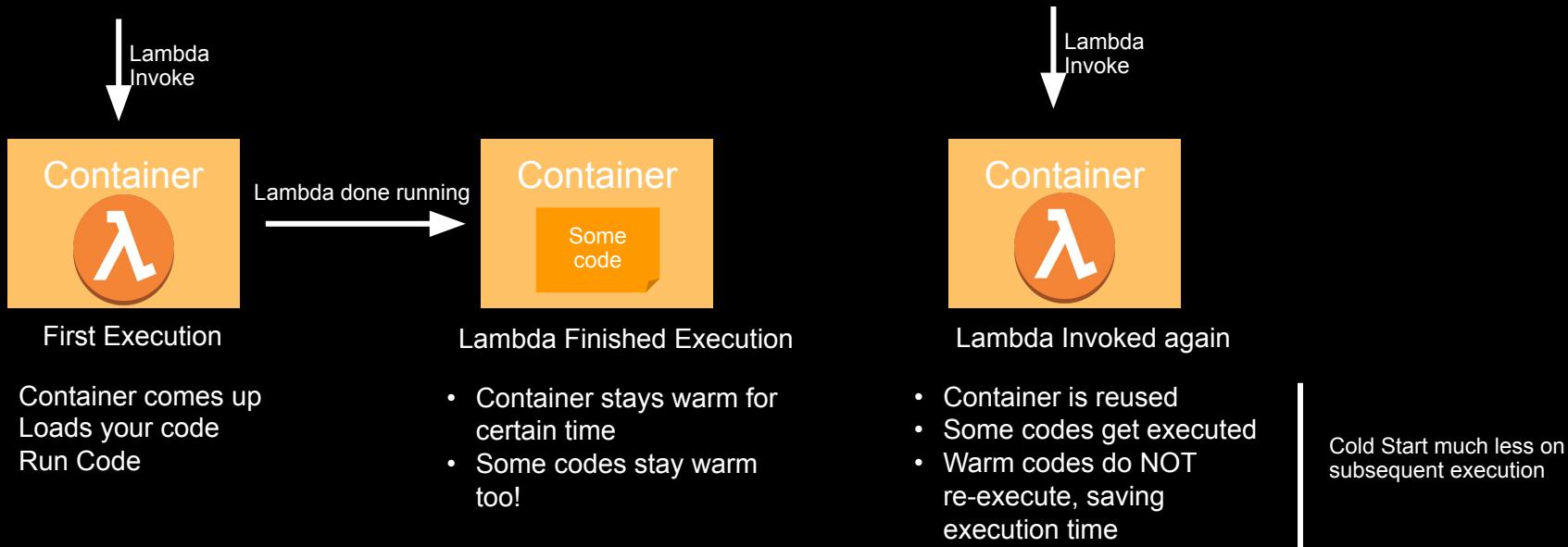


Cold Start

- Container comes up
- Loads your code
- Run Code

- Limit of Scaling
- Rate of Scaling

# Lambda Code Execution Under The Hood



What Codes Stay Warm?

## Concurrency

Unreserved account concurrency **1000**

- Use unreserved account concurrency
- Reserve concurrency

New! (re:Invent 2019)

## Provisioned concurrency

To enable your function to scale without fluctuations in latency, use provisioned concurrency. Provisioned concurrency runs continually and is billed in addition to standard invocation costs. [Learn more](#)

### Provisioned concurrency configurations (0)

 Edit Remove Add

 Find configuration

Qualifier	Type	Provisioned concurrency	Status	Details
-----------	------	-------------------------	--------	---------

No configurations

Add configuration

# Account Concurrency

## Concurrency

Unreserved account concurrency **1000**

- Use unreserved account concurrency
- Reserve concurrency

# Account Concurrency - Scaling Limit

## Concurrency

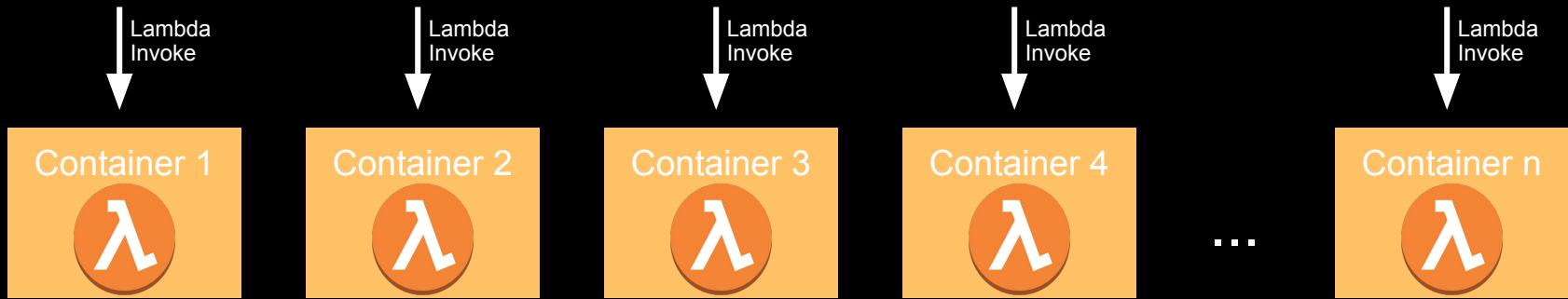
Unreserved account concurrency **1000**

- Use unreserved account concurrency
- Reserve concurrency

How about Rate of Scaling?

# What if Lambda Needs to Scale Super Fast!

Concurrent Invocation - 0 to 1000 in 1 second!



Rate of Scaling Can't Keep Up - Throttling

# Provisioned Concurrency to the Rescue

## Concurrency

Unreserved account concurrency **1000**

- Use unreserved account concurrency
- Reserve concurrency

New! (re:Invent 2019)

## Provisioned concurrency

To enable your function to scale without fluctuations in latency, use provisioned concurrency. Provisioned concurrency runs continually and is billed in addition to standard invocation costs. [Learn more](#)

### Provisioned concurrency configurations (0)

 Edit Remove Add

 Find configuration

Qualifier



Type



Provisioned concurrency



Status



Details

No configurations

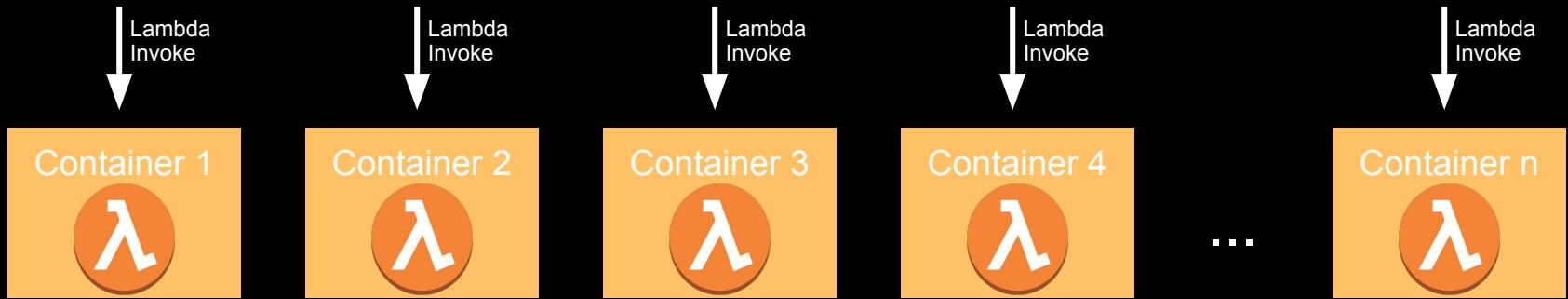
Add configuration

# Provisioned Concurrency

- Pre-initialized Execution Environments

# What if Lambda Needs to Scale Super Fast!

n Containers up with Lambda Code Loaded



# Provisioned Concurrency

- Pre-initialized Execution Environments
- No Cold Start or Throttling due to Super Fast Traffic Increase
- AWS will keep assigned capacity “Warm”

# Lambda Environment Variables

- Key-value pairs that you can dynamically pass to your function without making code changes
- Available via standard environment variable APIs
- Can be encrypted via AWS Key Management Service (AWS KMS)
- Useful for creating environments per stage (i.e., dev, testing, production)

# Lambda Environment Variables

```
import boto3
import os
import datetime

textract = boto3.client('textract')
sns = boto3.client('sns')
s3client = boto3.client('s3')

badimagesns=os.environ['bad_image_sns']
circuitname=os.environ['circuit_name']
s3resultbucket=os.environ['s3_result_bucket']

def lambda_handler(event, context):
    print(event)
    for item in event['Records']:
```

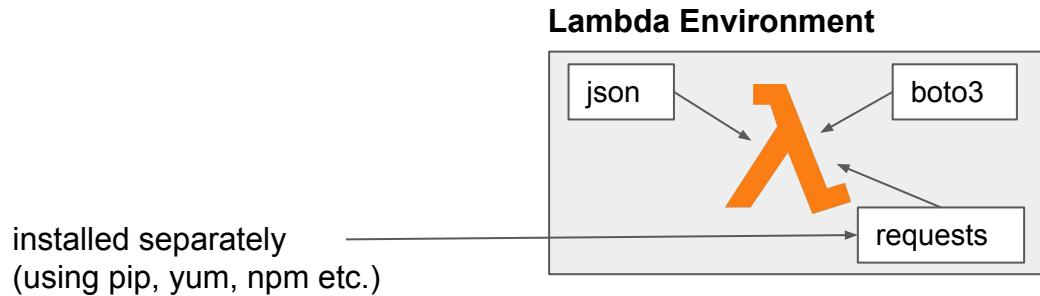
## Environment variables

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](#)

bad_image_sns	arn:aws:sns:us-east-1:453986252919:drcft1252-Ba	<button>Remove</button>
circuit_name	Kumo Torakku	<button>Remove</button>
s3_result_bucket	drcft1252-s3bucketforresult-bc9x5qr6cx9y	<button>Remove</button>
Key	Value	<button>Remove</button>

▶ [Encryption configuration](#)

# Lambda and External Dependencies



## Lambda Code

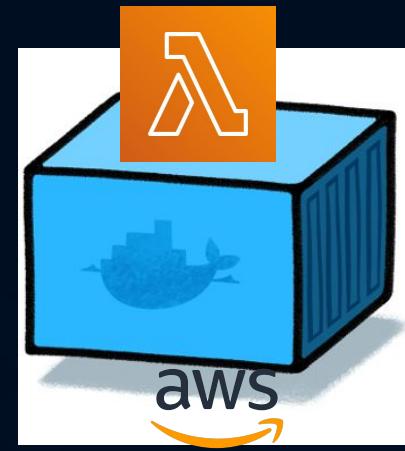
```
import json  ✓  
import boto3  ✓  
import requests ✓✗  
  
def lambda_handler(event, context):  
    .....  
    .....
```

# Lambda Container Image

# Agenda:

- Lambda container image – what and why
- Demo
- Best practices







- AWS provides underlying container, your code/dependencies gets loaded
- Package and deploy Lambda function as container images
- This does NOT run Lambda code on EKS/ECS (NOT Knative/CloudRun equivalent)
- Container image can be up to 10 GB in size  
(Compared to 50 MB for Zip deployment )



# Lambda Container Image Support





- AWS provides base images
- Runtime interface client manage interaction between Lambda service and your function code
- You can create custom image
  - Require to have runtime interface clients
  - Supports linux based image currently
  - Supports specific container image settings

# Advantages



- Utilize existing container tooling
- Create image with what you need – faster start up time
- Perform local testing with runtime interface emulator
- 10 GB package size compare to 50 MB Zip deployment

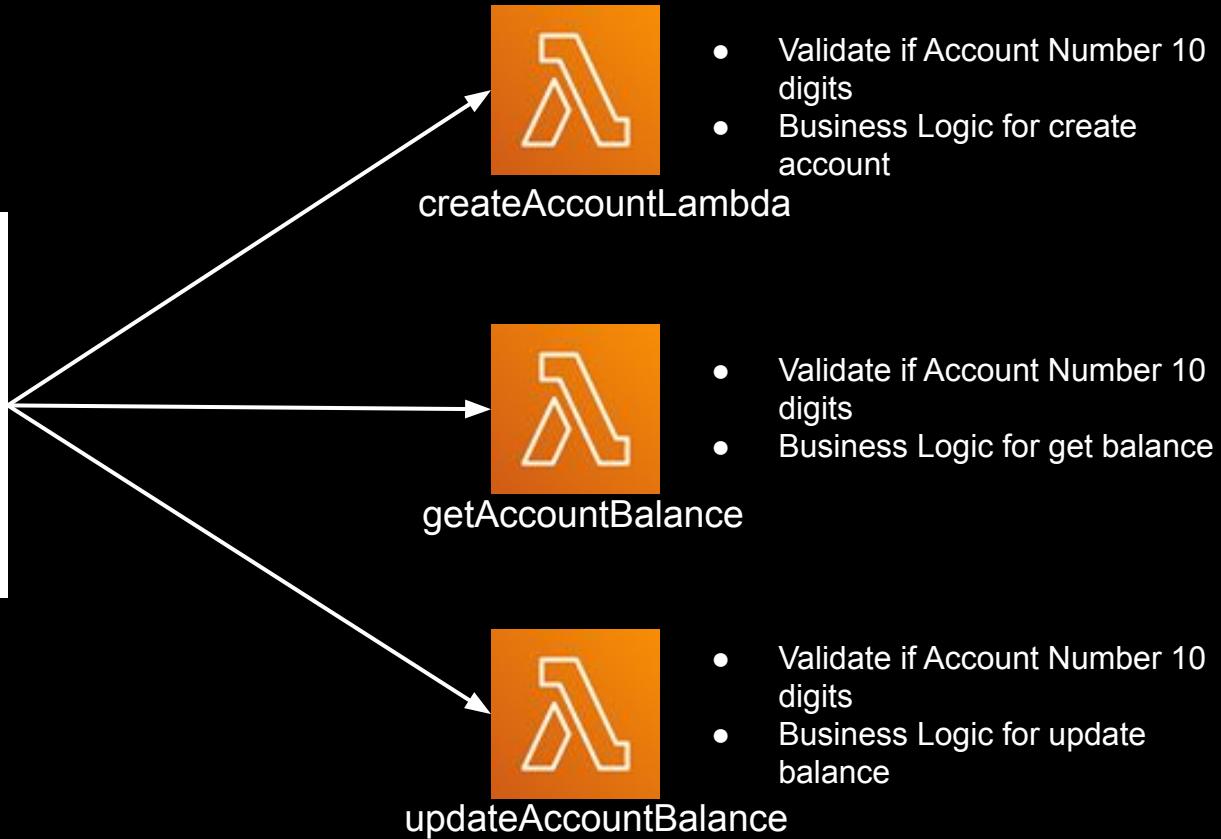
# Demo

- Demo of the Lambda container image
- Pay for ECR and Lambda
- Supported in AWS CLI, CloudFormation, SAM

# Lambda Layer

# Lambda Layer CloudWatch Insights

# Enterprise Scenario



# Enterprise Scenario



- Validate if Account Number 10 digits
- Business Logic for create account

createAccountLambda



- Validate if Account Number 10 digits
- Business Logic for get balance

getAccountBalance



- Validate if Account Number 10 digits
- Business Logic for update balance

updateAccountBalance

- Duplication of code over time
- If logic need to be changed, it need to be changed in many programs
- Testing and maintenance overhead

# Enterprise Scenario

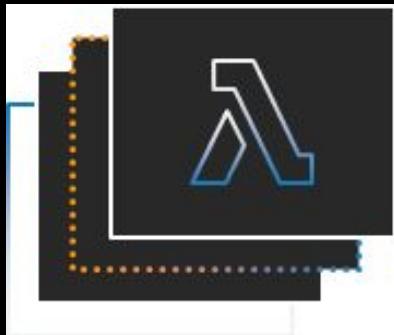
```
Function createAcchandler(event, context){  
    if validateAcct(event['AcctNo']) == 'PASS':  
        createAccount(event)  
    else:  
        errmsg="Invalid Account Number Format"  
..... <more logic>  
    return result;  
}
```

```
validateAcct(AcctNo){  
    ## logic here  
}
```

Duplicated logic in every Lambda

```
createAccount(inputevent){  
    ## logic here  
}
```

# Lambda Layers



Lets functions easily share code: Upload layer once, reference within any function

Layer can be anything: dependencies, training data, configuration files, etc.

Promote separation of responsibilities, lets developers iterate faster on writing business logic

Built in support for secure sharing by ecosystem

# Enterprise Scenario w/ layer



- Business Logic for create account

createAccountLambda



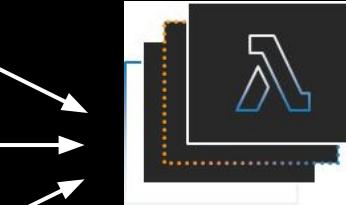
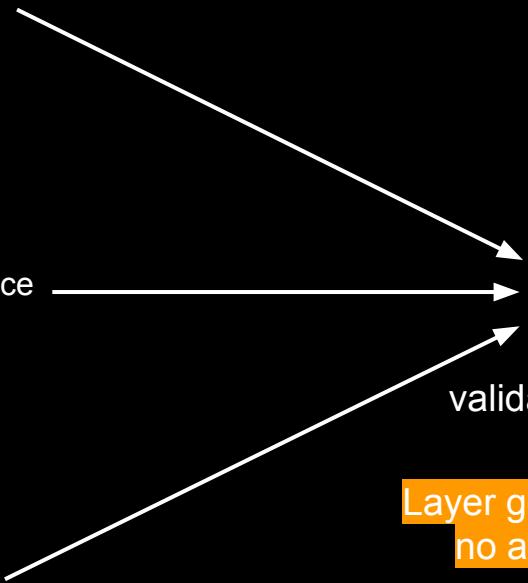
- Business Logic for get balance

getAccountBalance



- Business Logic for update balance

updateAccountBalance



validateAcct() in Lambda Layer

Layer gets loaded with function code,  
no additional execution latency

# Lambda Layers re:Invent 2018 Update

- Layers simplify sharing, versioning, and deploying common code.
- 250 MB total size limit unchanged (total layers, unzipped).
- Up to 5 layers per function.

## Layers DOs

Put shared code into discrete layers

Version layers and use to deploy across accounts

Simplify your deployment management

## Layers DON'Ts

Don't put unnecessary stuff into the layer, don't treat it as dumpstar! It will increase code loading time.

# Lambda EFS Integration



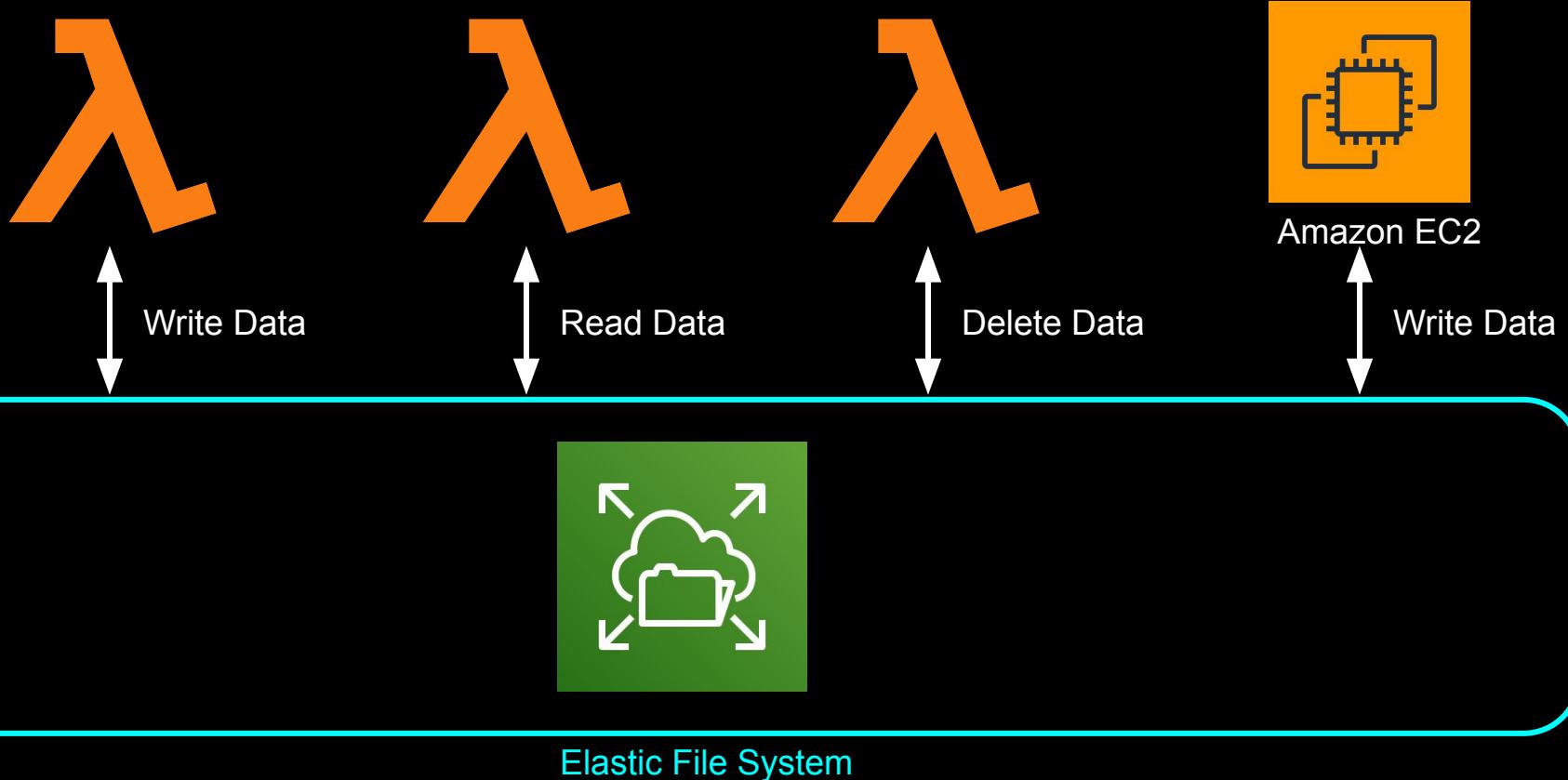
+



=



# Lambda EFS Integration



# Lambda EFS Integration

- Pay for what you use (Unlike EBS and RDS)
- EFS is shared across concurrent executions of a Lambda function
- EFS can be used with Provisioned Concurrency
- Some use cases:
  - Process large files across multiple functions
  - Import large ML Models, Code Libraries
  - Use other services (e.g. EC2, EKS etc.) with Lambda
  - And more... (Lambda is NOT STATELESS Anymore!)

# Lambda EFS Integration Demo



+

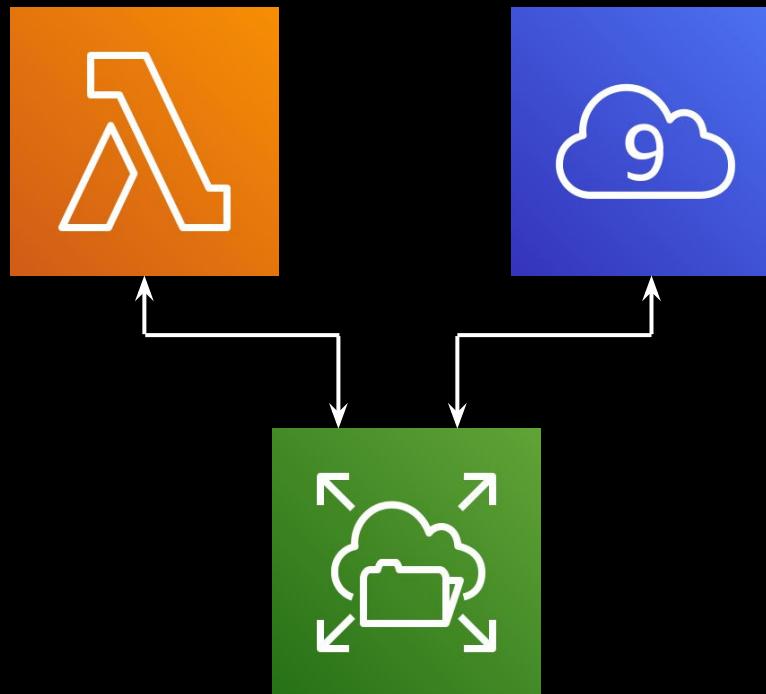


=



<https://aws.amazon.com/blogs/aws/new-a-shared-file-system-for-your-lambda-functions/>

# Lambda EFS



# AWS Serverless Application Repository

## alexa-skills-kit-nodejs-factskill

This Alexa sample skill is a template for a basic fact skill. Provided a list of interesting facts about a topic, Alexa will select a fact at random and tell it to the user when the skill is invoked.

skills   fact   alexa

Alexa Skills Kit

44.6K deployments

## hello-world

A starter hello-world serverless application

AWS

3.6K deployments

## alexa-skills-kit-nodejs-triviaskill

This Alexa sample skill is a template for a trivia style game with score keeping. Alexa will ask you a multiple choice questions and seek your response. Correct and Incorrect answers to questions are recorded.

skill   trivia   alexa

Alexa Skills Kit

3.6K deployments

## SecretsManagerRDSSingleMySQLRotationUser

Conducts an AWS SecretsManager secret rotation for RDS MySQL using the single user rotation scheme

AWS Secrets Manager

2.6K deployments

## alexa-skills-kit-nodejs-howtoskill

Create a parameter-based skill using a template called 'Minecraft Helper'. Ask how to craft an item in the game Minecraft, and this skill will give you instructions.

howto   skill   alexa

Alexa Skills Kit

2.2K deployments

## SecretsManagerRDSPostgreSQLRotationSingleUser

Conducts an AWS SecretsManager secret rotation for RDS PostgreSQL using single user rotation scheme

AWS Secrets Manager

1.6K deployments

## Datadog-Log-Forwarder

## microservice-http-endpoint

## alexa-skills-kit-python36-factskill

# AWS Serverless Application Repository

Home > Applications > Application details

## alex-skills-kit-nodejs-factskill

Alexa Skills Kit

 <https://github.com/alexa/skill-sample-nodejs-fact>

 965 stars

 Deploy

44.6K deployments

This Alexa sample skill is a template for a basic fact skill. Provided a list of interesting facts about a topic, Alexa will select a fact at random and tell it to the user when the skill is invoked.

[Readme](#)

[License](#)

[Permissions](#)

## alex-skills-kit-nodejs-factskill

This Alexa sample skill is a template for a basic fact skill. Provided a list of interesting facts about a topic, Alexa will select a fact at random and tell it to the user when the skill is invoked.

Made with ❤ by Alexa Skills Kit. Available on the [AWS Serverless Application Repository](#)

# LAMBDA DESTINATIONS

## CONSOLE CHANGES

The screenshot shows the AWS Lambda Destinations console. At the top left, there's a card for the function "iotbackend" with an orange icon, showing "Unsaved changes". Below it, a "Layers" section shows "Layers (0)". To the right, a red "NEW!" annotation points to a button with a plus sign and the text "Add destination", which is highlighted with a thick magenta border.

NEW!

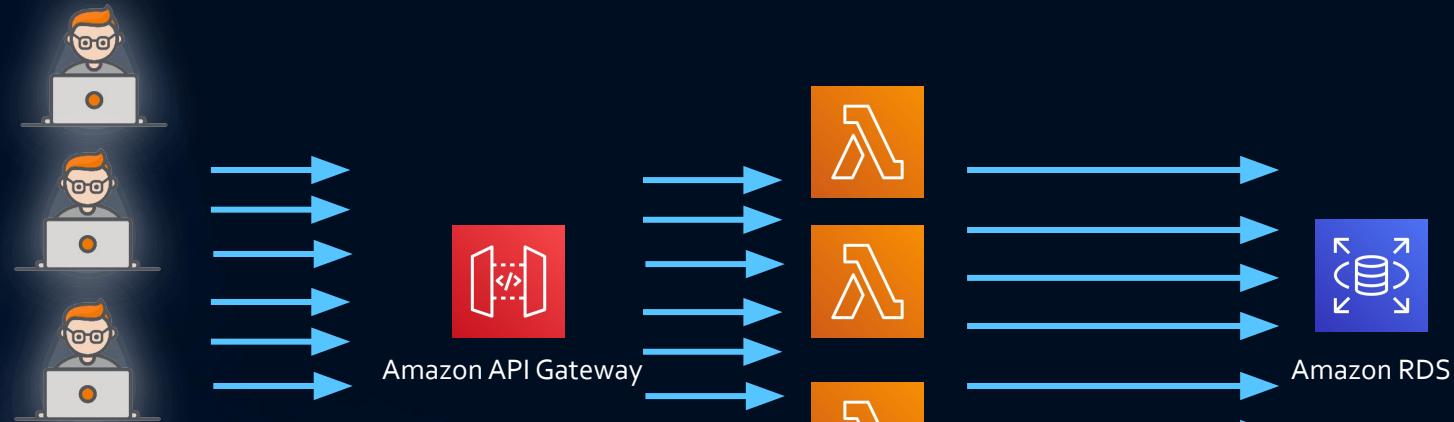
+ Add destination

# RDS Proxy

# Regular Lambda Database Design



# Increased Traffic



Lambda Code to handle connection!

- Can handle limited number of connections
- Orphan connections linger
- Database spends CPU/Memory for connection management
- Lambda can exhaust connection limit – throttling, error

# RDS Proxy



- Fully managed, highly available database proxy
- Allows applications to share pool of database connections
- Use Secrets Manager for database credential
- Failover without DNS change, 66% reduced failover time for Aurora, RDS
- Allocate how many connections Lambdas allow to use

# RDS Proxy Demo



# RDS Proxy – Couple More Points

- Currently works with
  - Amazon Aurora
  - RDS MySQL
  - RDS PostgreSQL
  - RDS MariaDB
  - RDS Oracle
  - RDS SQL Server
- Priced per vCPU per hour for each enabled database instance

# RDS Proxy – Demo

- **NOT FREE** - Priced per vCPU per hour for each enabled database instance



# RDS Proxy – Demo

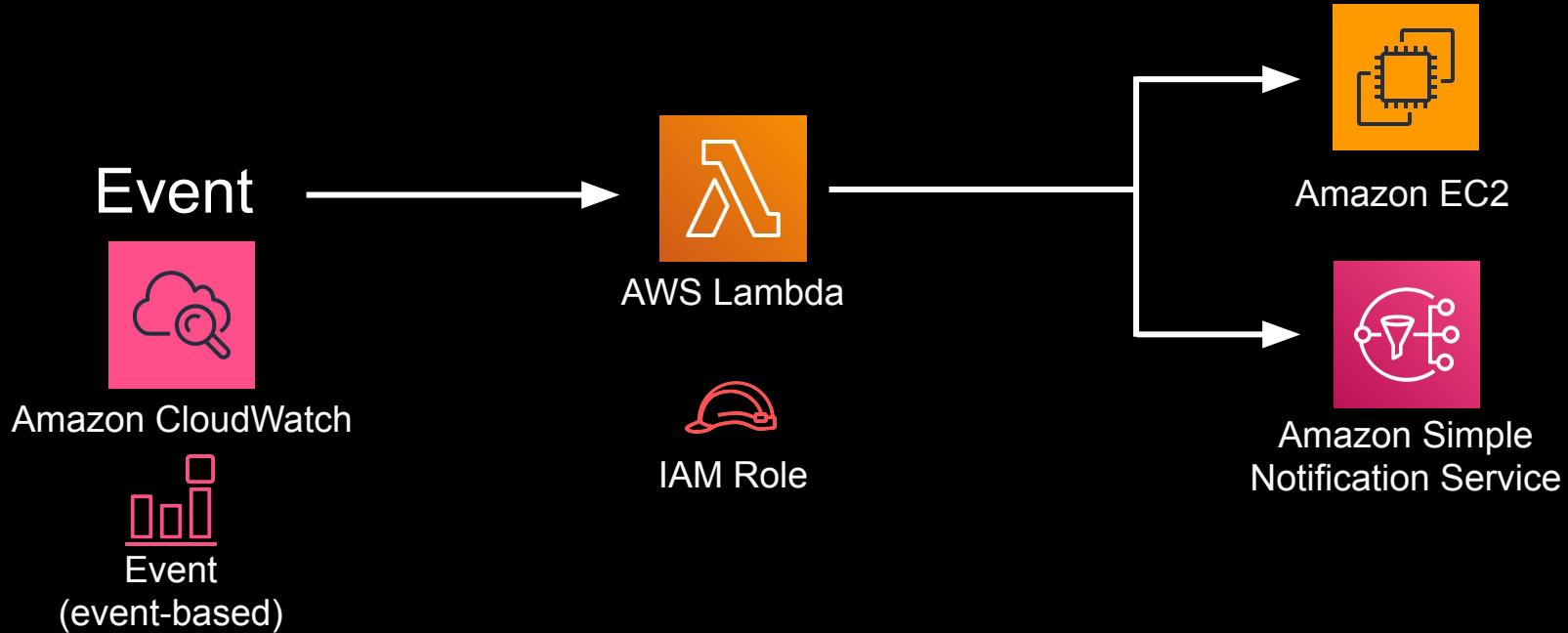
- IAM Prerequisites
  - IAM Role for RDS proxy
  - IAM Role for Lambda
- Everything within VPC
  - Proper Security Groups for Lambda to Proxy to RDS
- Lambda requires external dependency
- Demo shown for RDS MySQL

Code Any Boto3  
with Lambda

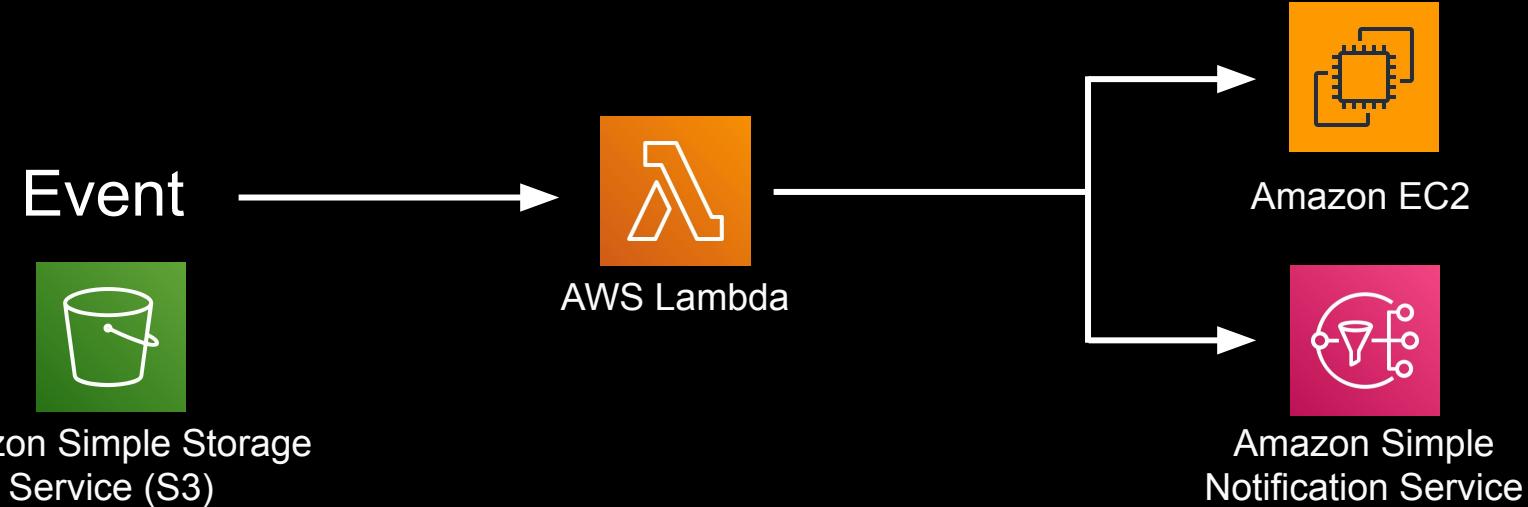
# What We Building

- EC2 Started without proper tag
- Stop EC2
- Send threatening email to employee!

# High Level Design



# High Level Design



# Boto3

- Boto3 is AWS Software Development Kit for Python
- Call ANY AWS Services from code
  - Code can be in EC2, Lambda, EKS etc.

# Let's Code



KEEP  
CALM  
AND  
LET'S  
CODE

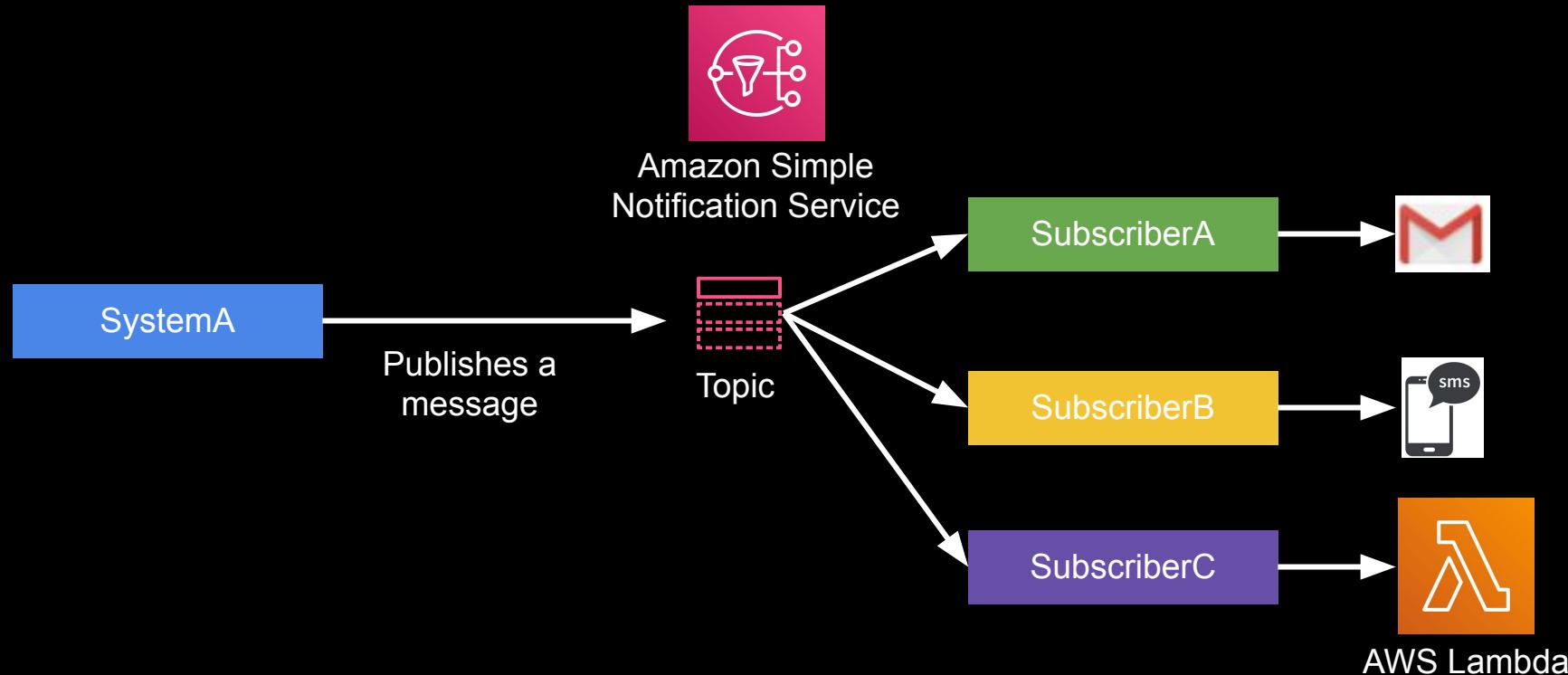
# Boto3 Verdict

- Boto3 Documentation is super good
- Practice doing using the documentation

# SNS, SQS, Lambda (The Perfect Love Triangle)

# What is SNS?

Amazon SNS (Simple Notification Service) is a highly available, durable, secure, fully managed pub/sub messaging service

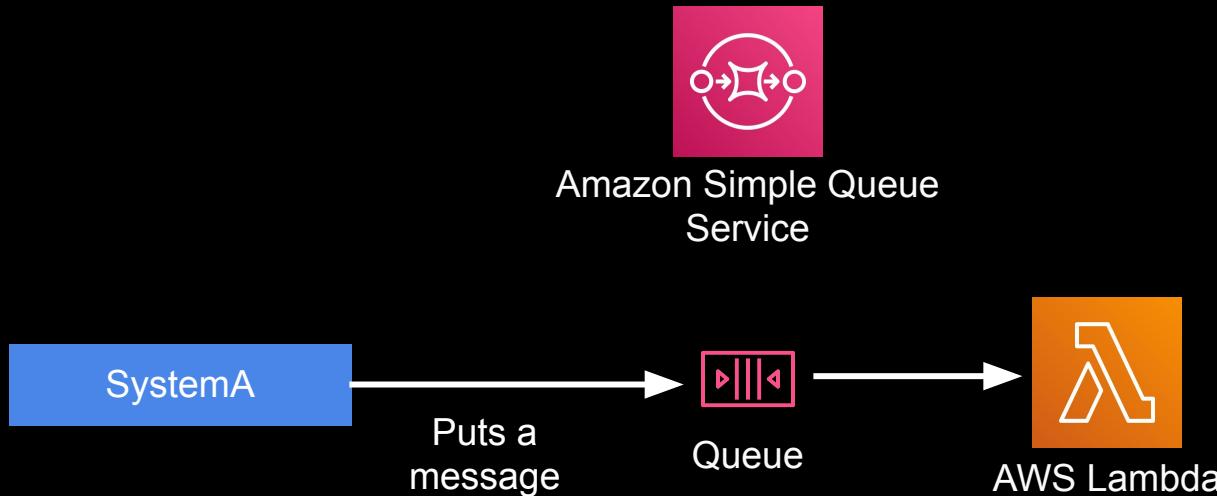


# Advantages of SNS

- Automatically scale
- Keep messages secure using AWS KMS Keys
- Messages can go to different subscribers based on fields in message (Message Filtering)
- Fan Out Architecture - Same message can be consumed by multiple consumers

# What is SQS?

Amazon SQS (Simple Queue Service) is a fully managed message queuing service



- Standard Queue
- FIFO Queue

# Standard Vs FIFO

## Standard

- Order is not guaranteed
- Messages may be delivered more than once
- Nearly unlimited messages processed per second
- \$0.40/million requests/month after free tier

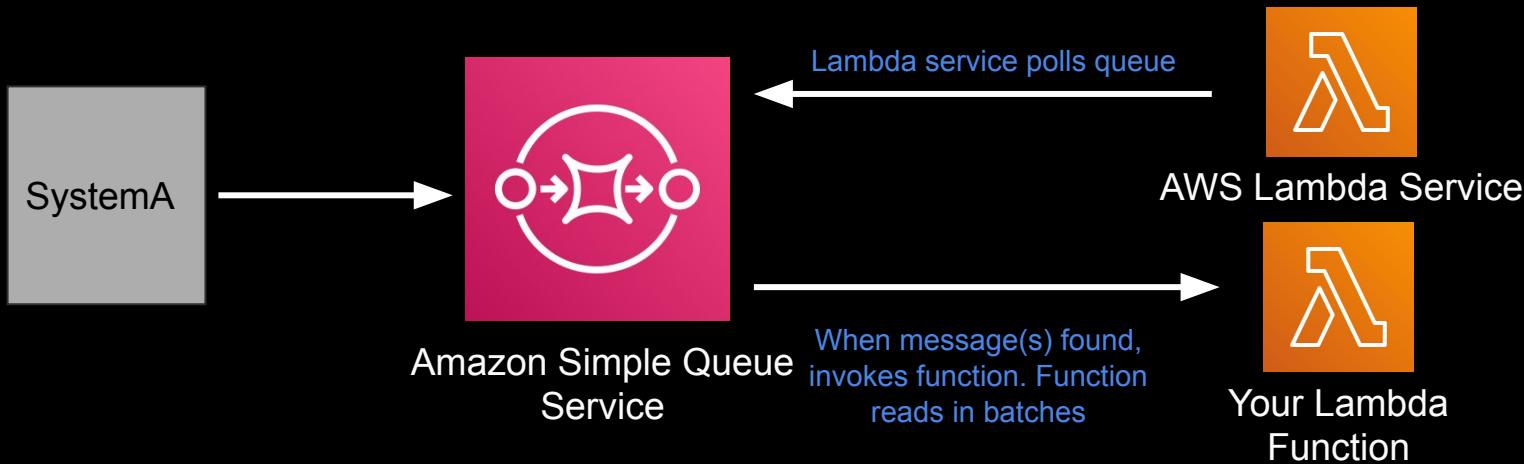
## FIFO

- Order is strictly preserved
- Messages follow exactly once processing, dedup configuration avoids duplicate message delivery
- 300 messages/second. With batching supported upto 3000 message/second
- \$0.50/million requests/month after free tier

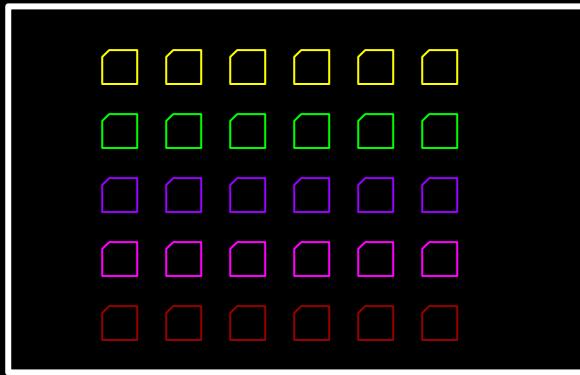
# Advantages of SQS

- Reliable, Dead Letter Queues can be enabled
- Scales automatically
- Keep messages secure using KMS
- Convert synchronous patterns to asynchronous.
- One message can't have multiple consumers. Once message is processed by consumer, it gets deleted from SQS

# Diving Deeper into SQS + Lambda



# Diving Deeper into SQS + Lambda



SQS Queue

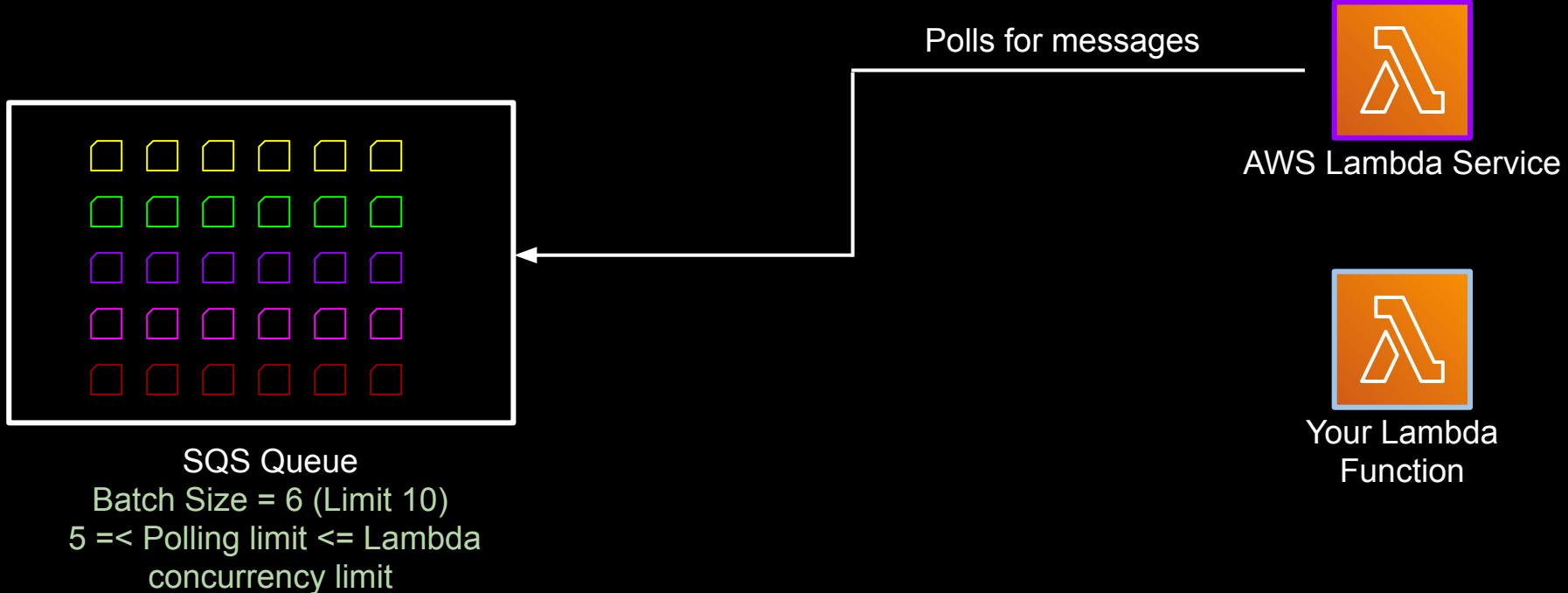


AWS Lambda Service

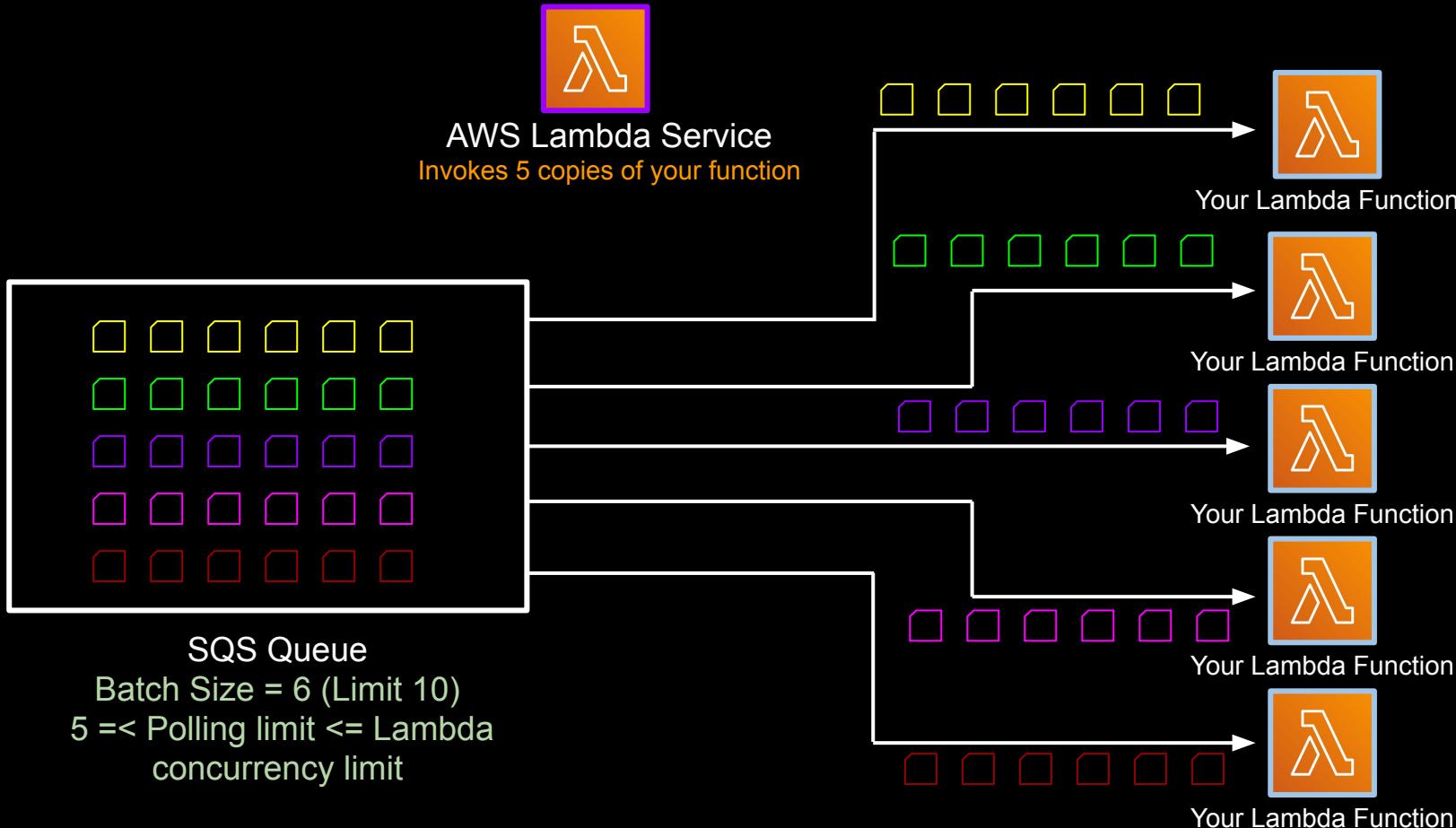


Your Lambda  
Function

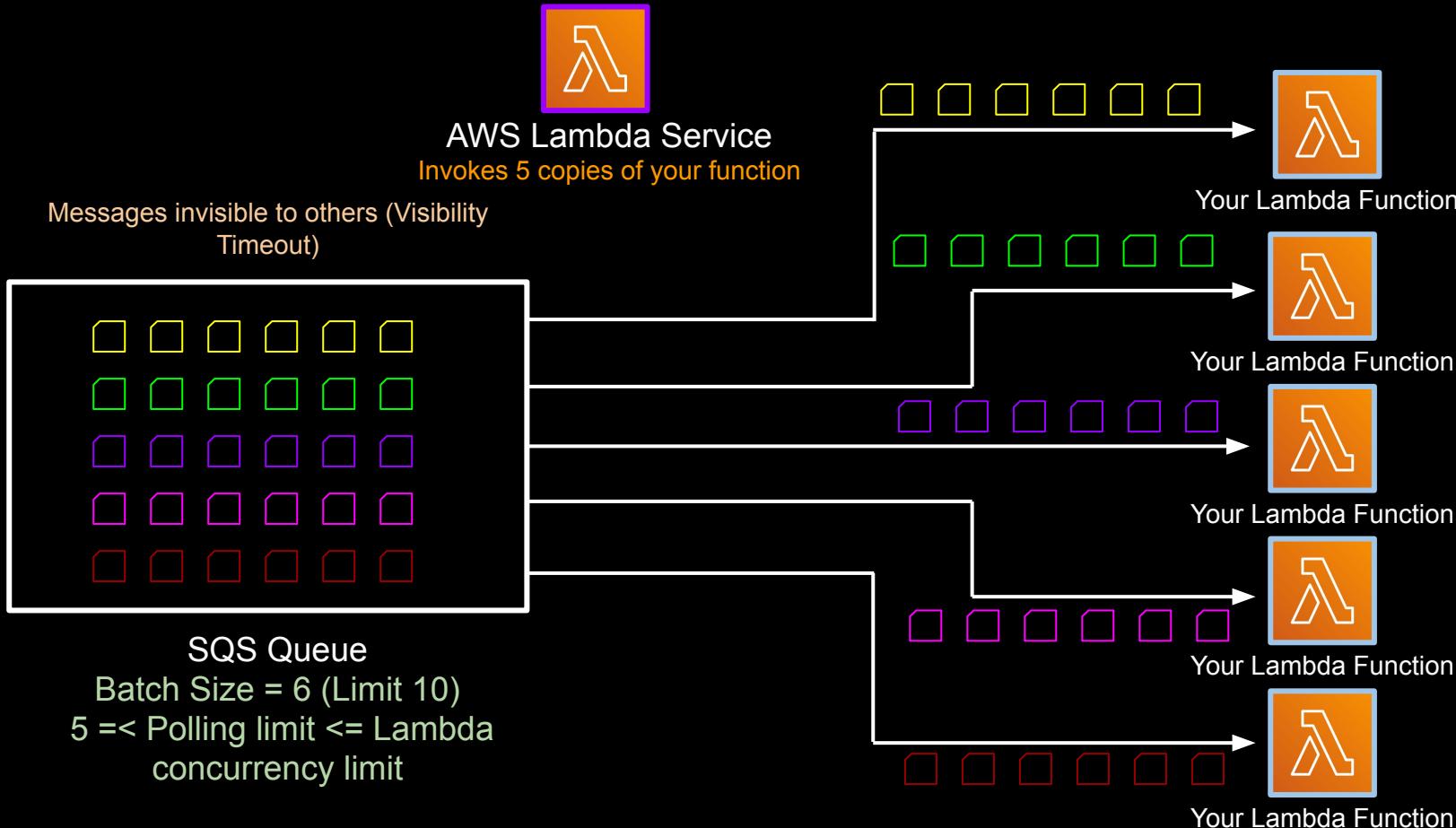
# Diving Deeper into SQS + Lambda



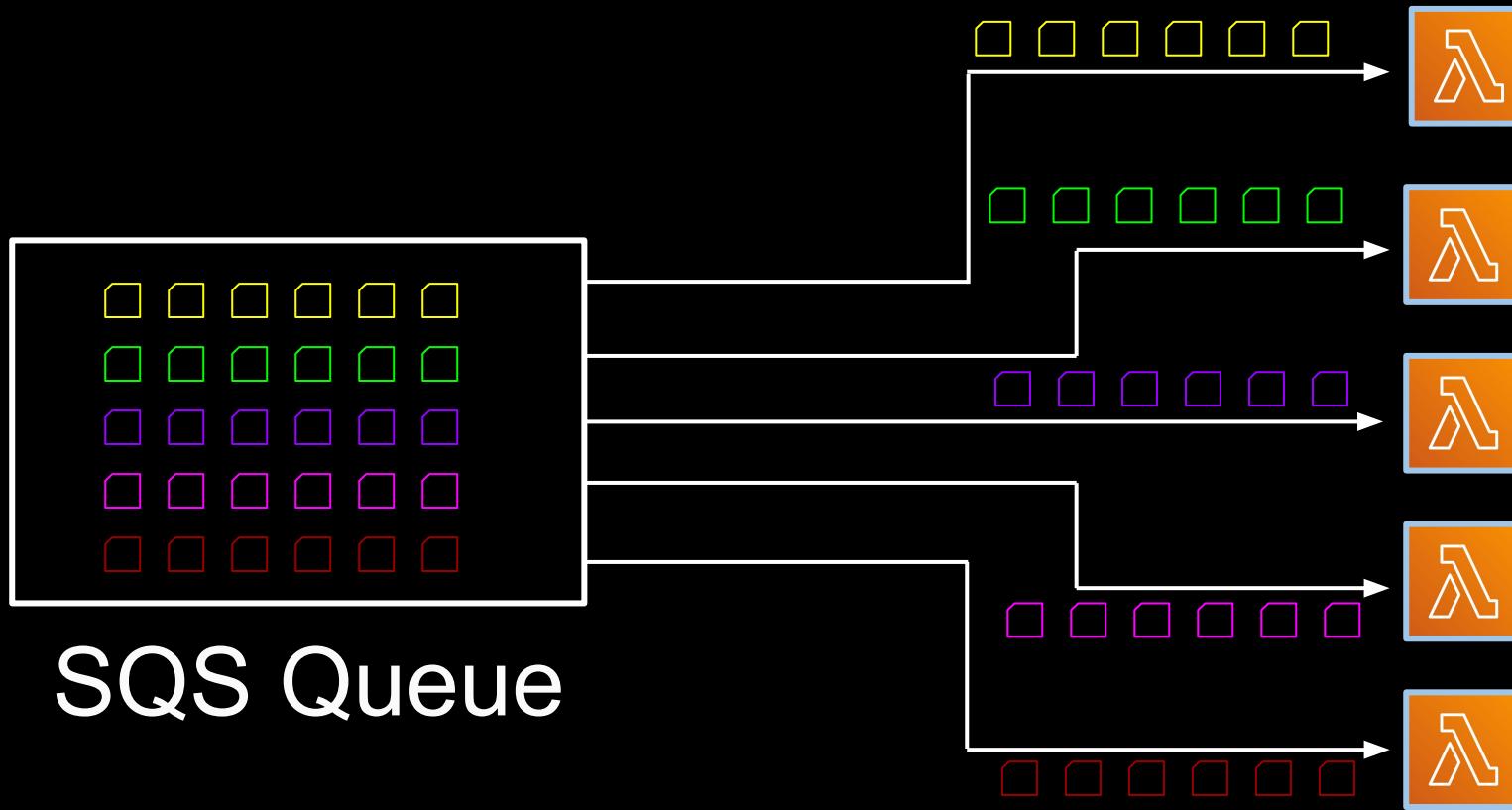
# Diving Deeper into SQS + Lambda



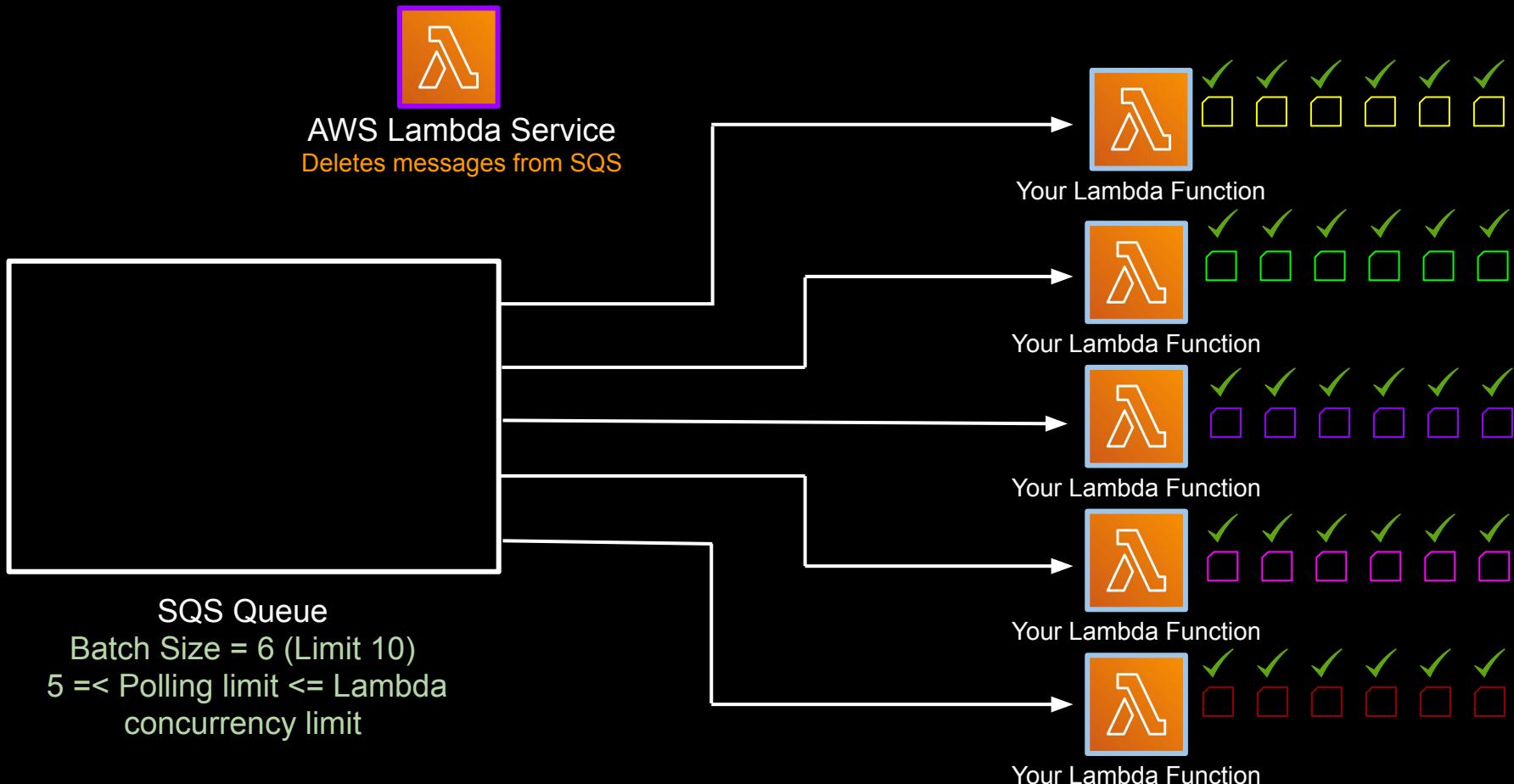
# Diving Deeper into SQS + Lambda



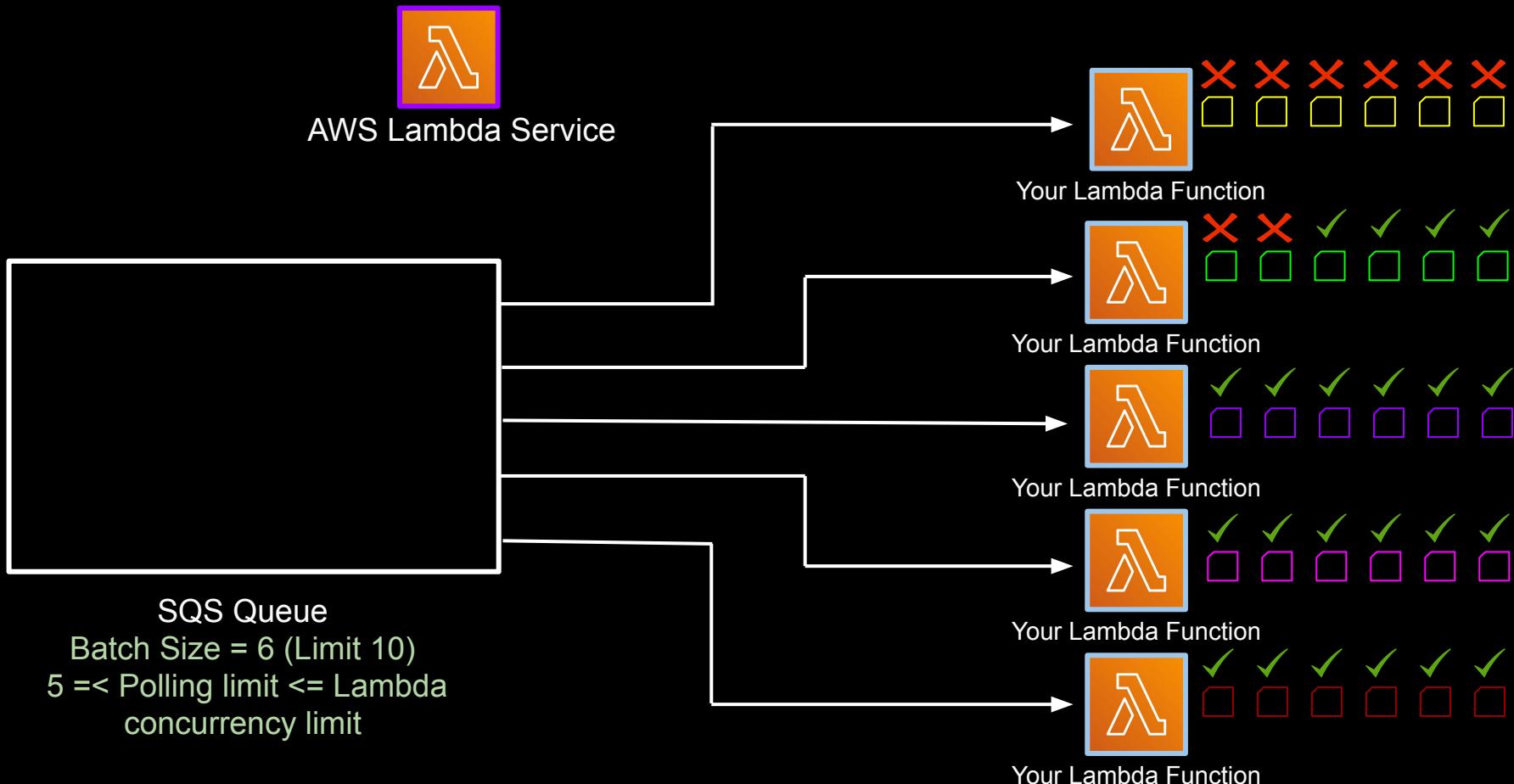
# SQS & SNS DEEPER DIVE



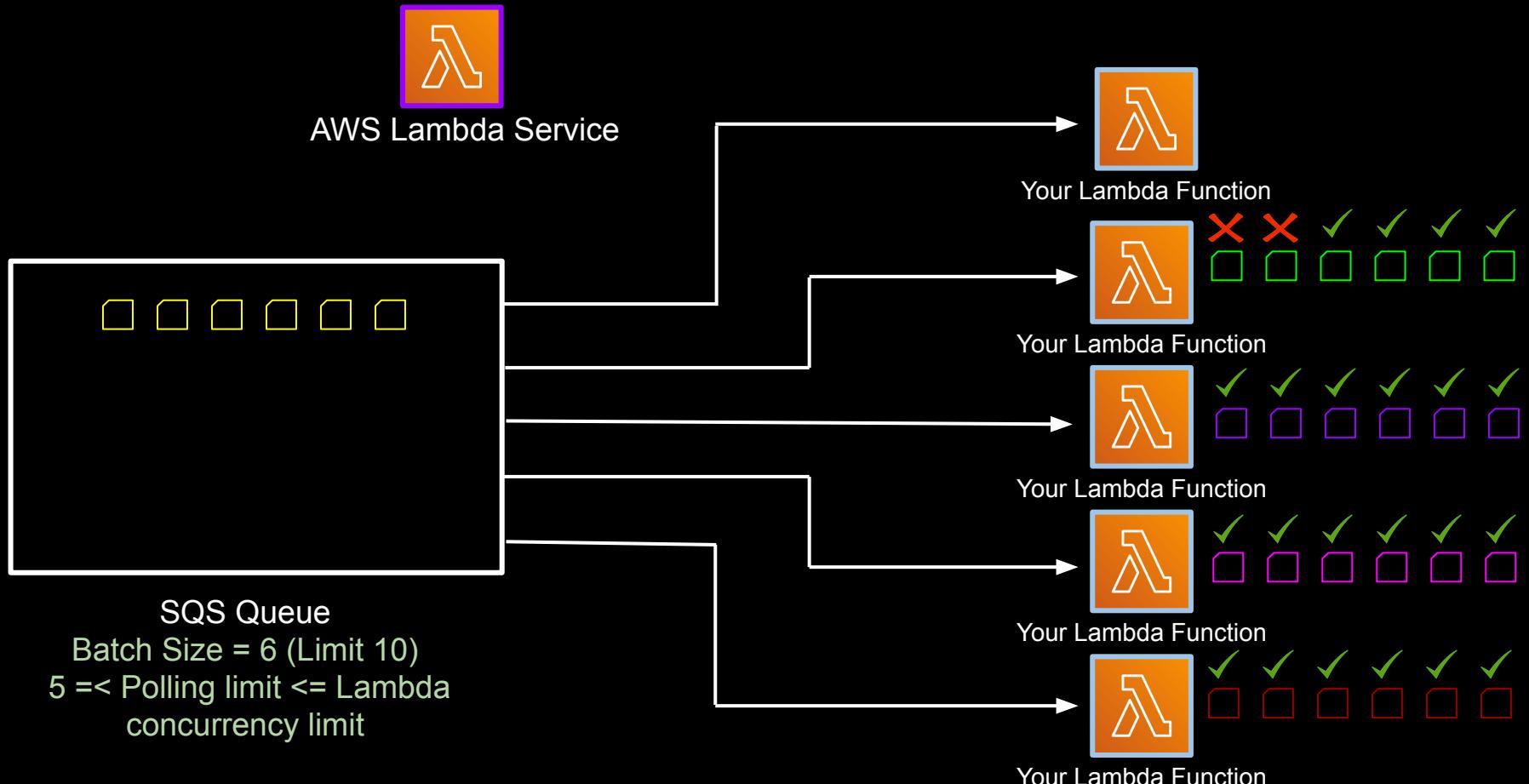
# Sunny Day Scenario



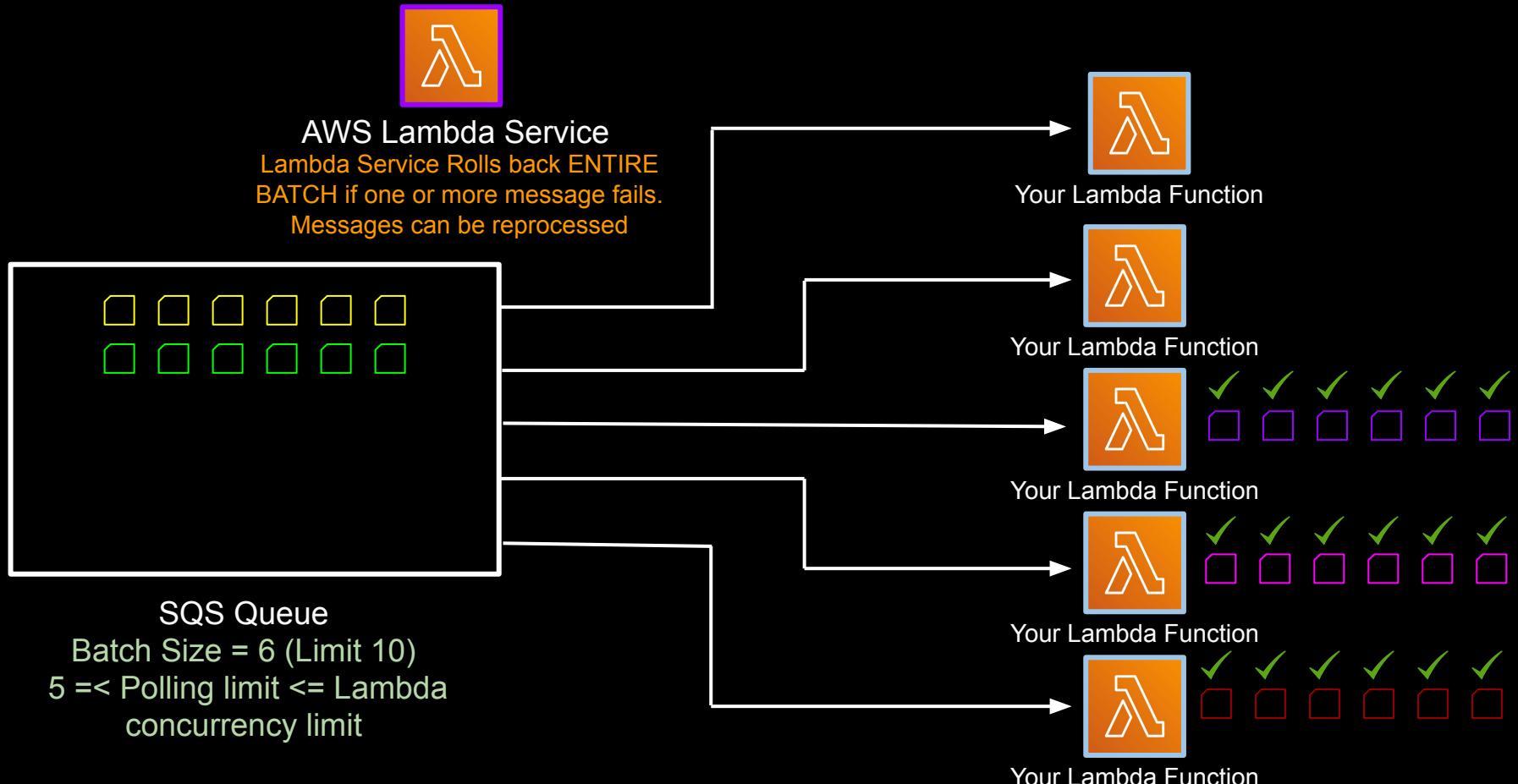
# Rainy Day Scenario



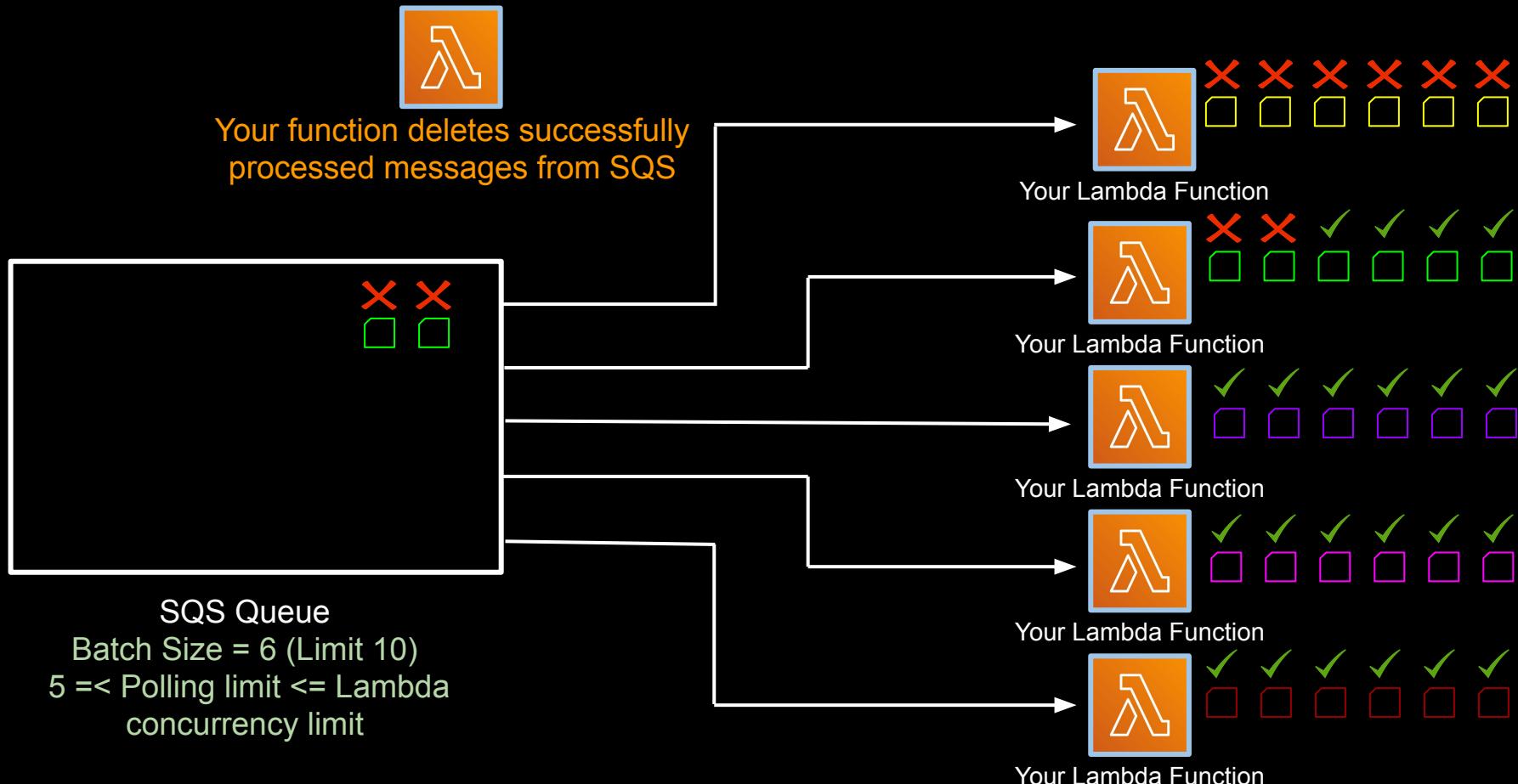
# Rainy Day Scenario



# Rainy Day Scenario



# Solving Message Re-Processing



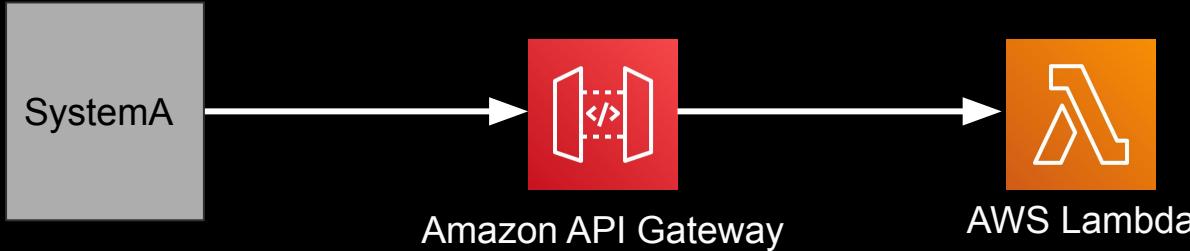
# Sync to Async: High Volume Traffic

System A calls a Lambda. Traffic volume very high.



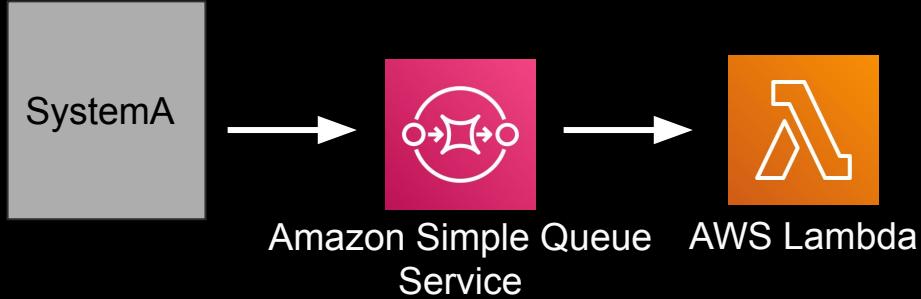
- In Synchronous Architecture, all components need to scale together
- Scaling is as high as scaling capacity of lowest scalable component
- Each component will keep running till the whole chain finishes
- If one component fails, whole call fails

# Sync to Async: High Volume Traffic



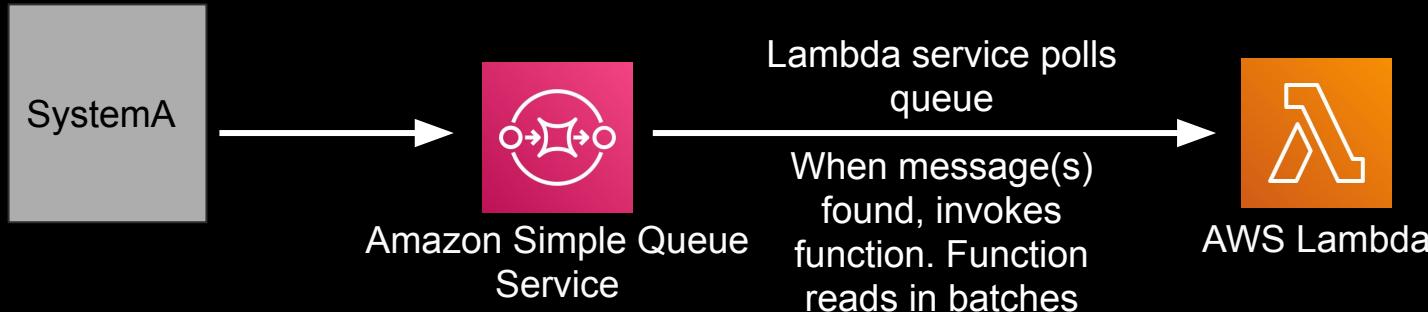
- In Synchronous Architecture, all components need to scale together
- Scaling is as high as scaling capacity of lowest scalable component
- Each component will keep running till the whole chain finishes
- If one component fails, whole call fails

# Sync to Async: High Volume Traffic



- In Asynchronous Architecture, all components can scale separately
- Less aggressive scaling requirement on Lambda
- Retry mechanism available even if one component fails
- Control traffic to downstream

# Tips For SQS & lambda



- Set function concurrency to 5 or more
- Less than 5 function concurrency can lead to throttling error
- Set queue's visibility timeout to at least 6 times the timeout of Lambda Function
- Configure Dead Letter Queue to keep messages to be reprocessed

# Food For Thought



- Can you break Sync to Async for high volume design?
- POST could be Async, GET could be Sync

# High Volume S3 Processing

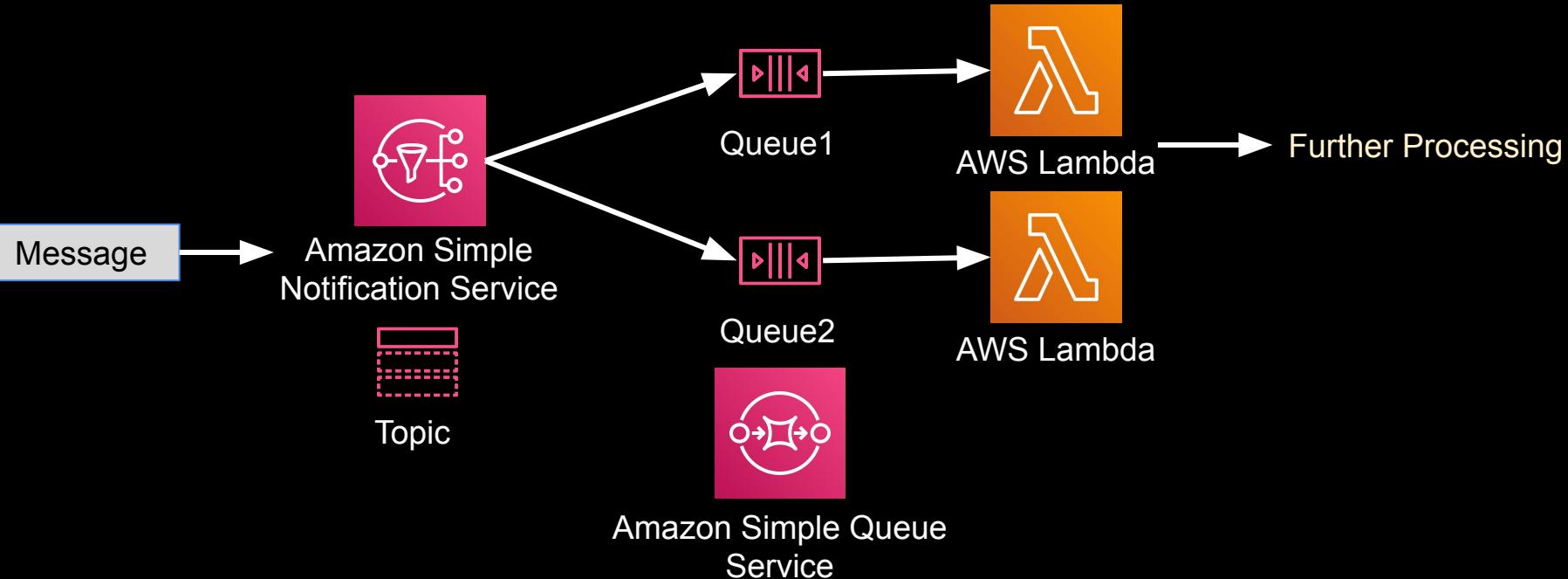


# Food For Thought

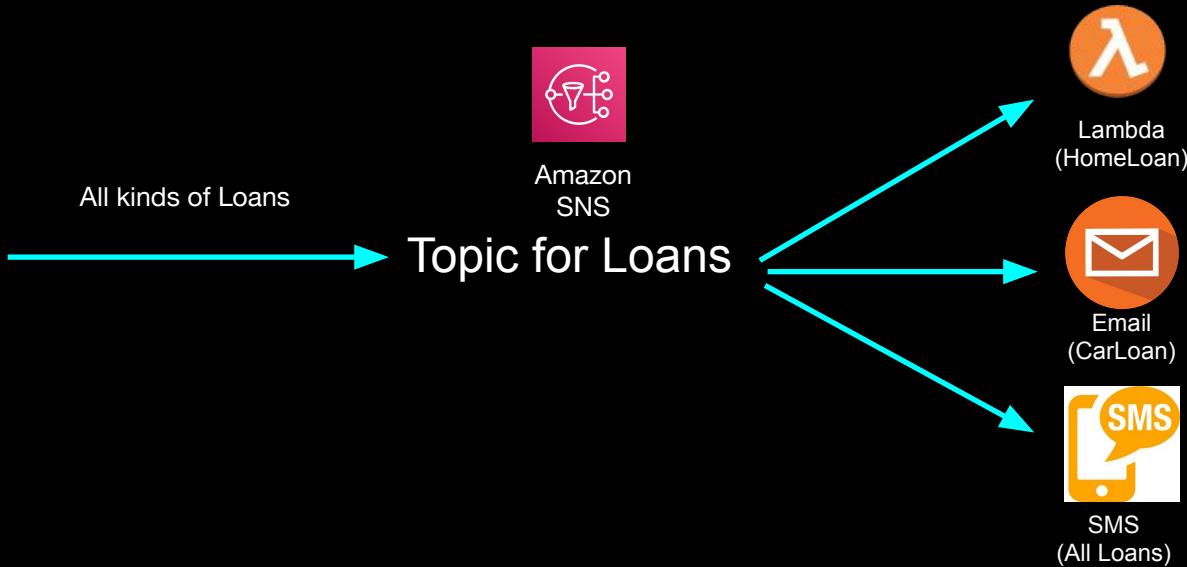


- Can you break Sync to Async for high volume design?
- POST could be Async, GET could be Sync

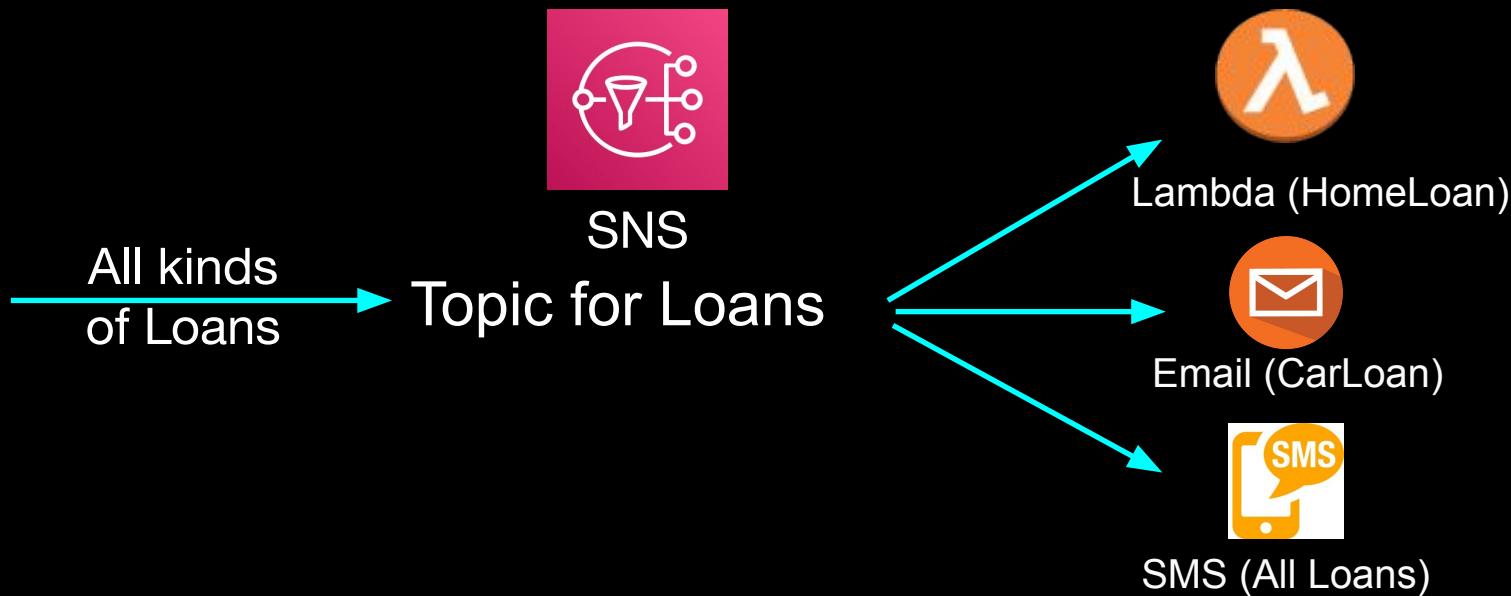
# Reliable Fanout Architecture



# SNS Message Filtering - What and Why



# SNS MESSAGE FILTERING



# EventBridge Vs SQS Vs SNS

# Basic definitions of SNS, SQS, EventBridge

## Differences:

Scaling

Conditional Message Processing

Message Replay

Message Ordering

Encryption

Durability

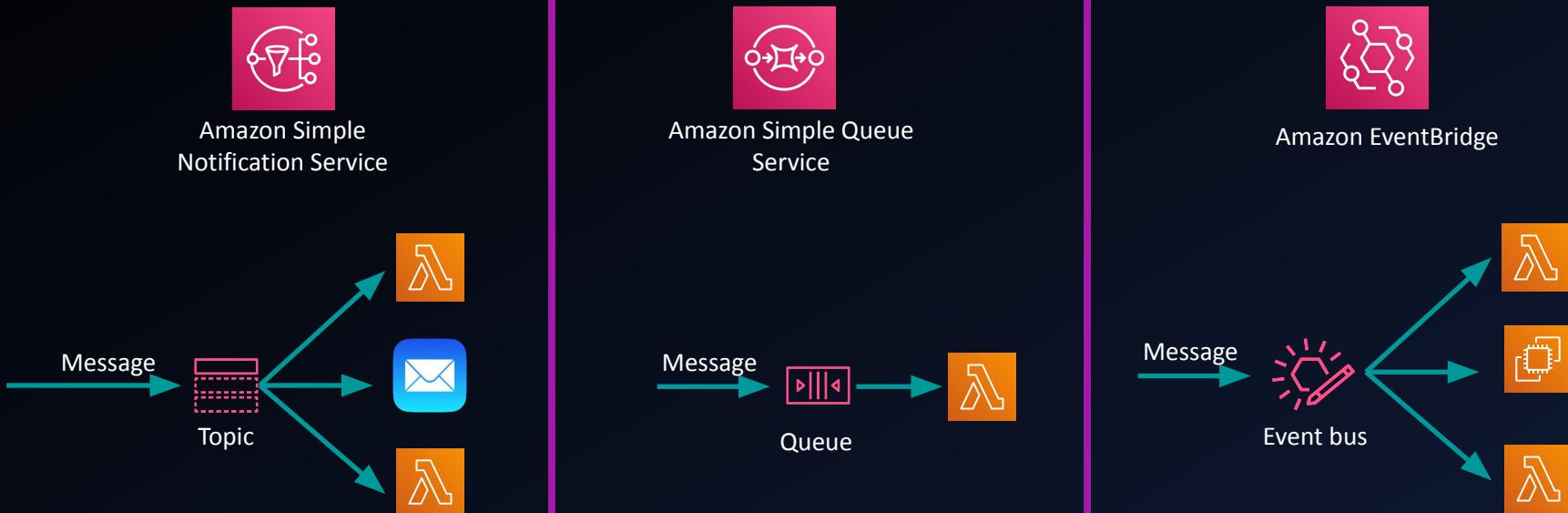
Pricing

Persistence

Consumption

Retry/Failure Handling

# Level Set



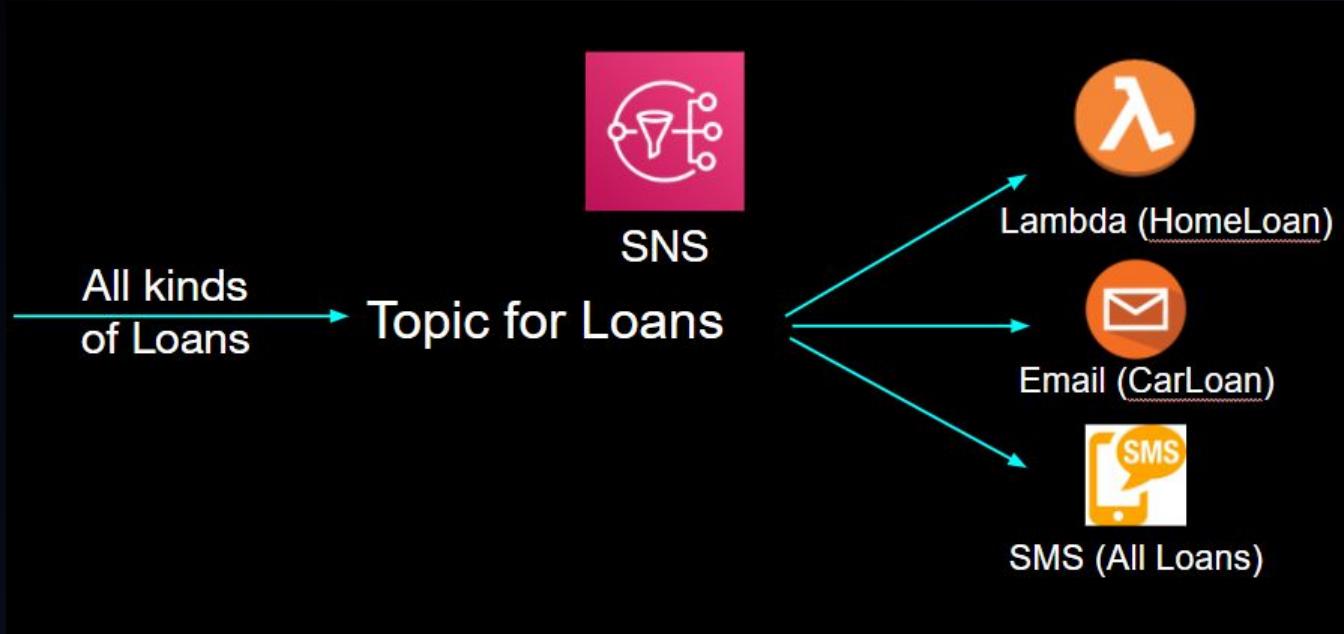
# Scaling/Concurrency Controls

Service	Scaling controls
<b>SNS</b>	Service automatically scales, use Lambda per function concurrency setting to control downstream consumption.
<b>SQS</b>	Service automatically scales, use Lambda trigger Batch size setting and Per Function Concurrency setting to control downstream consumption.
<b>EventBridge</b>	Service automatically scales automatically up with default soft quotas of 400 PutEvents and 750 target invocations requests per second (can be increased). Use Lambda per function concurrency setting to control downstream consumption.

# Conditional Message Processing

Service	Conditional message processing
SNS	Can invoke different subscriber based on values on message metadata using SNS message filtering
SQS	SQS can't decide consumer based on message. Use SNS message filtering with SQS to achieve this
EventBridge	Event filtering can route messages to targets based on message. Can transform events before sending to target. Contains schema registry.

# SNS Message Filtering



# Amazon EventBridge Schema

```
{  
    "version": "0",  
    "id": "315c1398-40ff-a850-213b-158f73e60175",  
    "detail-type": "Step Functions Execution Status Change",  
    "source": "aws.states",  
    "account": "012345678912",  
    "time": "2019-02-26T19:42:21Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:states:us-east-1:012345678912:execution:state-machine-name"  
    ],  
    "detail": {  
        "executionArn": "arn:aws:states:us-east-1:012345678912:execution:  
        "stateMachineArn": "arn:aws:states:us-east-1:012345678912:stateM  
        "name": "execution-name",  
        "status": "FAILED",  
        "startDate": 1551225146847,  
        "stopDate": 1551225151881,  
        "input": "{}",  
        "output": null  
    }  
}
```

Messag  
e

```
"components": {  
    "schemas": {  
        "AWSEvent": {  
            "type": "object",  
            "required": ["detail-type", "resources", "detail", "id", "source",  
            "x-amazon-events-detail-type": "Step Functions Execution Status  
            "x-amazon-events-source": "aws.states",  
            "properties": {  
                "detail": {  
                    "$ref": "#/components/schemas/StepFunctionsExecutionStatusCh  
                },  
                "account": {  
                    "type": "string"  
                },  
                "detail-type": {  
                    "type": "string"  
                },  
                "id": {  
                    "type": "string"  
                }  
            }  
        }  
    }  
}
```

Schem  
a

# Amazon EventBridge Schema Registry

- Store schemas in a central location
- Download code bindings to use in code quickly
- Predefined schemas available
- Can automatically discover schemas from messages

# Amazon EventBridge

[Create registry](#)[Create schema](#)

## Schemas Info

A schema defines the structure and content of events that are passed on an event bus in Amazon EventBridge. You can browse or search for the schemas of all AWS services on EventBridge. You can automatically generate schemas for events on an event bus, create or upload custom schemas, and organize your custom schemas in custom registries.

[All schemas](#)[AWS event schema registry](#)[Discovered schema registry](#)[Custom schema registry](#)

### Search AWS event schemas



< 1 2 3 4 5 6 7 8 ... >

[aws.a4b@RoomStateChange](#)

AWS event schema registry 1 version

Last updated Oct 23, 2020, 02:39 PM EDT

[aws.athena@AthenaQueryStateCha...](#)

AWS event schema registry 1 version

Last updated Jan 16, 2020, 04:27 PM EST

[aws.autoscaling@AWSAPICallViaClo...](#)

AWS event schema registry 1 version

Last updated Nov 30, 2019, 07:59 PM EST

[aws.autoscaling@EC2InstanceStateLaunc...](#)

AWS event schema registry 1 version

Last updated Nov 30, 2019, 07:59 PM EST

[aws.autoscaling@EC2InstanceStateLaunc...](#)

AWS event schema registry 1 version

Last updated Nov 30, 2019, 07:59 PM EST

[aws.autoscaling@EC2InstanceStateLaunc...](#)

AWS event schema registry 1 version

Last updated Nov 30, 2019, 07:59 PM EST

Events

Event buses

Rules

Partner event sources

Archives

Replays

Schema registry

[Schemas](#)

Documentation

# Amazon EventBridge Pattern Matching Rules

```
{  
    "time": [ { "prefix": "2017-10-02" } ],  
}
```

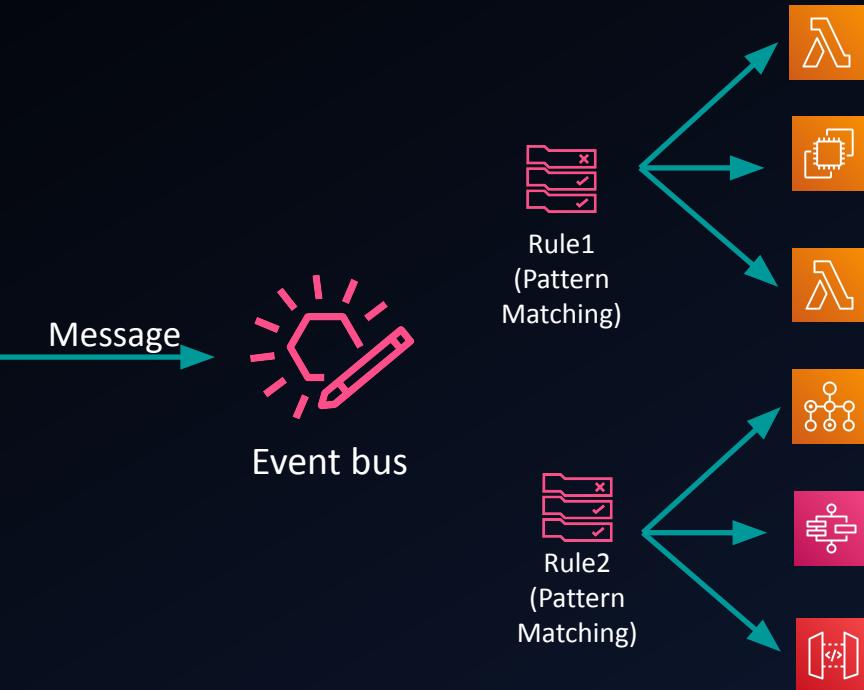
```
{  
    "detail": {  
        "c-count": [ { "numeric": [ ">", 0, "<=", 5 ] } ],  
        "d-count": [ { "numeric": [ "<", 10 ] } ],  
        "x-limit": [ { "numeric": [ "=", 3.018e2 ] } ]  
    }  
}
```

```
{  
    "source": [ { "anything-but": [ "aws.ec2", "aws.s3" ] } ]  
}
```

# Amazon EventBridge Message Transformation

Description	Template	Output
<b>Simple string</b>	"instance <instance> is in <state>"	"instance i-0123456789 is in RUNNING"
<b>String with escaped quotes</b>	"instance \"<instance>\" is in <state>"	"instance \"i-0123456789\" is in RUNNING"
<b>Simple JSON</b>	{ "instance" : <instance>, "state": <state> }	{ "instance" : "i-0123456789", "state": "RUNNING" }
<b>JSON with a mix of variables and static information</b>	{ "instance" : <instance>, "state": [ 9, <state>, true ], "Transformed" : "Yes" }	{ "instance" : "i-0123456789", "state": [ 9, "RUNNING", true ] }

Note that this is the behavior in the EventBridge console. The AWS CLI escapes the slash characters and the result is "instance "i-0123456789" is in RUNNING".



# EventBridge Integration Partners



pagerduty



And many more...

# Message Replay

Service	Conditional message processing
SNS	Messages are gone once delivered to subscribers. No replay functionality
SQS	Messages are gone once delivered to subscribers. No replay functionality
EventBridge	Messages can be archived based on rules. Can be replayed later

# Message Order

Service	Message ordering
SNS	SNS FIFO maintains order (New)
SQS	SQS FIFO queue maintains order
EventBridge	Message order not maintained

# Encryption and Compliance

Service	Encryption at rest and compliance
SNS	Messages at rest can be encrypted using KMS. Both customer managed and Amazon managed CMKs are supported FedRAMP High, HIPAA compliant
SQS	Messages at rest can be encrypted using KMS. Both customer managed and Amazon managed CMKs are supported FedRAMP High, HIPAA compliant
EventBridge	Messages at rest can't be encrypted using KMS HIPAA compliant. Check <a href="https://aws.amazon.com/compliance/services-in-scope/">https://aws.amazon.com/compliance/services-in-scope/</a> for status updates

# Durability

Service	Durability
SNS	SNS stores all messages within a single, highly-available AWS region with multiple redundant Availability Zones (AZs)
SQS	SQS stores all messages within a single, highly-available AWS region with multiple redundant Availability Zones (AZs)
EventBridge	EventBridge stores all messages within a single, highly-available AWS region with multiple redundant Availability Zones (AZs)

# Pricing

Service	Model	Cost Per Mil	Factor
<b>SNS</b>	Per request	\$0.50*	Each 64KB chunk of delivered data is billed as 1 request
<b>SQS</b>	Per request	\$0.40*	Each 64 KB chunk of a payload is billed as 1 request
<b>EventBridge</b>	Per request	\$1.00 – SaaS, custom, and cross-account events  Free – same-account AWS events	Each request can be up to 256 KB in size

# Persistence

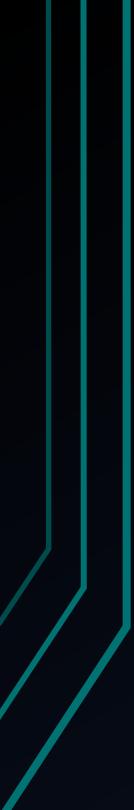
Service	Persistence of requests
SNS	No formal persistence model beyond delivery retry logic that extends up through potentially 23 days.
SQS	By default messages are stored for 4 days. This can be modified to as little as 60 seconds up to 14 days by configuring a queue's <b>MessageRetentionPeriod</b> attribute
EventBridge	No formal persistence model beyond delivery retry logic that extends up through potentially 24 hours

# Consumption

Service	Invocation model	Guidance
SNS	Consumer – Lambda, SQS, email, mobile push, SMS, HTTP Async to Lambda. SNS can “fanout” to multiple subscribing Lambda functions the same message	Use Message Filtering to control which messages go to which subscribers. Use Message delivery status to track failures
SQS	Consumer – Lambda, any service that can run AWS SDK (EC2, EKS etc.) Lambda service polls messages from queue and invokes Lambda on your behalf. Scales polling based on inflight messages.	Can call message delete from within your code or let the service handle it via successful Lambda function execution
EventBridge	Consumer – Lambda, EC2, Step Functions, API Gateway etc. Async to Lambda. EventBridge can “fanout” to have up to 5 targets per rule and multiple rules on the same event source. 300 rules per event bus.	Use event patterns set on rules to control which events are subscribed to by different rules.

# Retry/Failure handling

Service	Retry/failure capabilities
<b>SNS</b>	If Lambda is not available, SNS will retry 2 times at 1 seconds apart, then 10 times exponentially backing off from 1 seconds to 20 minutes and finally 38 times every 20 minutes for a total 50 attempts over more than 23 days before the message is discarded. Use with SQS for DLQ.
<b>SQS</b>	Messages remain in the queue until deleted. They are prevented by being accessed by other consumers during the “visibility timeout”. Successful Lambda invocations will cause deletions of messages automatically. If an invocation fails or doesn’t delete a message during the visibility timeout, it becomes available for other consumers. DLQ can be used.
<b>EventBridge</b>	If Lambda or other target service is not available, EventBridge will automatically retry delivery with backoff up to 24 hours before message is discarded from EventBridge. As soon as successful message delivery to target occurs, message is discarded. Use with SQS for DLQ(New)



Which one should you choose?

**Event producer**

**Message order**

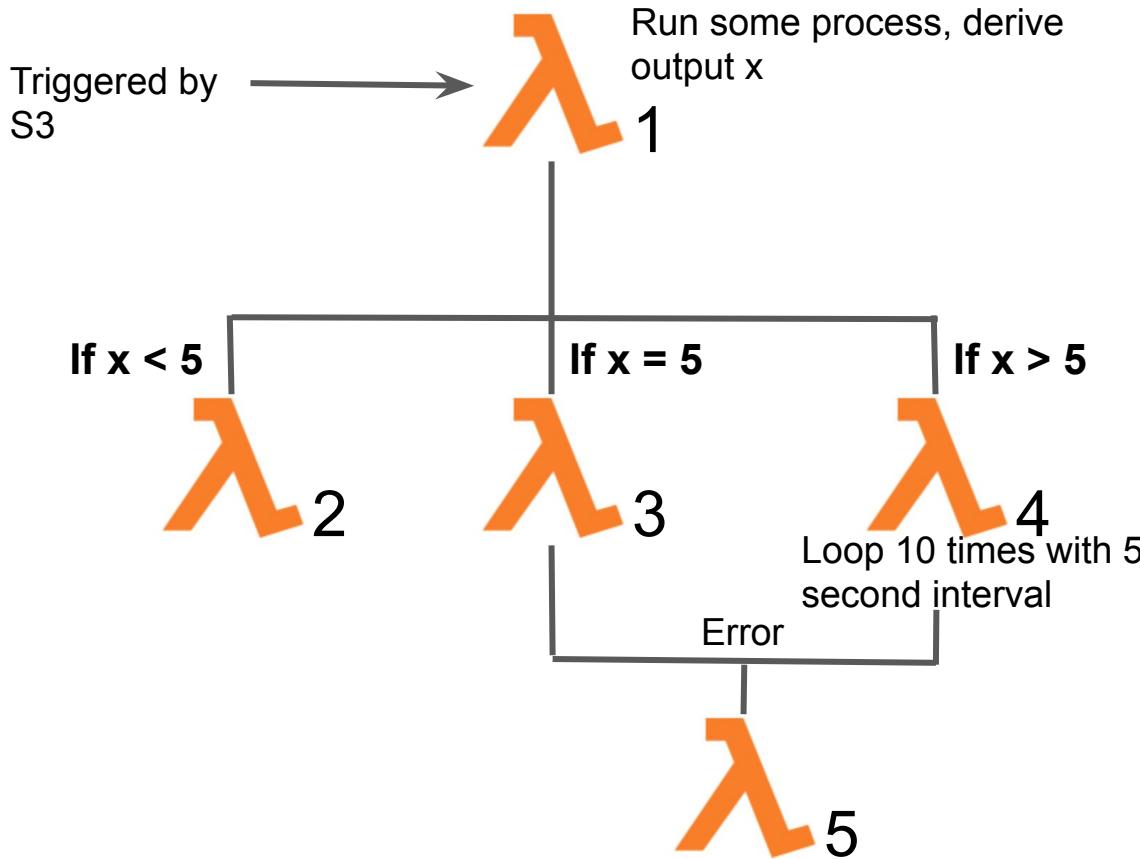
**Scaling requirement**

**Security consideration**

**Cost calculation**

**Consumer**

# AWS Step Function - Why and What?



- Lots of coding for flow control in Lambda
  - If the flow changed, lambda needs to be changed and retested
  - Not easy to change the flow
- 
- Step Function takes care of all the coordination and flow control
  - Create/Change flow in Visual Console
  - Lambdas become cleaner

# Key Components of AWS Step Function

```
"HelloWorld": {  
    "Type": "Task",  
    "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",  
    "Next": "AfterHelloWorldState",  
    "Comment": "Run the HelloWorld Lambda function"  
}
```

## States

States are elements in your state machine.

## Tasks

All work in your state machine is done by *tasks*. A task can be an activity or a Lambda function.

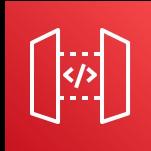
## Transitions

Tells the State what to do Next or where to Begin.

# Step Function Workflow Types

- Standard And Express Workflows
- Standard - long running, slightly higher latency workflows
- Express - high-volume, low latency, lower duration (Express!) workflows
- Once selected for a State Machine, can't be change afterwards for that State Machine
- Console is great to help you choose!

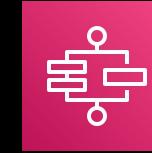
# How To Trigger Step Functions



Amazon API Gateway



Amazon CloudWatch Events

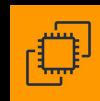


AWS Step Functions



Amazon EventBridge

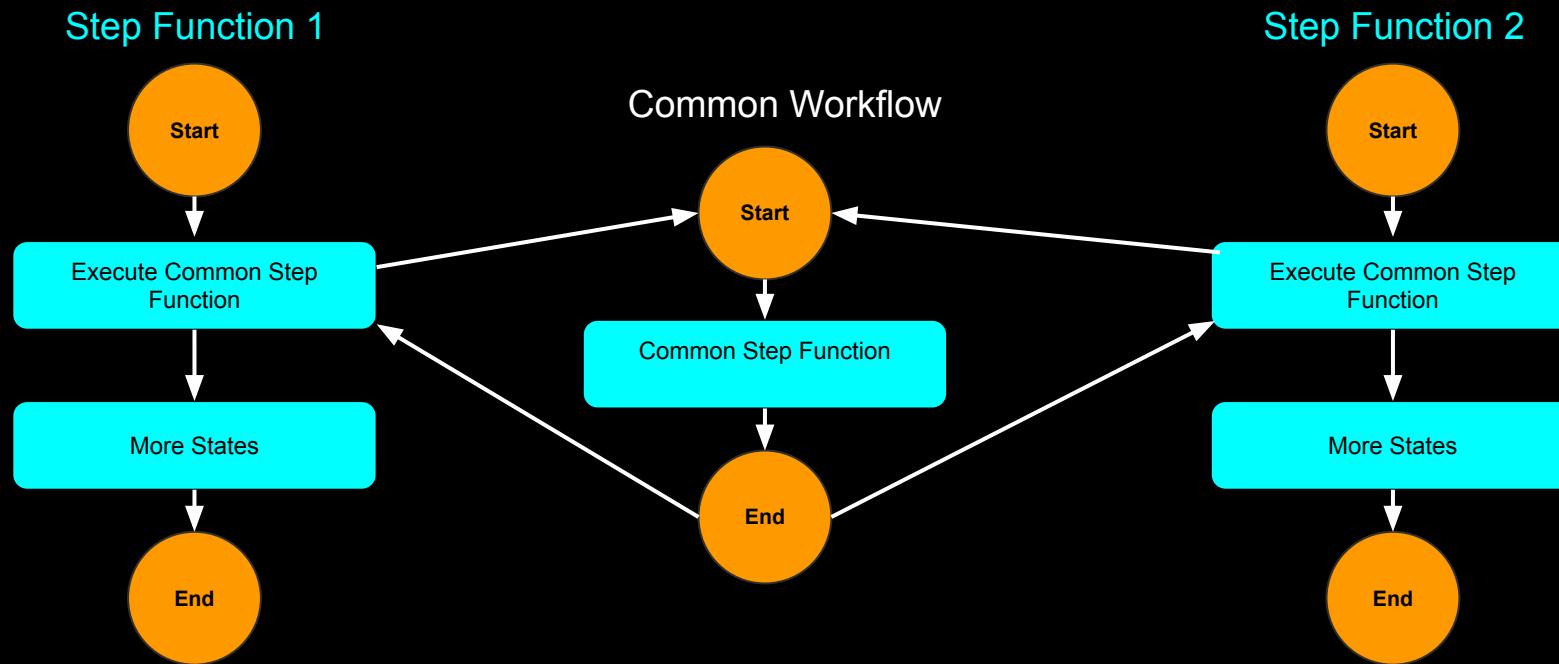
Step Function API  
(Basically any code!)



Many more..

# Nested Workflows

- Reuse common workflows, without copy pasting into multiple workflows
- Standard Workflow can call Express Workflow and vice versa

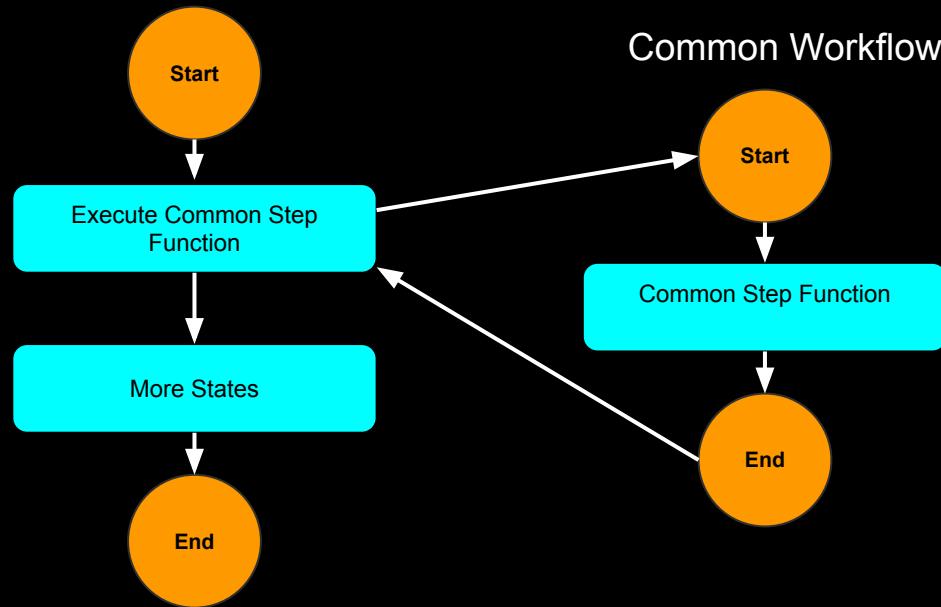


# Nested Workflows Tricky Note

- Keep in mind Request Response vs Sync

```
"Type": "Task",
"Resource": "arn:aws:states:::states:startExecution.sync:2",
"Parameters": {
    "StateMachineArn": "arn:aws:states:us-west-2:719217631821:stateMachine:ValidateAccount",
```

Step Function 1

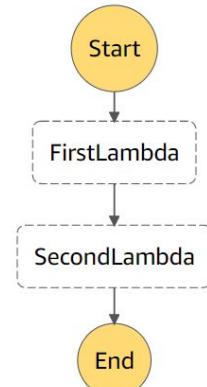


# Lambda Example

Generate code snippet ▾

Format JSON

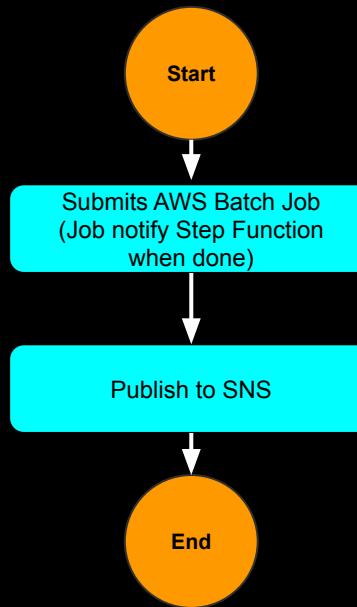
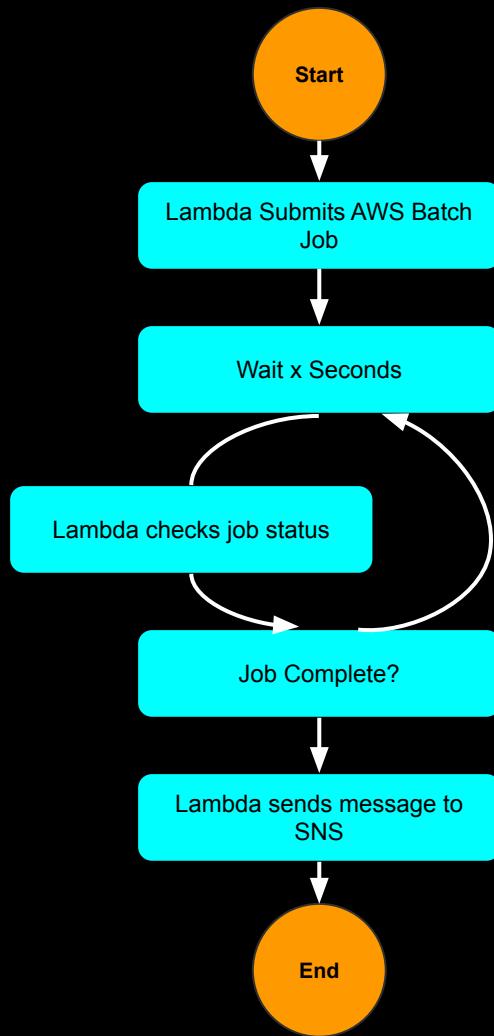
```
1 {  
2     "Comment": "Sample Lambda Sequence Step Functions Demo",  
3     "StartAt": "FirstLambda",  
4     "States": {  
5         "FirstLambda": {  
6             "Type" : "Task",  
7             "Resource": "arn:aws:lambda:us-east-1:719217631821:function:findCountryFromAddress",  
8             "InputPath": "$",  
9             "Next": "SecondLambda"  
10        },  
11        "SecondLambda": {  
12            "Type" : "Task",  
13            "Resource": "arn:aws:lambda:us-east-1:719217631821:function:greetFromCountry",  
14            "End": true  
15        }  
16    }  
17}  
18
```



# Service Integration

Supported Service Integrations			
Service	Request Response	Run a Job (.sync)	Wait for Callback (.waitForTaskToken)
Lambda	✓		✓
AWS Batch	✓	✓	
DynamoDB	✓		
Amazon ECS/AWS Fargate	✓	✓	✓
Amazon SNS	✓		✓
Amazon SQS	✓		✓
AWS Glue	✓	✓	
Amazon SageMaker	✓	✓	
Amazon EMR	✓	✓	
CodeBuild	✓	✓	
AWS Step Functions	✓	✓	✓

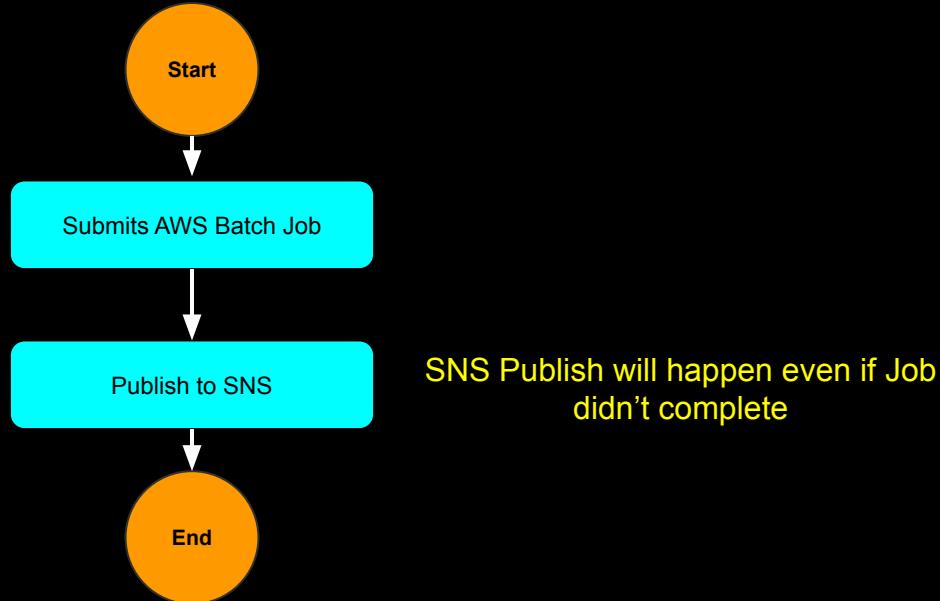
# Advantage



- Less code to maintain, no Lambda
- Out of the box integration
- Supports multiple integration patterns

# Request Response

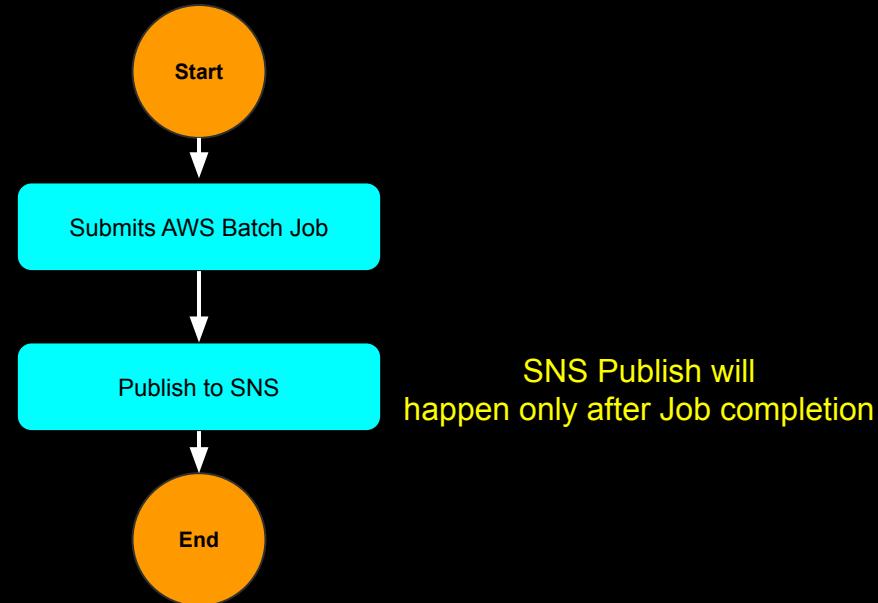
- Call service, gets HTTP response, goes to next state immediately
  - Don't wait for processing



# Run a Job (.sync)

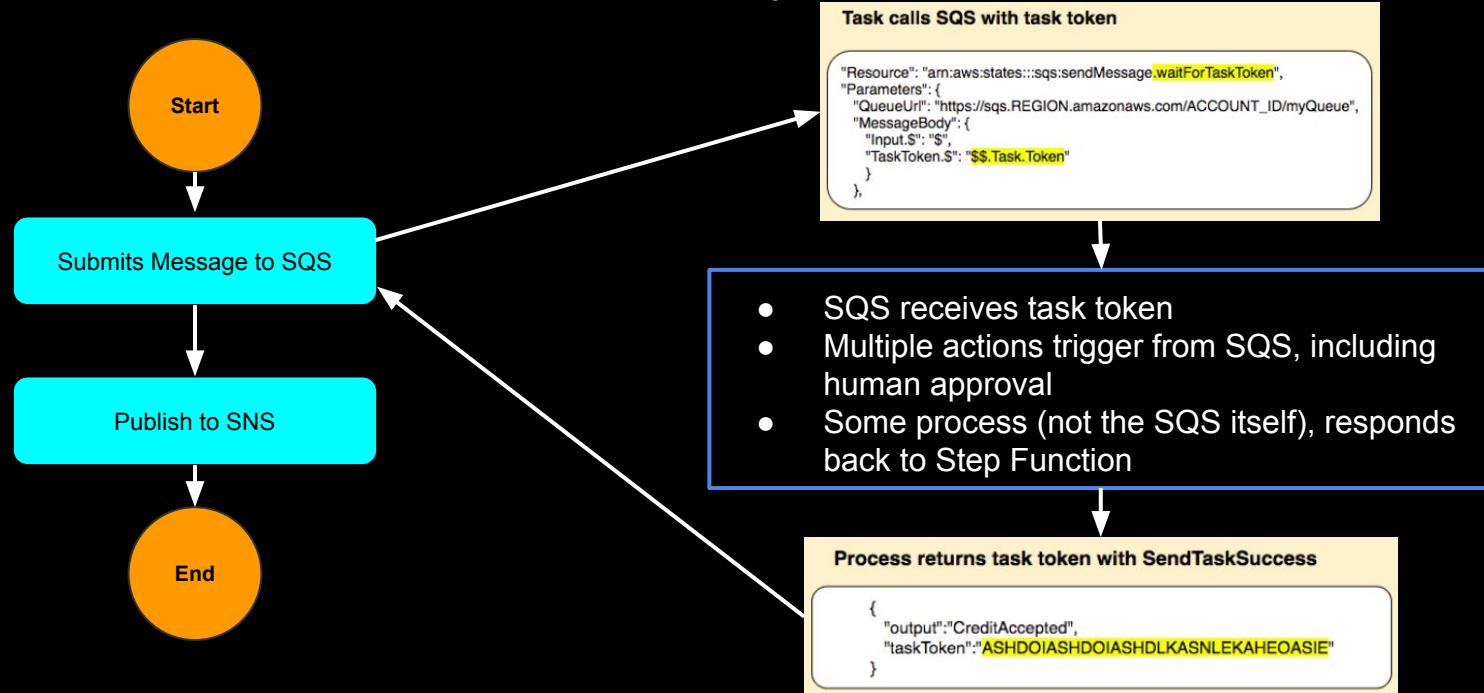
- Step Function wait for the processing to complete before proceeding

```
"Manage Batch task": {  
  "Type": "Task",  
  "Resource": "arn:aws:states:::batch:submitJob.sync",  
  "Parameters": {  
    "JobDefinition": "arn:aws:batch:us-east-2:123456789012:job-definition/testJobDefinition",  
    "JobName": "testJob",  
    "JobQueue": "arn:aws:batch:us-east-2:123456789012:job-queue/testQueue"  
  },  
  "Next": "NEXT_STATE"  
}
```



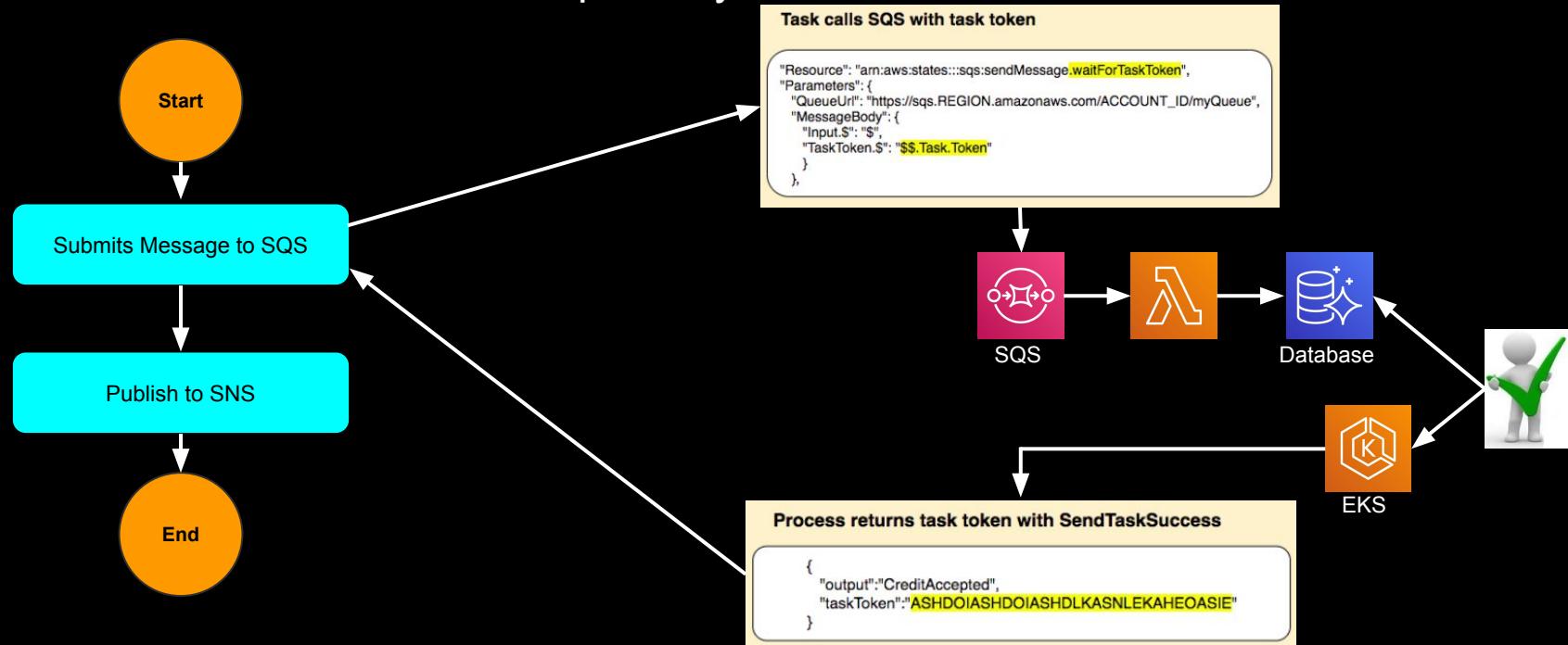
# Wait For Callback (.waitForTaskToken)

- Step Function pass a task token to integrated service
- Workflow paused until task token is returned
  - Can wait upto an year!



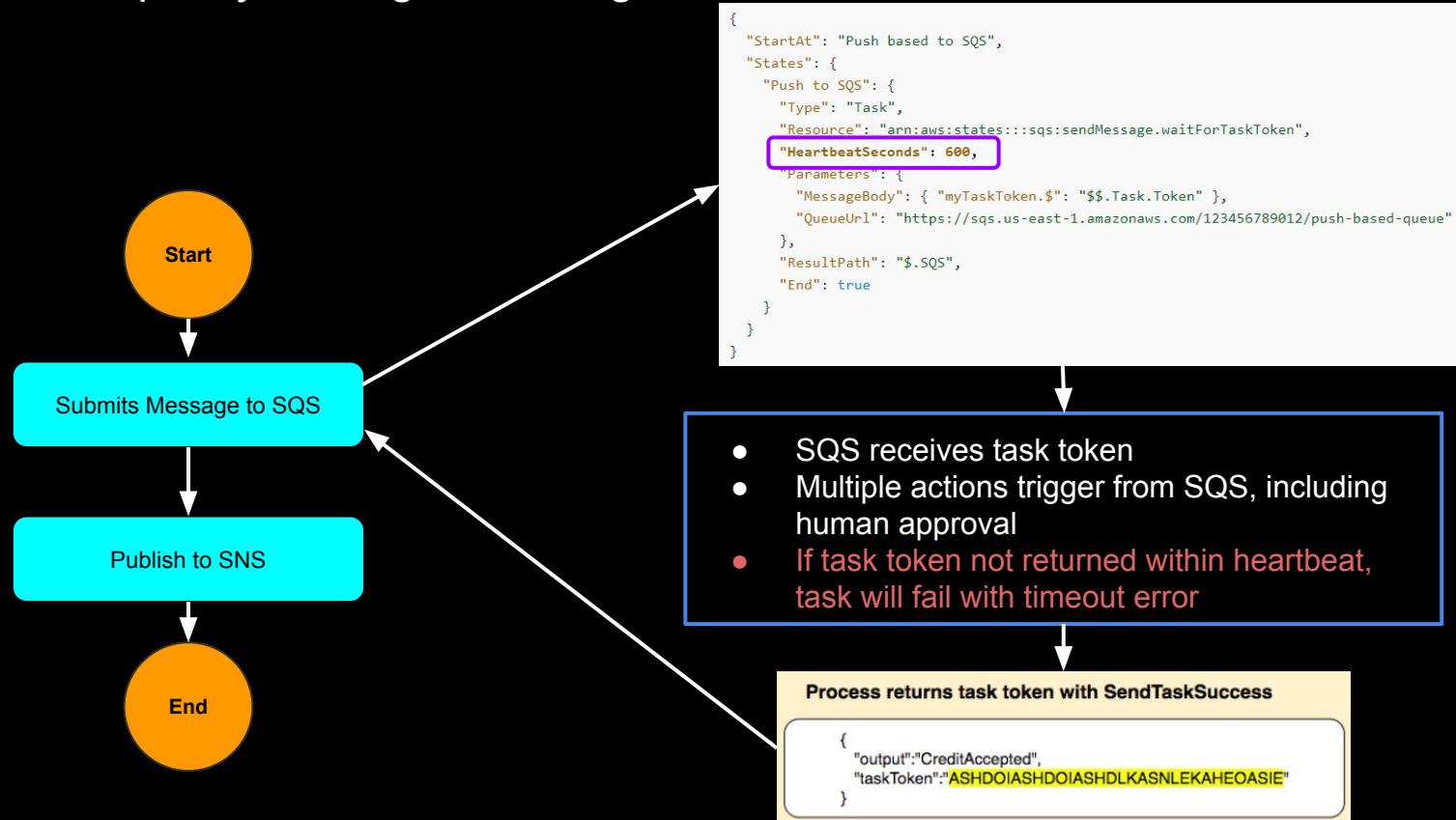
# Wait For Callback (.waitForTaskToken)

- Step Function pass a task token to integrated service
- Workflow paused until task token is returned
  - Can wait upto an year!

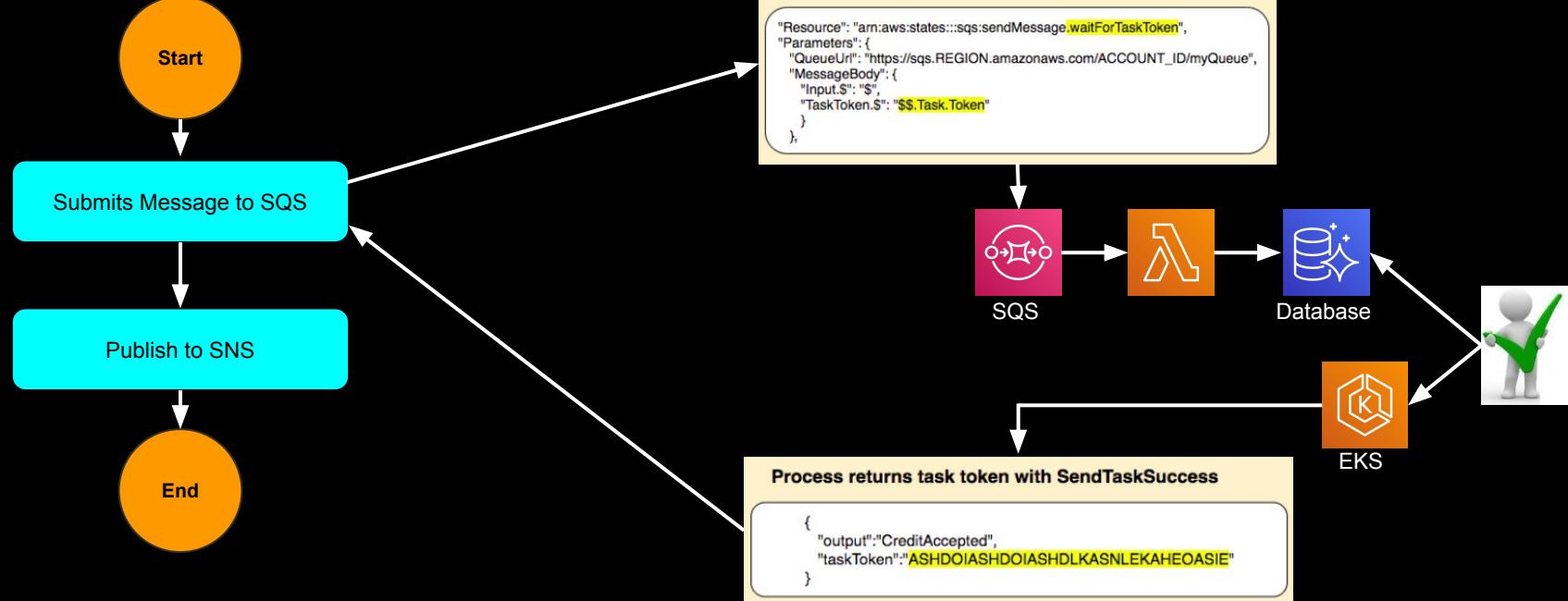


# Wait For Callback (.waitForTaskToken)

- Specify waiting limit using Heartbeat Timeout

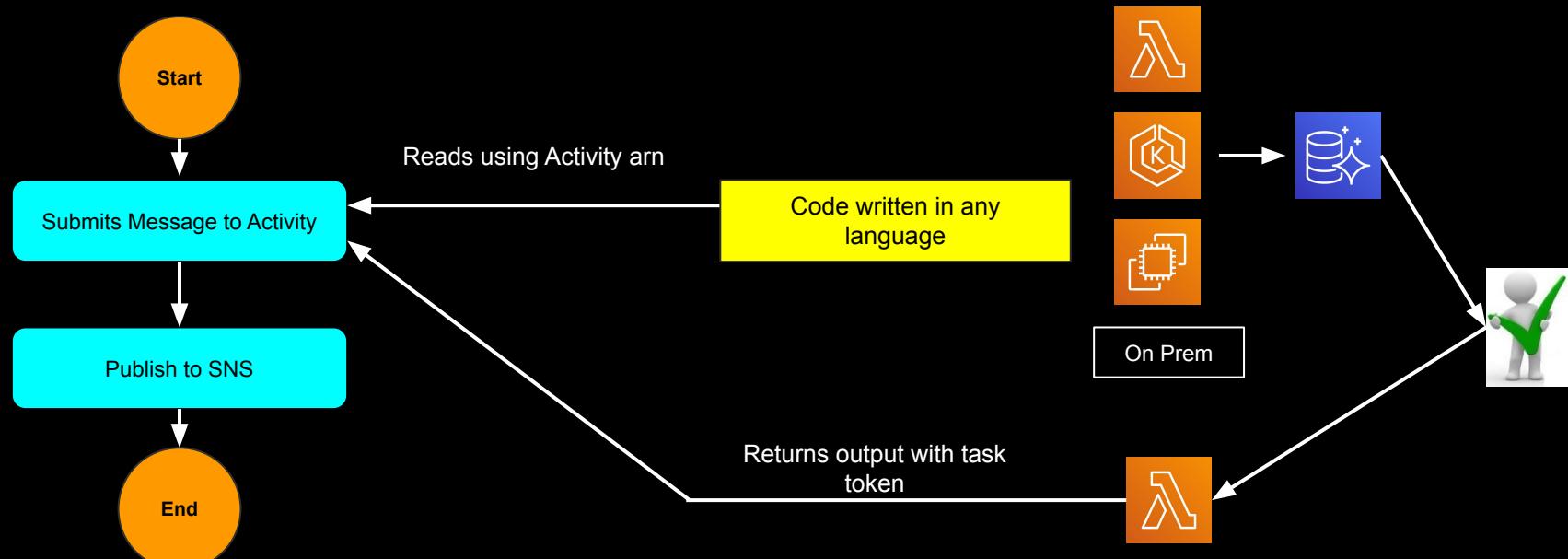


# Activity Workers



# Activity Workers

- Step Function runs activity with input, and task token
- Some other process (NOT invoked by Step Function) reads, processes tasks and returns output with task token



# Lambda Logging & Monitoring



AWS Lambda



Amazon CloudWatch

- Logs
- Metrics
- CloudWatch Insights

# API Gateway & CloudWatch - Metrics

# CloudTrail Logging

- CloudTrail does Infrastructure Logging
  - Example:
    - Creation/Deletion of S3 bucket
    - Creation/Deletion of VPC
    - Creation/Deletion of Security group
- CloudWatch does Application Logging
  - Example:
    - API request/response payload
    - Logging from your lambda code
    - Logs for execution of your API
- CloudTrail logs can be sent to CloudWatch logs
- All the logs can be fed to an analytic system for actionable insights

# API Gateway & CloudWatch Logging - Beyond the Basic Execution Logging

- Logs related to execution of the API
- Includes Logs for
  - Request and Response payloads
  - API Keys
  - Usage Plans
  - Data used by Lambda authorizers (custom authorizers)
- Log group would be created automatically, named  
API-gateway-Execution-Logs\_{rest-api-id}/{stage\_name} format

## Access Logging

- Logs related to access of the API
- Includes Logs for
  - Who accessed your API
  - How the caller accessed the API
- Create a new Log Group or send to existing one
- Logs can be generated in CLF, JSON, XML, CSV formats for ease of consumption by log analysis system/process if applicable

**Both are charged as per standard CloudWatch rates**

# Cloudwatch Log Insights

Announced at re:Invent 2018

## How do you get insights today on your log?

- Amount of logs are massive and harder to query
- Utilize third party tools to give you insight



CloudHealth<sup>®</sup>  
TECHNOLOGIES



IO|pipe



# Cloudwatch Log Insights

Announced at re:Invent 2018

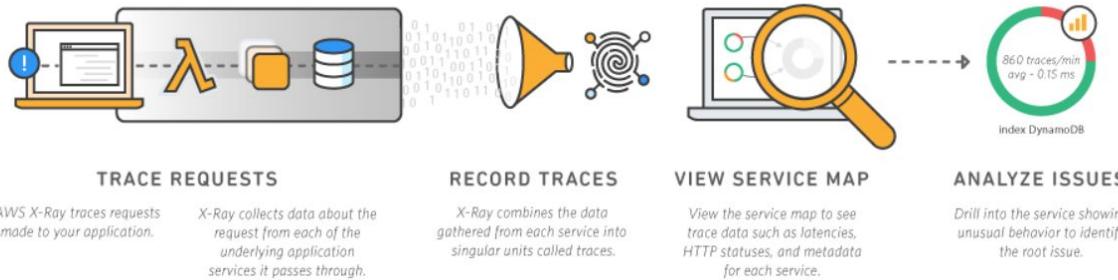
- Fully managed log query tool
  - AWS managed service
  - No setup, maintenance required
- Queries massive amount of logs in seconds
- Produces visualizations
- **Lots of pre built queries**

Let's get hands on

# What is AWS X-Ray

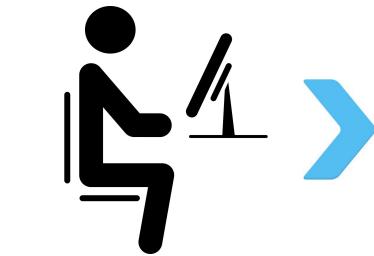
- Distributed Tracing System
- Shows map of underlying components
- Identify root cause of performance issues

## How It Works



Reference: <https://aws.amazon.com/xray/>

# Demo of AWS X-Ray with API Gateway



Invokes API



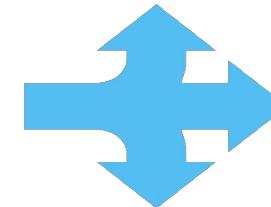
API Gateway calls  
Lambda backend



Trace API Gateway calls



Lambda does all  
the computations  
and send  
response back



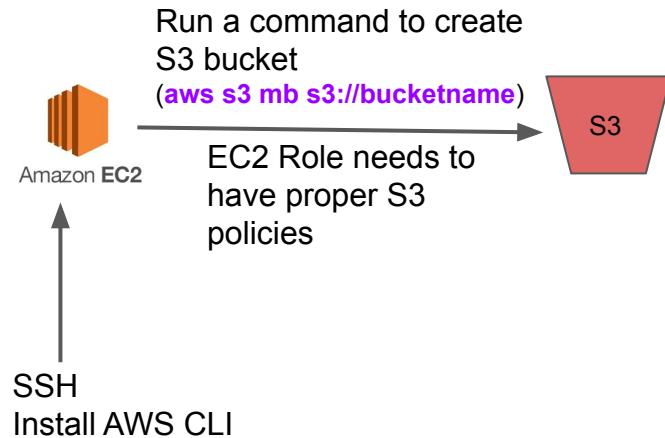
Trace Lambda invocation  
and execution



AWS Step Functions

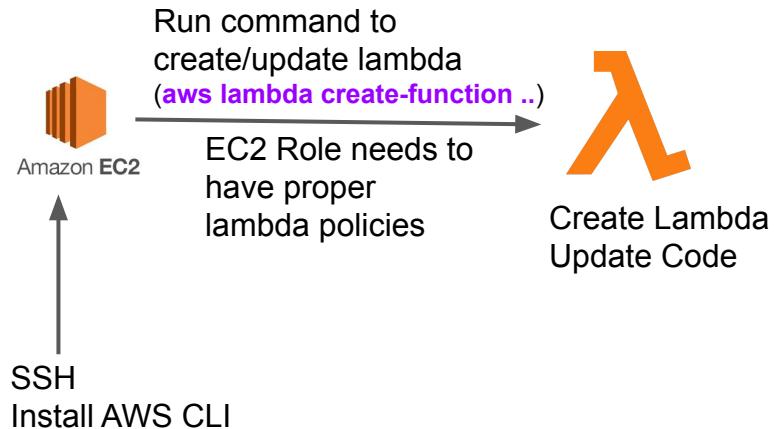
# AWS CLI

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.



# AWS CLI

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.



# AWS Cloud9 - What and Why?

Home Desktop



- Latest IDE and patches
- That super cool color scheme addon
- Most of personal projects



Office Laptop



- Not so latest IDE Version
- Your favorite addons not compatible
- Moved personal projects using drives/disks



That machine is still here?



- What is IDE?!
- Huh, Addons!?
- Where is the USB port?!



# AWS Cloud9 - What and Why?

- Cloud-based IDE that write, run, debug your code with just a browser
- Runs in underlying EC2, granting you massive processing power
- Code, Addons and Customizations are saved in cloud
- Consistent experience, no matter which machine you log in from
- And yes, your favorite color scheme is saved as well!



```
 1  symbols.function = "lambda_handler"
 2
 3  def lambda_handler(event, context):
 4      # print event
 5      print("Received event: " + str(event))
 6
 7      location = event["location"]
 8      # print("Location: " + location)
 9
10      # Create a client object, passing the AWS Region
11      # Note: changing this to 'us-west-2' will enable AWS Lambda functions
12      # located in the us-west-2 region
13      # AWS Lambda uses the 'us-east-1' region by default
14      # client = boto3.client('lambda')
15      client = boto3.client('lambda', region_name='us-east-1')
16
17      # Call the Lambda function
18      response = client.invoke(
19          FunctionName='Geocode',
20          Payload=json.dumps(location))
21
22      # Get the invoke result
23      result = response['Payload'].read()
24
25      # Print the result
26      print(result)
27
28      # Parse the JSON result
29      result = json.loads(result)
30
31      # Extract the address
32      address = result['address']
33
34      # Extract the latitude and longitude
35      latitude = result['latitude']
36      longitude = result['longitude']
37
38      # Print the address
39      print("Latitude, Longitude, Formatted Address")
40      print("%s, %s, %s" % (latitude, longitude, address))
41
42      # Return the address
43      return address
44
45  # Handler for Lambda function
46  def lambda_handler(event, context):
47      # print("Received event: " + str(event))
48
49      # Create a client object, passing the AWS Region
50      # Note: changing this to 'us-west-2' will enable AWS Lambda functions
51      # located in the us-west-2 region
52      # AWS Lambda uses the 'us-east-1' region by default
53      # client = boto3.client('lambda')
54      client = boto3.client('lambda', region_name='us-east-1')
55
56      # Call the Lambda function
57      response = client.invoke(
58          FunctionName='Geocode',
59          Payload=json.dumps(event))
60
61      # Get the invoke result
62      result = response['Payload'].read()
63
64      # Print the result
65      print(result)
66
67      # Parse the JSON result
68      result = json.loads(result)
69
70      # Extract the address
71      address = result['address']
72
73      # Extract the latitude and longitude
74      latitude = result['latitude']
75      longitude = result['longitude']
76
77      # Print the address
78      print("Latitude, Longitude, Formatted Address")
79      print("%s, %s, %s" % (latitude, longitude, address))
80
81      # Return the address
82      return address
```



# AWS Cloud9 - Features

- Code together with friends
- Seamless integration with Lambda - write, run and DEBUG Lambdas!
- Direct Terminal access utilizing AWS CLI
- Supports languages beyond Lambda
- Tons of color schemes!

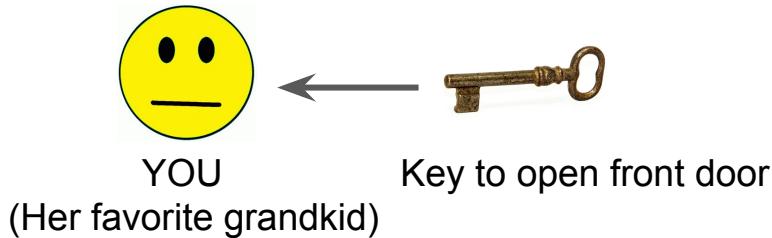
Demo for all the above features shown in the course!

# Controlling Access to API and Lambda

- API Keys and Usage Plans
- AWS Cognito User Pools
- AWS Secrets Manager
- Lambda Resource Policies
- Lambda Authorizer (Custom Authorizer)
- API Gateway Resource Policies

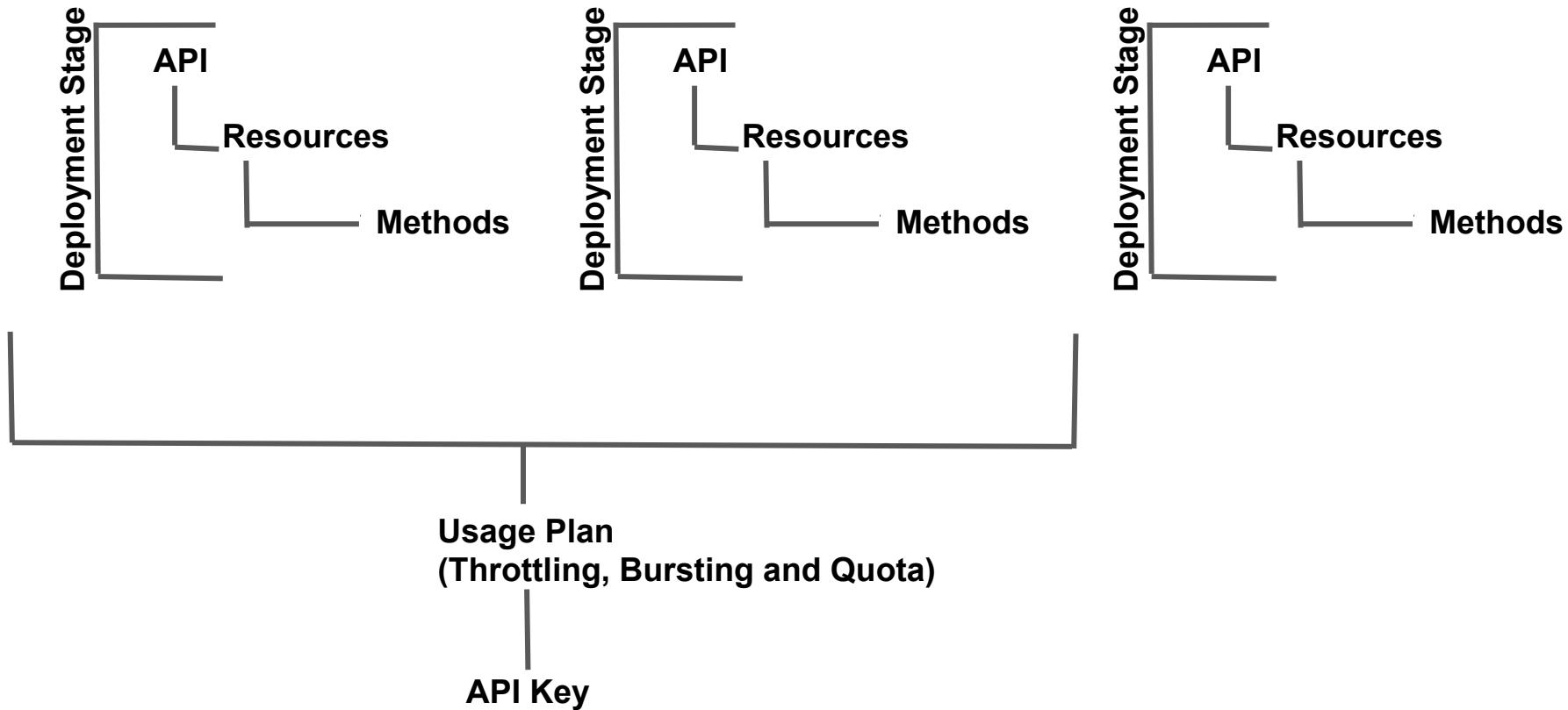


# Security - Using API Key



**Learning: Whoever has the key, has the access**

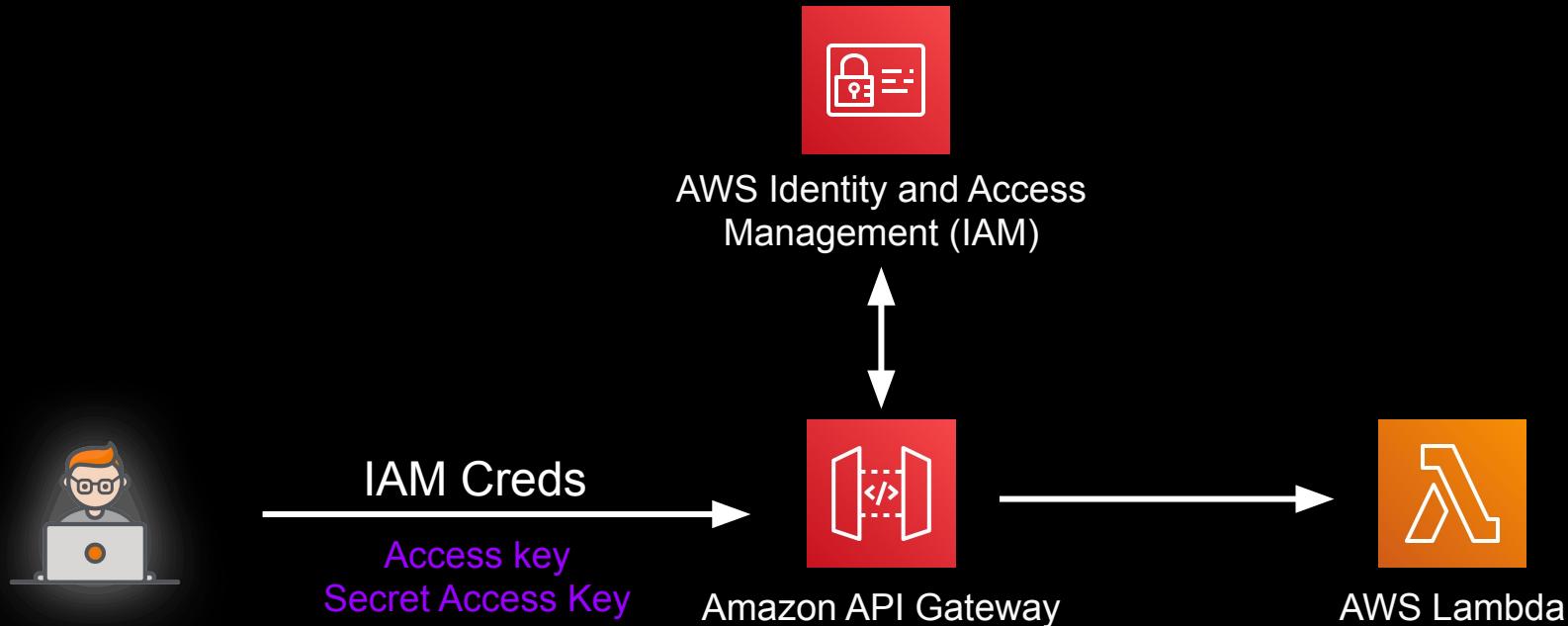
# Usage Plan and API Key



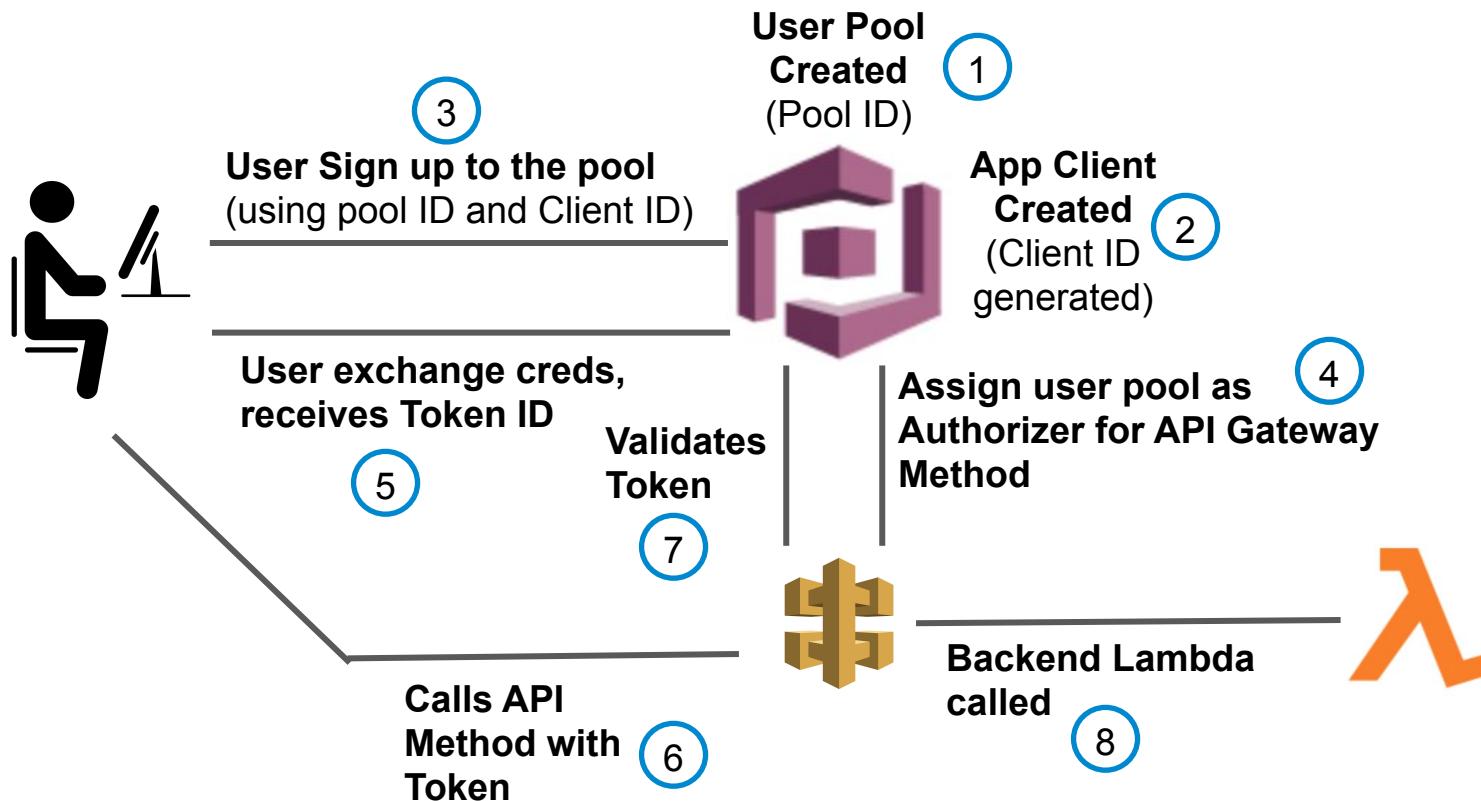
# Controlling Access

- DEMO of API Key and Usage Plans

# Securing API With IAM



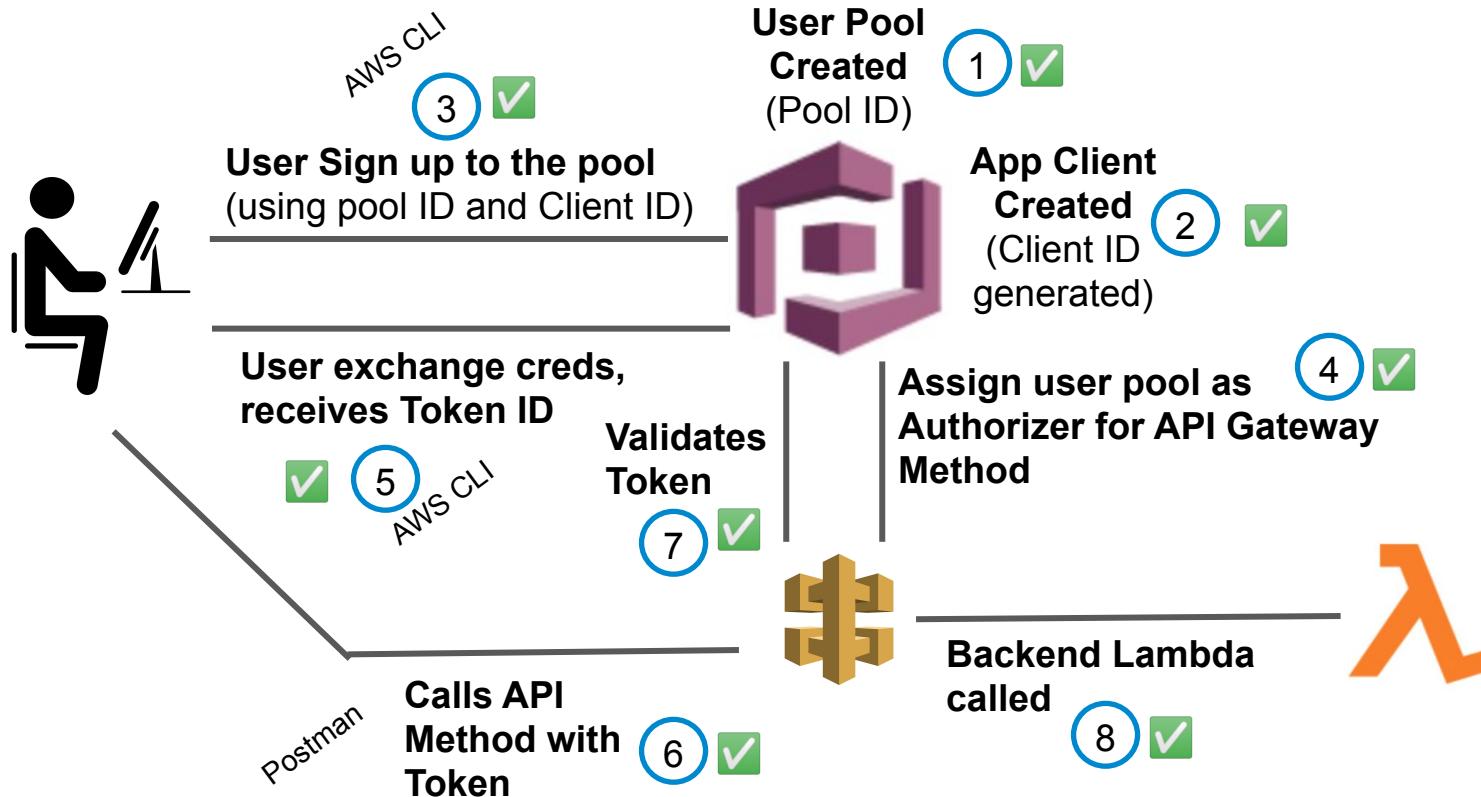
# Cognito User Pool and API Gateway Flow



# API Key Vs Cognito

API Key	Cognito
<p><b>Pros</b></p> <ul style="list-style-type: none"><li>• Easy to implement</li><li>• Most popular way of API security</li><li>• Less moving parts</li></ul> <p><b>Cons</b></p> <ul style="list-style-type: none"><li>• If keys are compromised, API can be invoked by anyone</li><li>• By default, keys are non expiring. Custom process required to rotate keys periodically</li></ul>	<p><b>Pros</b></p> <ul style="list-style-type: none"><li>• More secure than API key because:<ul style="list-style-type: none"><li>◦ Tokens automatically rotate periodically</li><li>◦ Dual checkpoint - during obtaining token, then again during token validation by API gateway</li></ul></li><li>• Similar to OAUTH2, which is becoming the new standard</li></ul> <p><b>Cons</b></p> <ul style="list-style-type: none"><li>• Overhead of additional Authentication server setup</li><li>• Could introduce delay to the overall response time</li></ul> <p style="color: green; font-size: 2em; position: absolute; top: 0; right: 0;">RECOMMENDED</p>

# Demo - Cognito User Pool & API Gateway



# Cognito Federated Identities

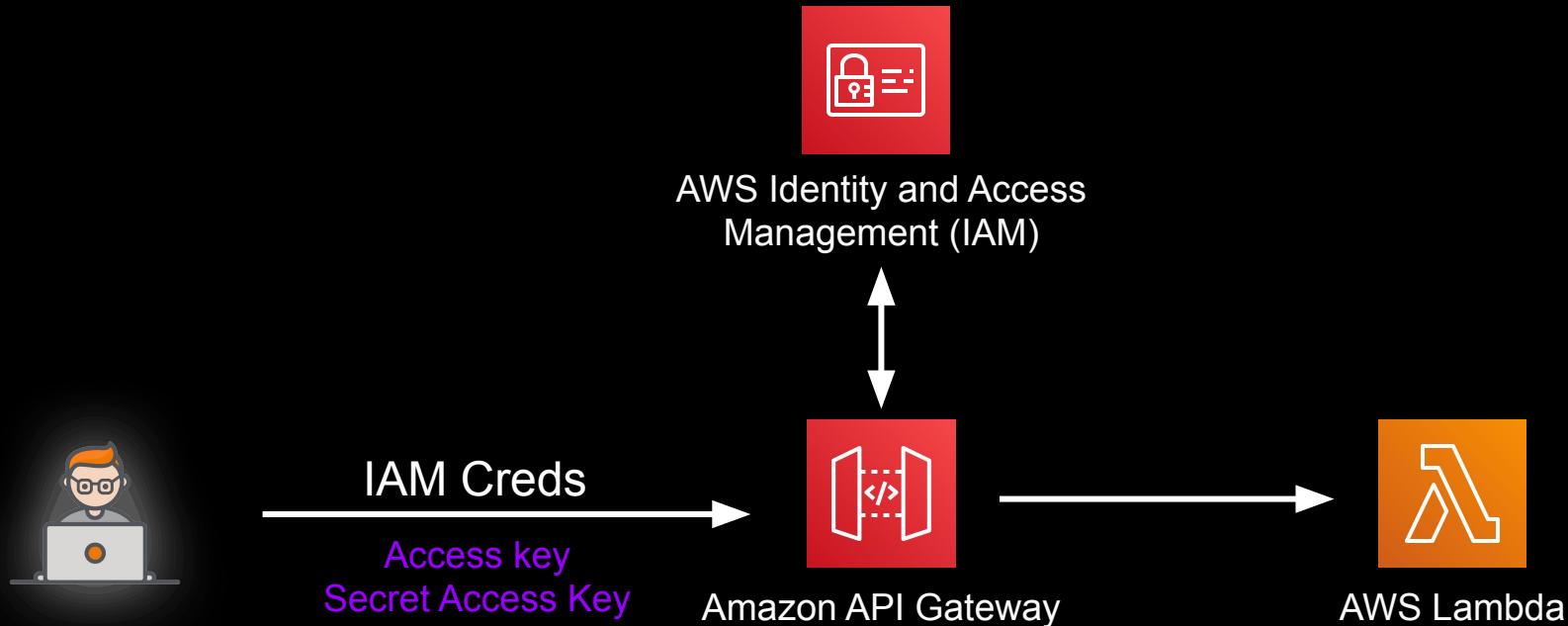
VS

Cognito User Pool

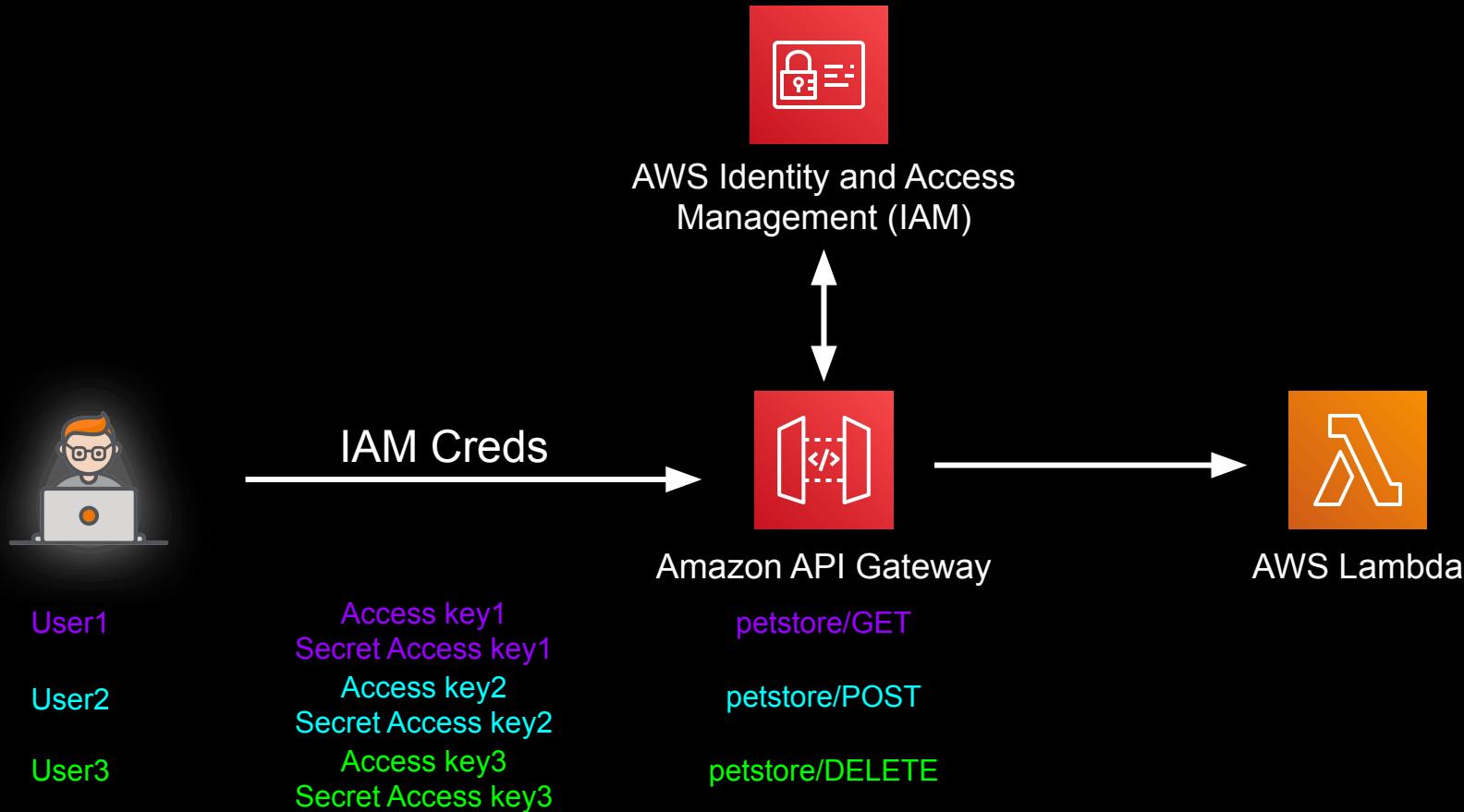
VS

AWS\_IAM

# Securing API With IAM



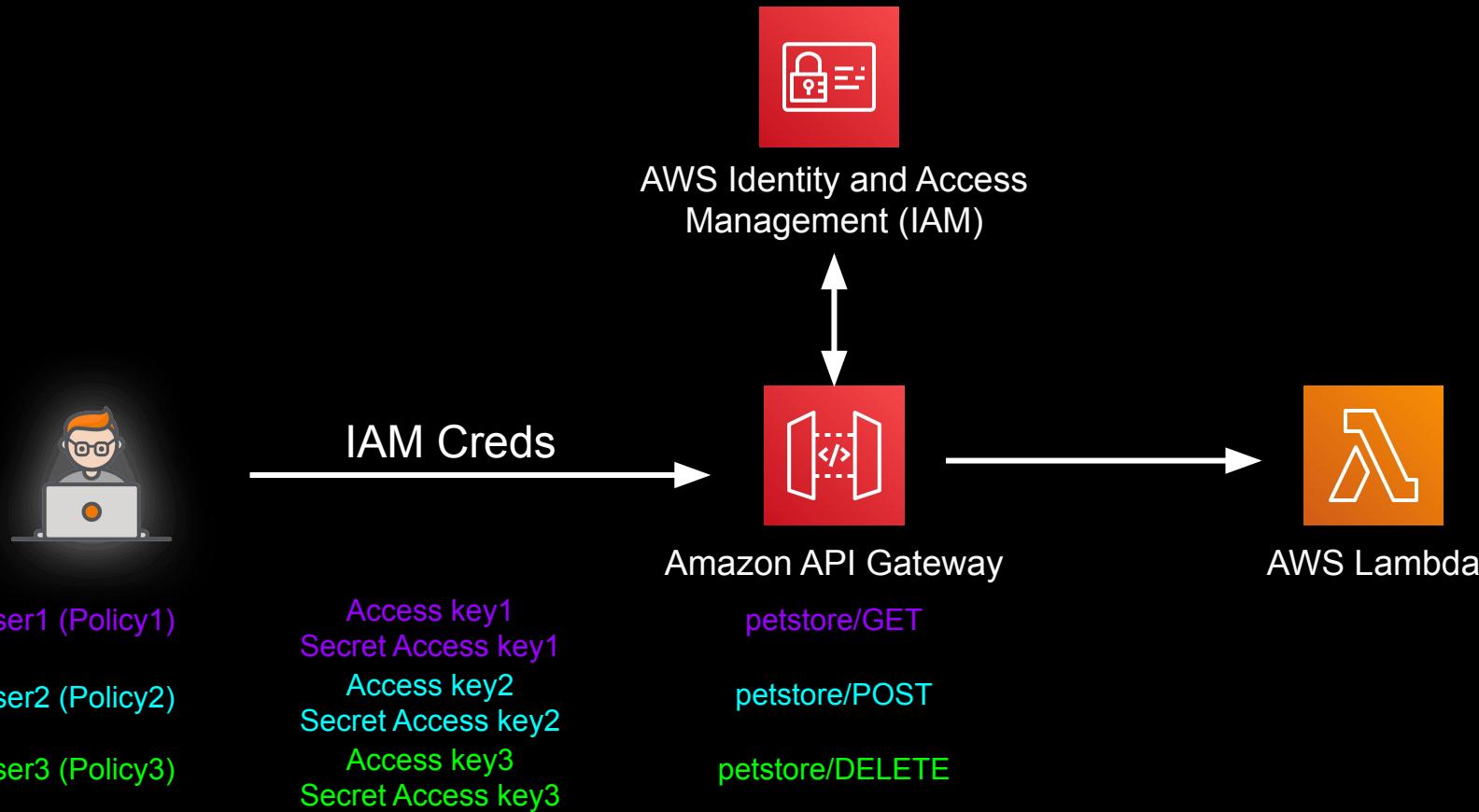
# Securing API With IAM



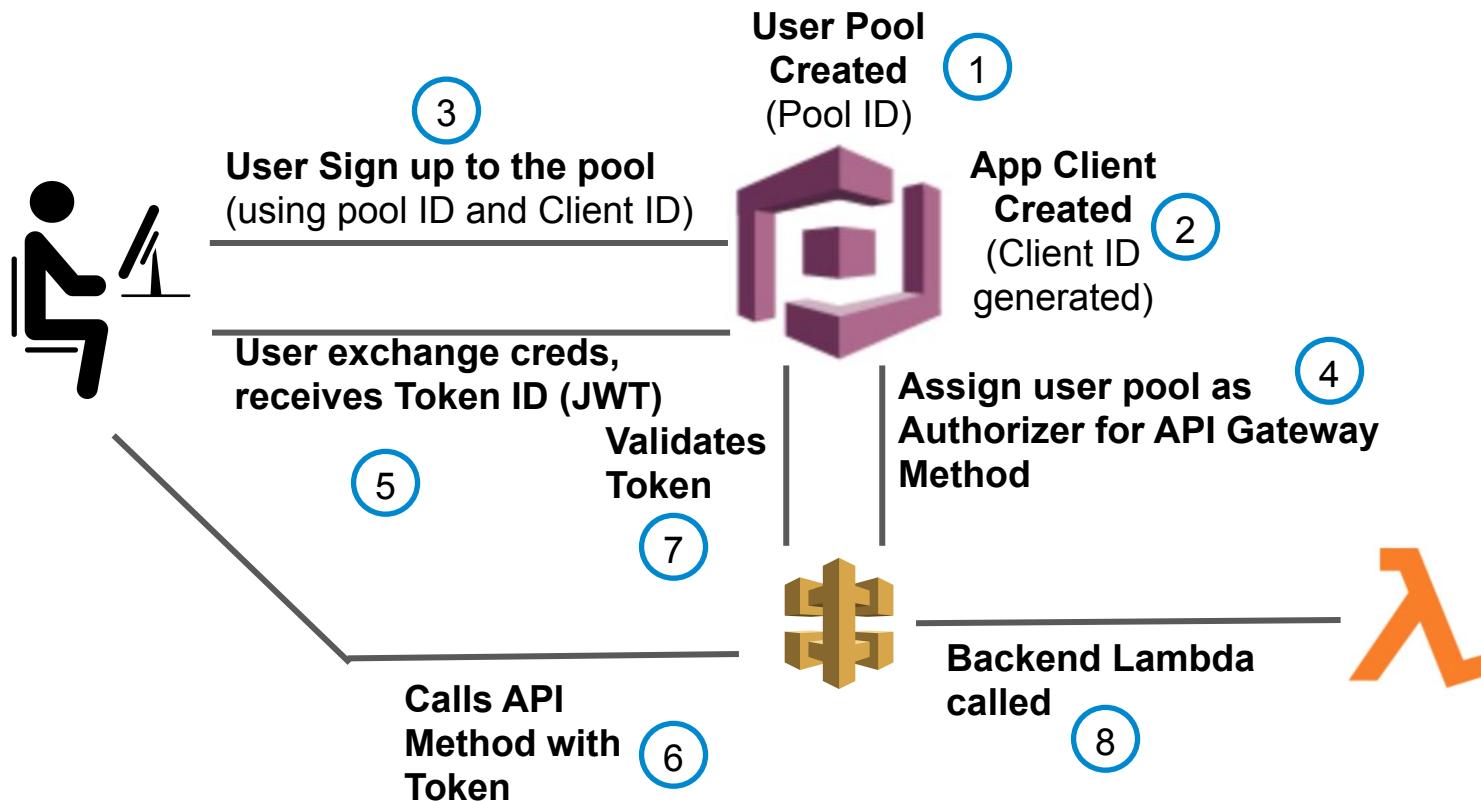
# How Does IAM Policy Look?

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": "execute-api:Invoke",  
            "Effect": "Allow",  
            "Resource": "arn:aws:execute-api:us-east-1:*:a123456789/prod/POST/petstore/*"  
        }  
    ]  
}
```

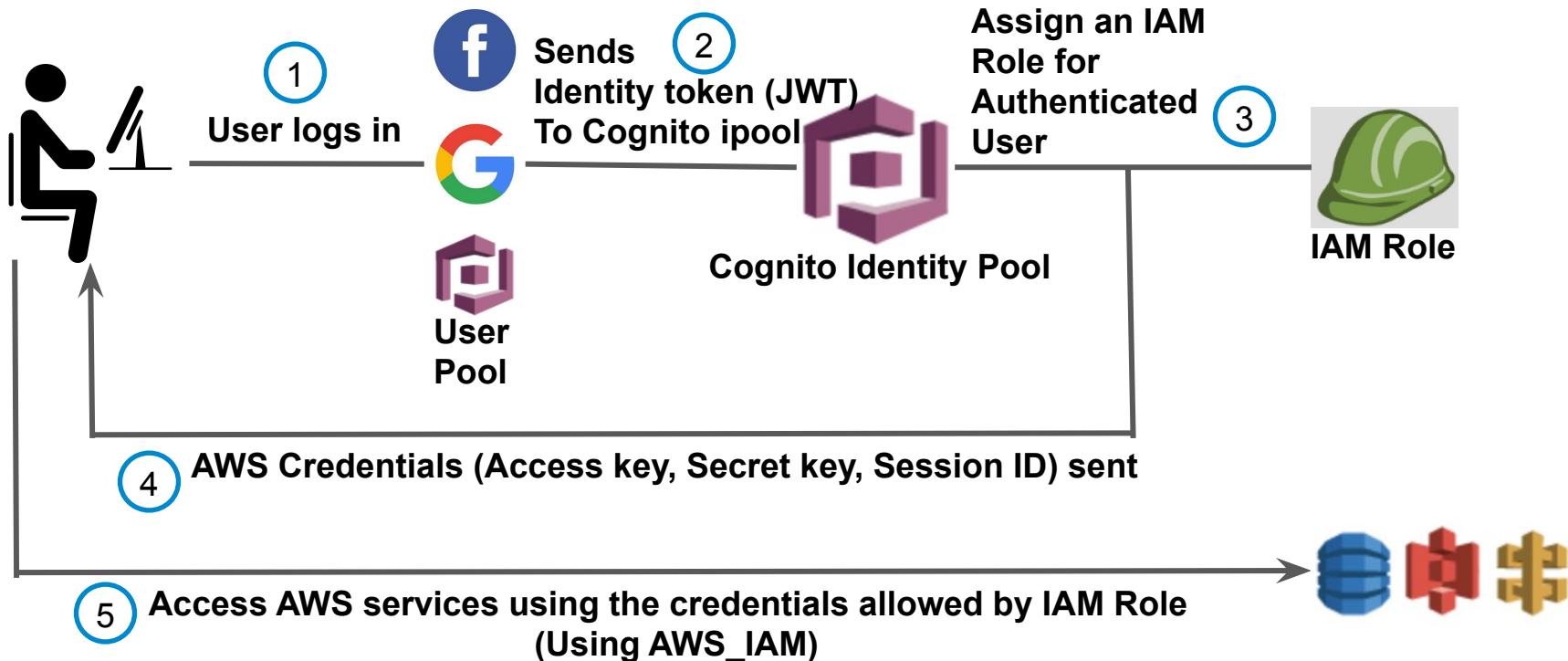
# Securing API With IAM



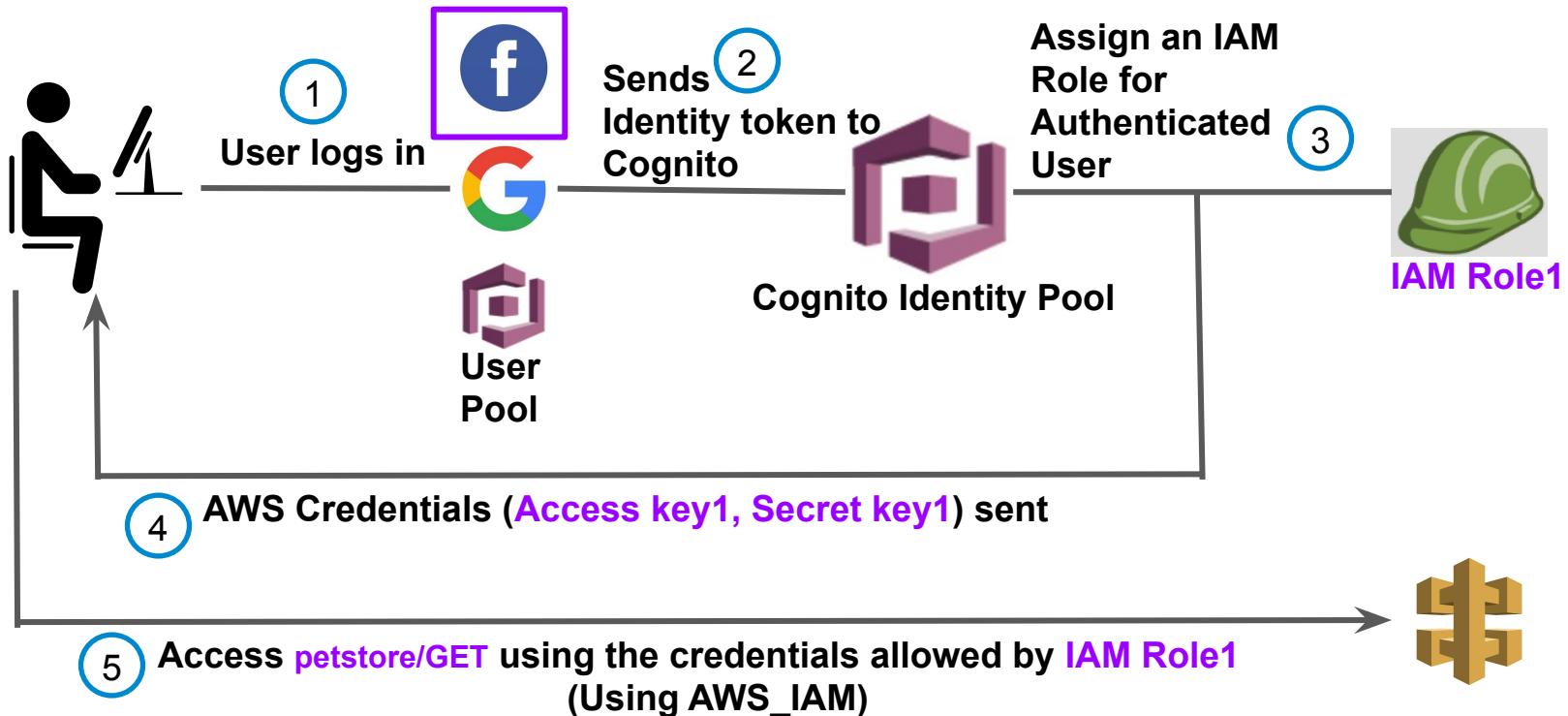
# Cognito User Pool and API Gateway Flow



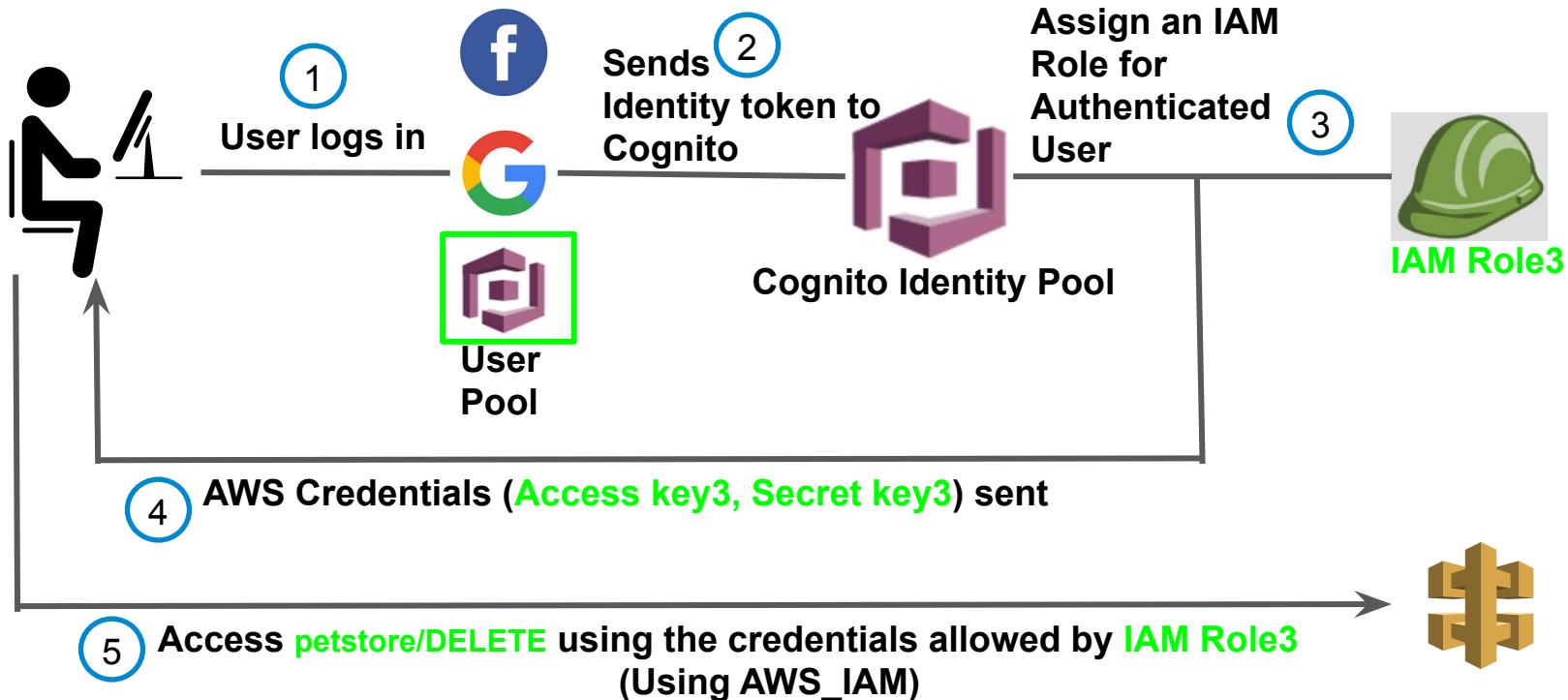
# Understanding Cognito Identity Pool



# Understanding Cognito Identity Pool



# Understanding Cognito Identity Pool



# AWS Secrets Manager

## Hardcoding Creds in Code

```
def findNews():
    #News credit to newsapi.org
    #Fetch headlines using the API
    #IMPORTANT: Register in newsapi.org to get your own API key, it's super easy!
    response = requests.get("https://newsapi.org/v2/top-headlines?country=us&category=business&apiKey=84e3ef10a0b44603a7c64de19aca7848")
    d=response.json()
```

## Importing Creds

```
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
```

Vulnerable to code download and code repository  
read

# AWS Secrets Manager

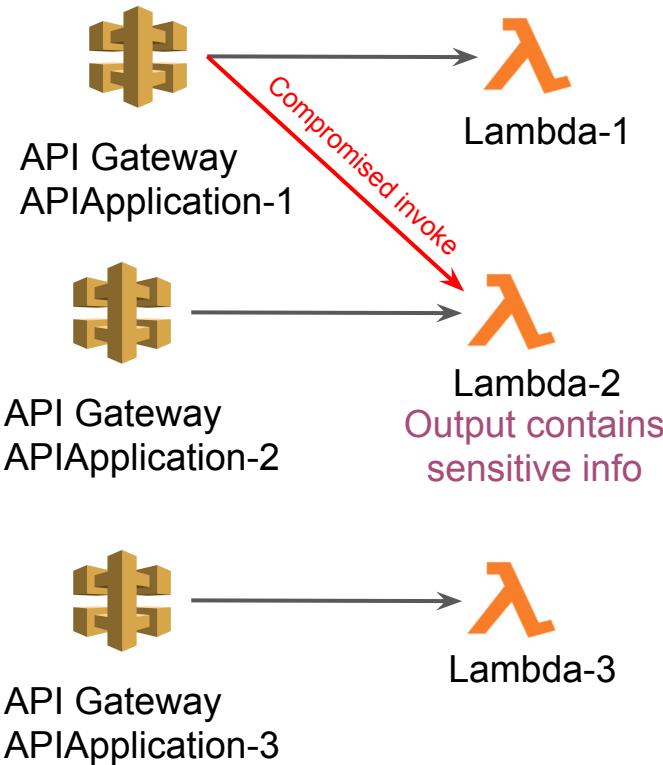


- Stores API Key, DB Credentials, encrypted with KMS key
- Can rotate the credentials periodically

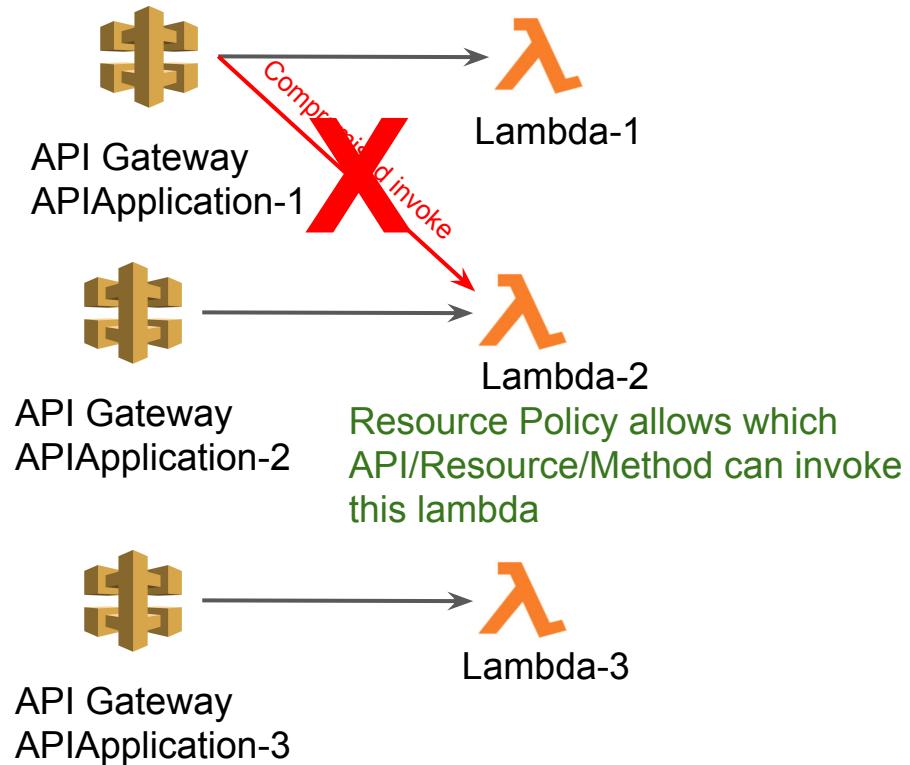
## Code snippet with Secrets Manager

```
apikeyinjson=json.loads(get_secret())
newsapikey=apikeyinjson['APIKey']
urlwithkey="https://newsapi.org/v2/top-headlines?country=us&category=business&apiKey=' + newsapikey
response = requests.get(urlwithkey)
d=response.json()
```

# Security using Lambda Resource Policy



## With Resource Based Policy (Function Policy)



# Lambda Resource Policy Continued

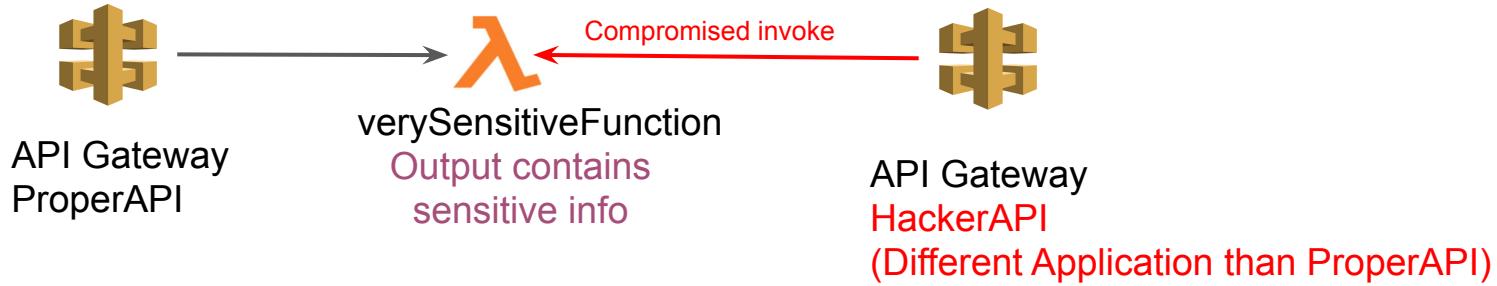
## Default Behaviour

- By default, if you invoke lambda from API in same AWS account, resource policy of Lambda is automatically updated to allow the invocation
- For real-world projects, console access beyond development should be prohibited and should be deployed through CI/CD toolchain with resource policy defined in Cloudformation
  - CI/CD ensures userid (who is deploying) belongs to application ID
- By default, if you invoke lambda from API Gateway from different AWS account, resource policy of Lambda needs to be updated explicitly
- At this point (Sept, 2018), Lambda resource policy can't be updated through console
  - Can be updated from AWS CLI and Cloudformation

Demo

Lambda Resource Policy

# Demo - Lambda Resource Policy



## AWS CLI Commands

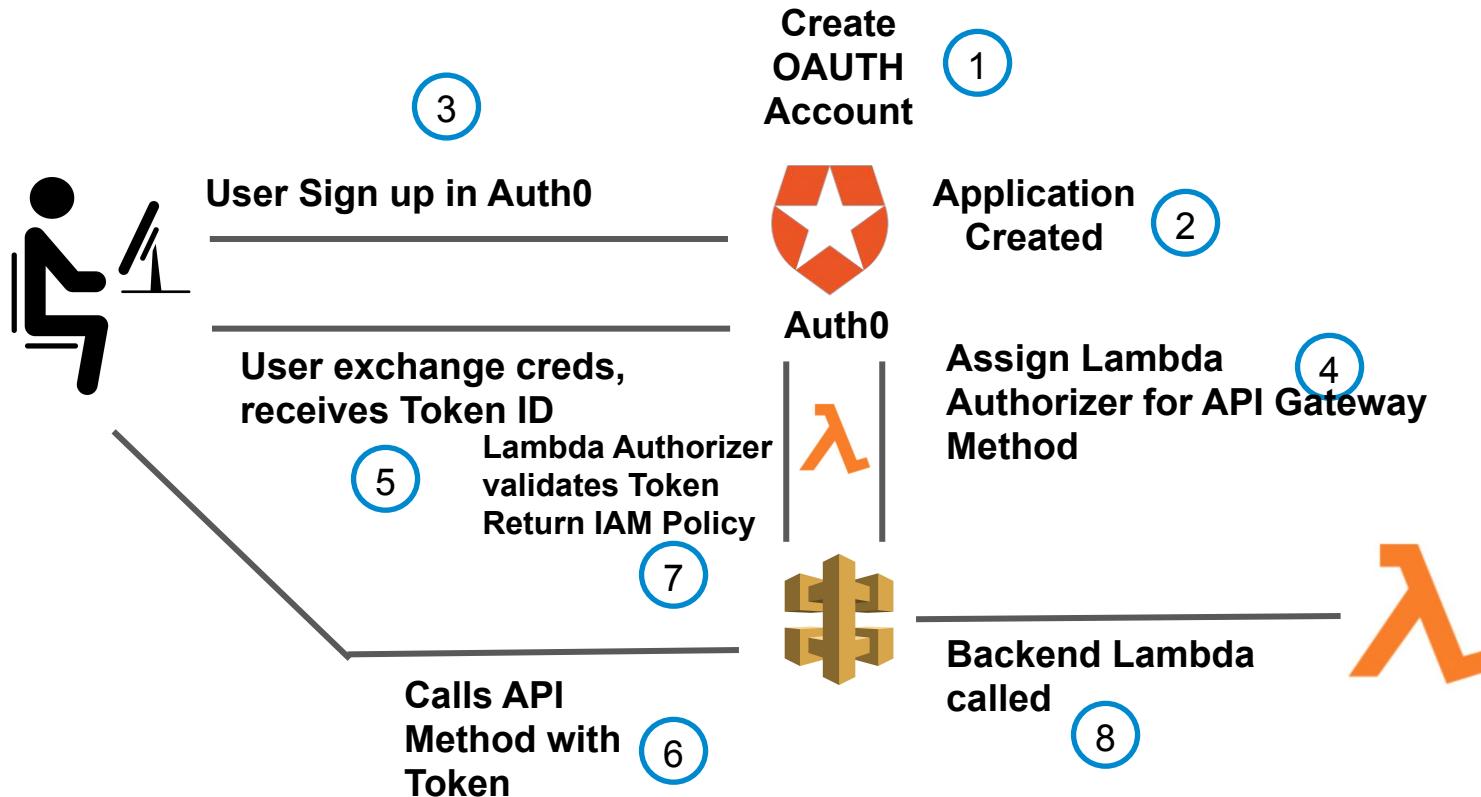
```
aws lambda get-policy  
--function-name <lambda name or arn>
```

---

```
aws lambda add-permission  
--function-name <lambda name or arn>  
--source-arn <arn of the API Gateway Method>  
--principal apigateway.amazonaws.com  
--statement-id <string>  
--action lambda:InvokeFunction
```

```
aws lambda remove-permission  
--function-name <lambda name or arn>  
--statement-id <statement id of the permission>
```

# Lambda Authorizer and API Gateway Flow

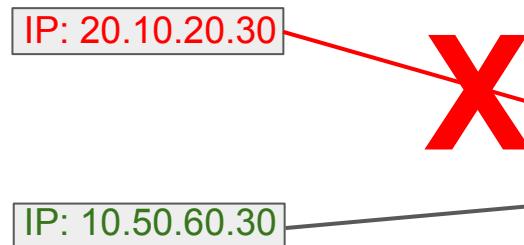


# When To Use What?

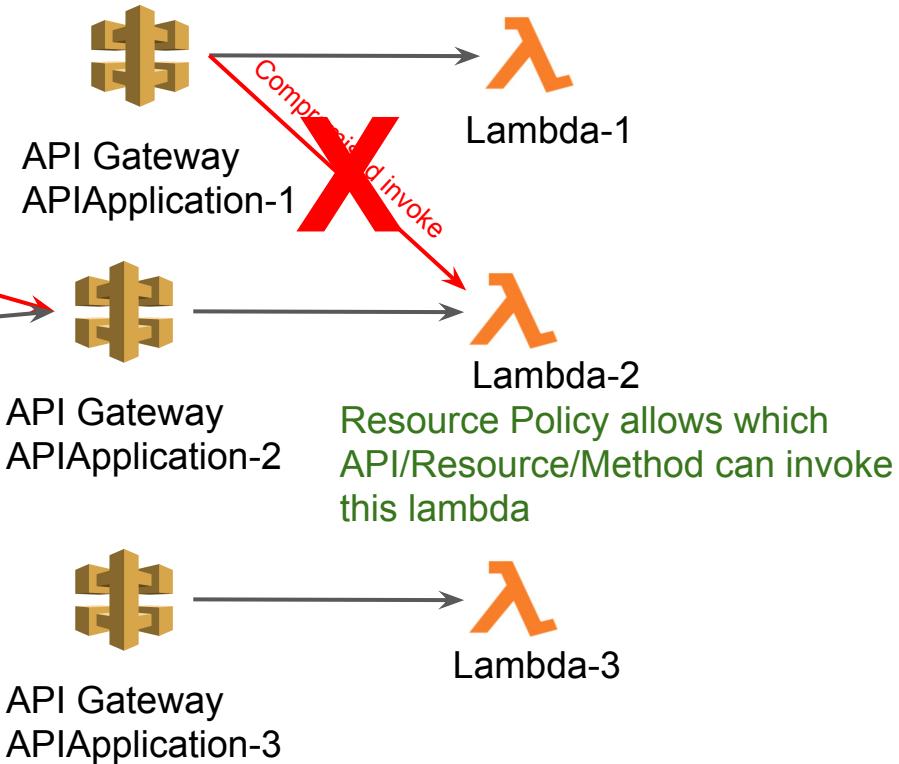
Mechanism	Use Case
API Key	Test app, Internal APIs, some times Prod APIs with key rotation
AWS_IAM	Almost never by itself
Cognito User Pool	When using AWS Identity Provider (Cognito)
Cognito Identity Pool	Reuse credentials from specific idp (Facebook, Google, Amazon), and/or IAM granularity needed
Lambda Authorizer	When using external Idp (Okta, AD, Auth0 etc.)

# Security using API Gateway Resource Policy

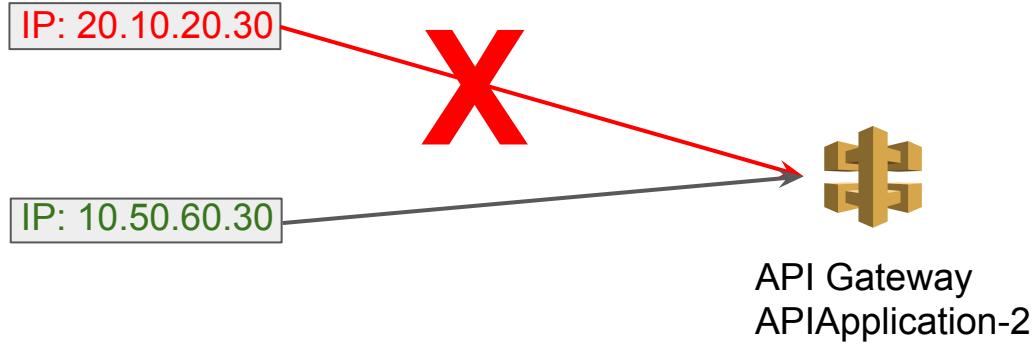
## With API gateway Resource Policy



## With Resource Based Policy (Function Policy)



# Security using API Gateway Resource Policy



```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "execute-api:Invoke",  
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": ["10.50.60.30"]  
                }}}]}{
```

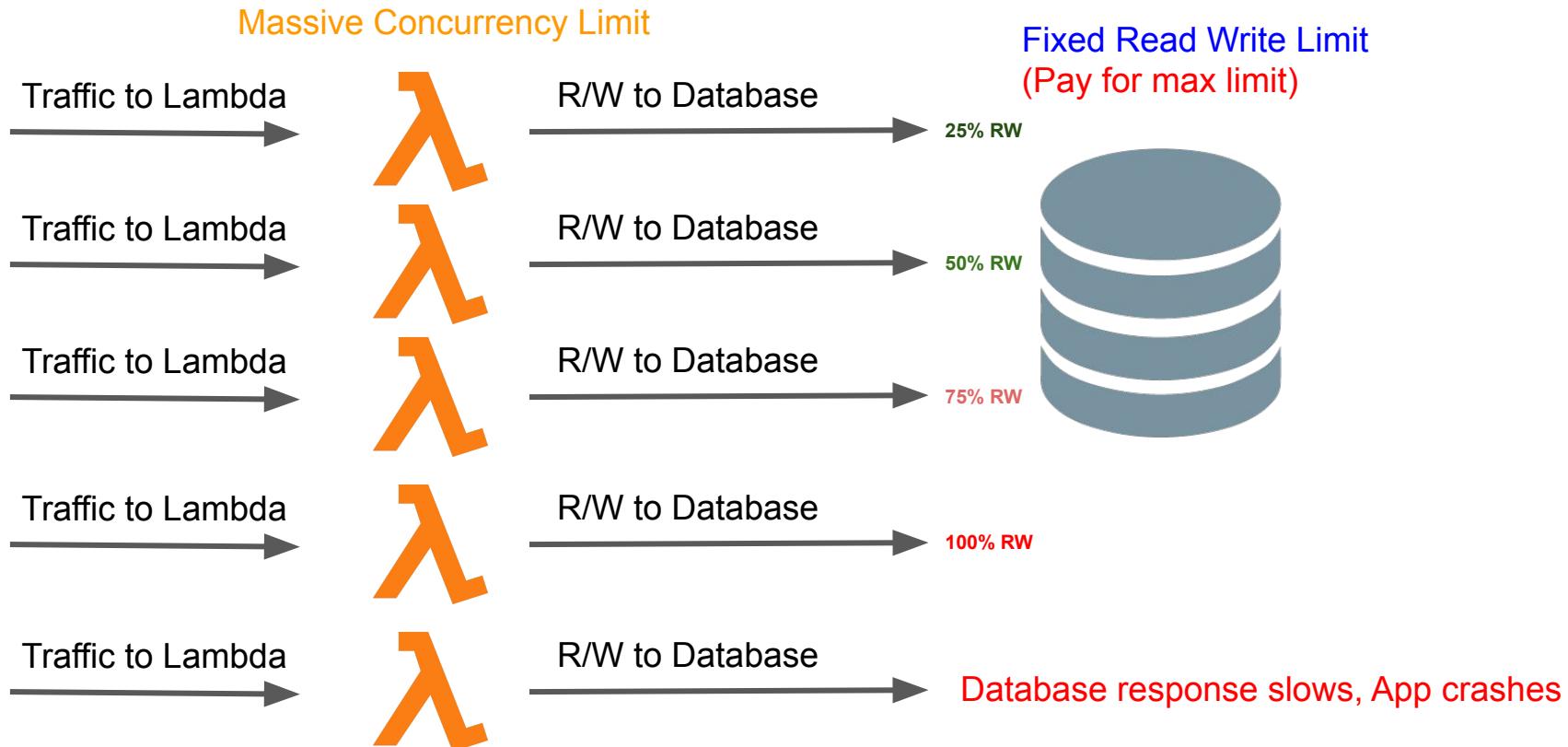
# Storage For Serverless

# SQL Vs NoSQL Database

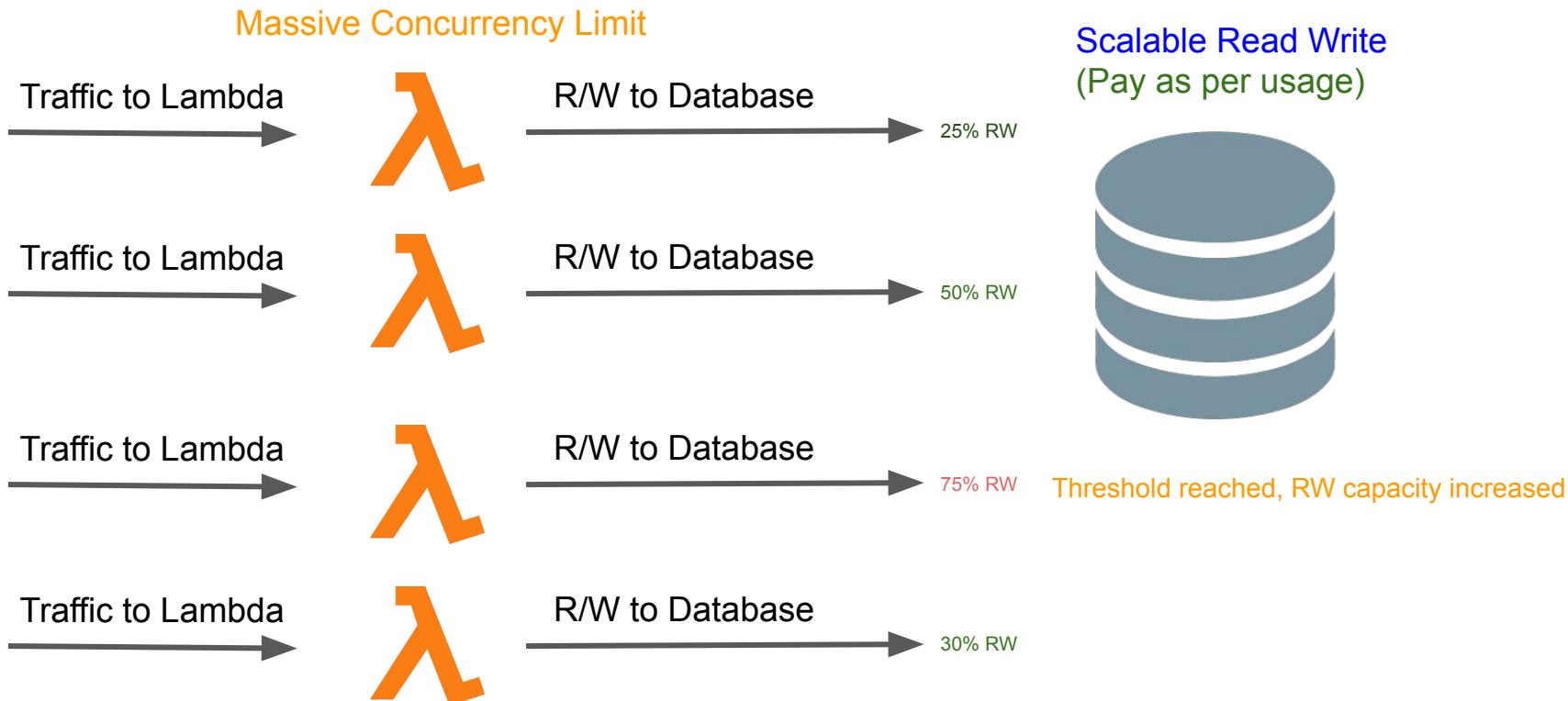
SQL Database (RDBMS)	NoSQL Database
Tables have predefined schema	Schemaless
Holds structured data	Holds structured and unstructured data
Good fit for joins and complex queries	Generally, not good fit for complex multi table queries
Emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability)	Follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance )
Example - Oracle, DB2, MS-SQL, AWS RDS	Example - AWS DynamoDB, MongoDB, Cassandra

**Important:** With the advent of technology, segregation of use cases for SQL and NoSQL are NOT as black and white as it used to be, multiple factors need to be considered

# Issue with Lambda and Traditional Database



# Ideal Lambda and Database Behaviour



# Auto Scalable Storage Options for Lambda

## NoSQL - DynamoDB

Auto Scaling

<input checked="" type="checkbox"/> Read capacity	<input checked="" type="checkbox"/> Write capacity
<input type="checkbox"/> Same settings as read	
Target utilization 70 %	70 %
Minimum provisioned capacity 5 units	5 units
Maximum provisioned capacity 40000 units	40000 units
<input checked="" type="checkbox"/> Apply same settings to global secondary indexes	
<input checked="" type="checkbox"/> Apply same settings to global secondary indexes	

## RDBMS - Aurora Serverless (As of Aug 2018!)

Minimum Aurora capacity unit <a href="#">Info</a>	Maximum Aurora capacity unit <a href="#">Info</a>	
2 4Gb RAM	256 488Gb RAM	
▼ Additional scaling configuration		
<input checked="" type="checkbox"/> Pause compute capacity after consecutive minutes of inactivity <a href="#">Info</a>		
You are only charged for database storage while the compute capacity is paused		
00 hours	05 minutes	00 seconds
Max: 24 hours		

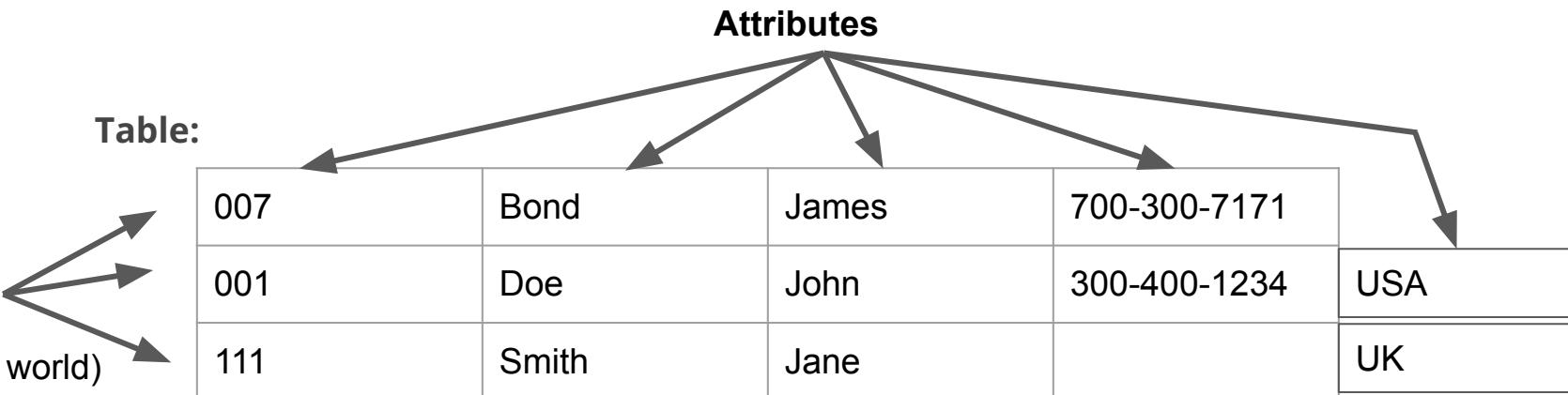
# What is DynamoDB?

- Fully managed NoSQL database
  - Hardware provisioning, setup and configuration, replication, software patching, or cluster scaling managed by AWS
- Store and retrieve any amount of data
- Serve any level of request traffic
- **Autoscaling**
- Highly available and durable
- Multi-region, multi-master database using Global tables
- Encryption at rest

For further details, refer to [DynamoDB Developer Guide](#)  
<https://aws.amazon.com/documentation/dynamodb/>

# DynamoDB Core Components

- **Tables** - A *table* is a collection of data. Same as every other database.
- **Items** - An *item* is a group of attributes that is uniquely identifiable among all of the other items.
- **Attributes** - Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken down any further.



# DynamoDB Items and Attributes

People

```
{  
    "PersonID": 101,  
    "LastName": "Smith",  
    "FirstName": "Fred",  
    "Phone": "555-4321"  
}  
  
{  
    "PersonID": 102,  
    "LastName": "Jones",  
    "FirstName": "Mary",  
    "Address": {  
        "Street": "123 Main",  
        "City": "Anytown",  
        "State": "OH",  
        "ZIPCode": 12345  
    }  
}  
  
{  
    "PersonID": 103,  
    "LastName": "Stephens",  
    "FirstName": "Howard",  
    "Address": {  
        "Street": "123 Main",  
        "City": "London",  
        "PostalCode": "ER3 5K8"  
    },  
    "FavoriteColor": "Blue"  
}
```

Music

```
{  
    "Artist": "No One You Know",  
    "SongTitle": "My Dog Spot",  
    "AlbumTitle": "Hey Now",  
    "Price": 1.98,  
    "Genre": "Country",  
    "CriticRating": 8.4  
}  
  
{  
    "Artist": "No One You Know",  
    "SongTitle": "Somewhere Down The Road",  
    "AlbumTitle": "Somewhat Famous",  
    "Genre": "Country",  
    "CriticRating": 8.4,  
    "Year": 1984  
}  
  
{  
    "Artist": "The Acme Band",  
    "SongTitle": "Still in Love",  
    "AlbumTitle": "The Buck Starts Here",  
    "Price": 2.47,  
    "Genre": "Rock",  
    "PromotionInfo": {  
        "RadioStationsPlaying": [  
            "KHCR",  
            "KQBX",  
            "WTNR",  
            "WJH"  
        ],  
        "TourDates": {  
            "Seattle": "20150625",  
            "Cleveland": "20150630"  
        },  
        "Rotation": "Heavy"  
    }  
}  
  
{  
    "Artist": "The Acme Band",  
    "SongTitle": "Look Out, World",  
    "AlbumTitle": "The Buck Starts Here",  
    "Price": 0.99,  
    "Genre": "Rock"  
}
```

# DynamoDB Primary Key

**What is Primary Key:** The primary key uniquely identifies each item in the table, so that no two items can have the same key.

- **Partition key** - A simple primary key, composed of one attribute known as the *partition key*.
- **Partition key and sort key** - Referred to as a *composite primary key*, this type of key is composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*. No two items can have same combo of these two attributes.

Partition key is also referred as *hash attribute*, sort key also referred as *range attribute*

# DynamoDB Items and Attributes

People

{ "PersonID": 101, "LastName": "Smith", "FirstName": "Fred", "Phone": "555-4321" }
{ "PersonID": 102, "LastName": "Jones", "FirstName": "Mary", "Address": { "Street": "123 Main", "City": "Anytown", "State": "OH", "ZIPCode": 12345 } }
{ "PersonID": 103, "LastName": "Stephens", "FirstName": "Howard", "Address": { "Street": "123 Main", "City": "London", "PostalCode": "ER3 5K8" }, "FavoriteColor": "Blue" }

- The primary key consists of one attribute (*PersonID*).
- Other than the primary key, the *People* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- Some of the items have a nested attribute (*Address*).  
DynamoDB supports nested attributes up to 32 levels deep.

# DynamoDB Items and Attributes

Music

```
{  
    "Artist": "No One You Know",  
    "SongTitle": "My Dog Spot",  
    "AlbumTitle": "Hey Now",  
    "Price": 1.98,  
    "Genre": "Country",  
    "CriticRating": 8.4  
}  
  
{  
    "Artist": "No One You Know",  
    "SongTitle": "Somewhere Down The Road",  
    "AlbumTitle": "Somewhat Famous",  
    "Genre": "Country",  
    "CriticRating": 8.4,  
    "Year": 1984  
}  
  
{  
    "Artist": "The Acme Band",  
    "SongTitle": "Still in Love",  
    "AlbumTitle": "The Buck Starts Here",  
    "Price": 2.47,  
    "Genre": "Rock",  
    "PromotionInfo": {  
        "RadioStationsPlaying": [  
            "KHCR",  
            "KQBX",  
            "WTNR",  
            "WJJH"  
        ],  
        "TourDates": {  
            "Seattle": "20150625",  
            "Cleveland": "20150630"  
        },  
        "Rotation": "Heavy"  
    }  
}  
  
{  
    "Artist": "The Acme Band",  
    "SongTitle": "Look Out, World",  
    "AlbumTitle": "The Buck Starts Here",  
    "Price": 0.99,  
    "Genre": "Rock"  
}
```

- The primary key for *Music* consists of two attributes (*Artist* and *SongTitle*). Each item in the table must have these two attributes. The combination of *Artist* and *SongTitle* distinguishes each item in the table from all of the others.
- Other than the primary key, the *Music* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- One of the items has a nested attribute (*PromotionInfo*), which contains other nested attributes.

# DEMO

Creating DynamoDB table from Console

# DynamoDB Primary Key

**What is Primary Key:** The primary key uniquely identifies each item in the table, so that no two items can have the same key.

- **Partition key** - A simple primary key, composed of one attribute known as the *partition key*.  
Example - The *People* table has a simple primary key (*PersonID*).
- **Partition key and sort key** - Referred to as a *composite primary key*, this type of key is composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*. No two items can have same combo of these two attributes.

Example - The *Music* table has composite primary key (*Artist* and *SongTitle*). You can access any item in the *Music* table directly, if you provide the *Artist* and *SongTitle* values for that item.

Partition key is also referred as *hash attribute*, sort key also referred as *range attribute*

# DEMO

How to write items into DynamoDB from  
Lambda

# DEMO

How to delete DynamoDB items using Lambda

# DynamoDB Secondary Indexes

**Why Secondary Indexes:** A *secondary index* lets you query the data in the table using an alternate key, in addition to queries against the primary key.

- **Global secondary index** - An index with a partition key and sort key that can be different from those on the table.
- **Local secondary index** – An index that has the same partition key as the table, but a different sort key.

**Note:** Table design should be done in such a way that, you can achieve maximum efficiency with minimum number of indexes. Index updates can be expensive!

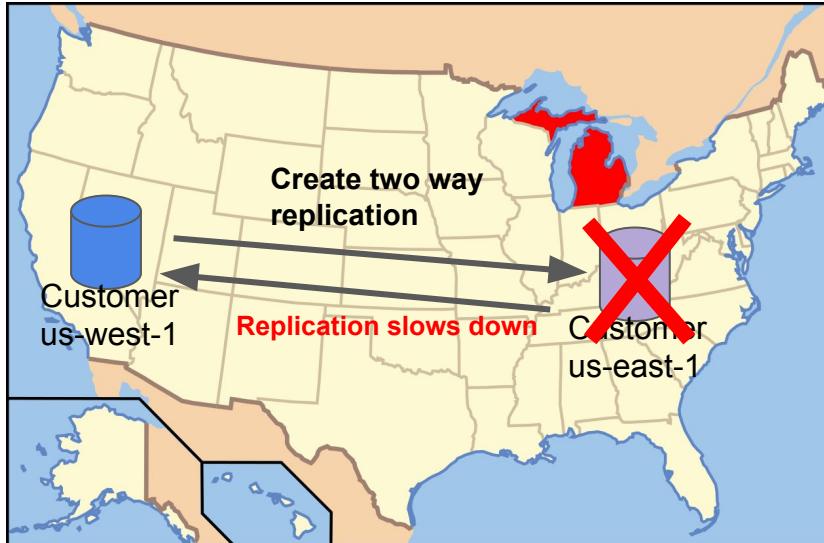
# DynamoDB Secondary Indexes



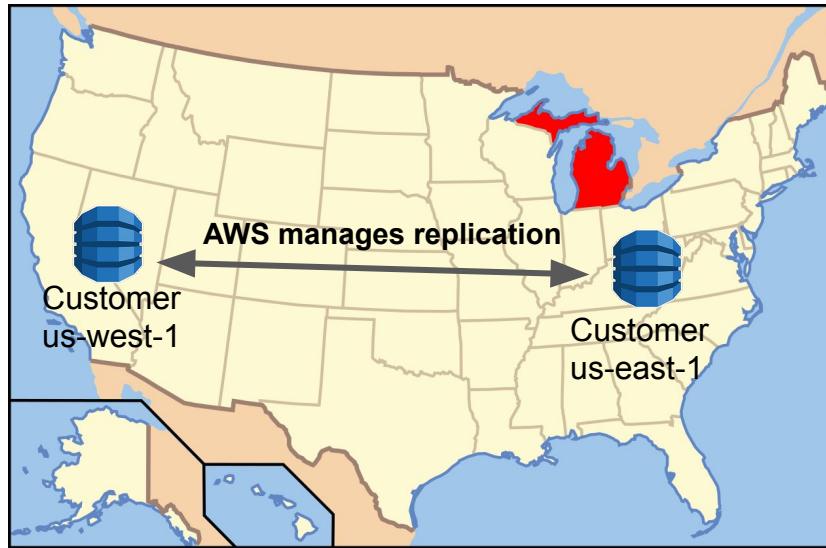
- *Music* table, with a new index called *GenreAlbumTitle*. In the index, *Genre* is the partition key and *AlbumTitle* is the sort key.
- Query the data either by *Artist and/or SongTitle* OR *Genre and/or AlbumTitle*

# DynamoDB Global Tables

## Simulating Global Tables



# DynamoDB Global Tables



## When to use it?

- Massively scaled application, with globally dispersed users
- Require world-wide low latency data
- Does cost extra!

# DynamoDB Read Consistency

## Concept of multi-AZ durability

AZ1

```
{"acctno":1234,  
 "balance":500.00  
 }
```

AZ2

```
{"acctno":1234,  
 "balance":500.00  
 }
```

AZ3

```
{"acctno":1234,  
 "balance":500.00  
 }
```

Update bank balance of acctno 1234 from 500\$ to 700\$

AZ1

```
{"acctno":1234,  
 "balance":700.00  
 }
```

AZ2

```
{"acctno":1234,  
 "balance":500.00  
 }
```

AZ3

```
{"acctno":1234,  
 "balance":500.00  
 }
```

Need to replicate

Read  
balance, get  
old balance



# DynamoDB Consistent Read

- **Eventually Consistent Reads**
  - Response might return old data
  - If you repeat read request after a short time, latest data returned
- **Strongly Consistent Reads**
  - Returns most up-to-date data
  - By default reads are eventually consistent
  - Reads can be made strongly consistent by setting *ConsistentRead* parameter to true
  - Strongly consistent reads have less throughput (KB/Second) than eventually consistent reads

# DynamoDB Reads and Writes

DynamoDB read and write throughput is measured through capacity units

- **One read capacity unit**
  - One strongly consistent read per second
  - Two eventually consistent read per second
  - For item upto 4KB in size, for larger item, more capacity unit is required
- **One write capacity unit**
  - One write per second
  - For item upto 1 KB, for larger item, more capacity unit is required

## Math Time!

**Q** - If you create a table with 10 read and 10 write capacity units, assuming 4 KB item size for read and 1 KB item size for write, what would be the throughput (KB/Second) for strongly consistent read and write?

**A** - 1 read capacity unit = 1 strongly consistent read/second for 4 KB

10 read capacity unit = 10 strongly consistent read/second for 4 KB each = **40 KB/Second** (10 read/sec X 4 KB)

Similarly, for write, 1KB X 10 capacity units = **10 KB/Second**

# DynamoDB Auto Scaling

What have you learnt so far on DynamoDB read/write?

- Difference between eventual and strongly consistent reads
- Math behind read and write capacity units
- Given the lowest and highest load in KB/second for your application
  - You can derive lowest and highest read/write capacity units

## Time to set up DynamoDB Auto Scaling!

Auto Scaling

<input checked="" type="checkbox"/> Read capacity	<input checked="" type="checkbox"/> Write capacity
<input type="checkbox"/> Same settings as read	
Target utilization	70 %
Minimum provisioned capacity	5 units
Maximum provisioned capacity	500 units
<input checked="" type="checkbox"/> Apply same settings to global secondary indexes	<input checked="" type="checkbox"/> Apply same settings to global secondary indexes

# DynamoDB On Demand - Pay per Request Billing

Announced at re:Invent 2018

## When to use DynamoDB On Demand?

- Useful if application traffic difficult to predict
- Workload has large spikes of short duration

## What is DynamoDB On Demand?

- Scaling without capacity planning
- Pay-per-request pricing, only pay for what you use
- Possible to switch back and forth between Provisioned and On-demand mode
- Supports all Dynamo-DB features - encryption, global tables, point in time recovery etc.
- Indexes created on the table inherit same scalability and billing model

## Pricing

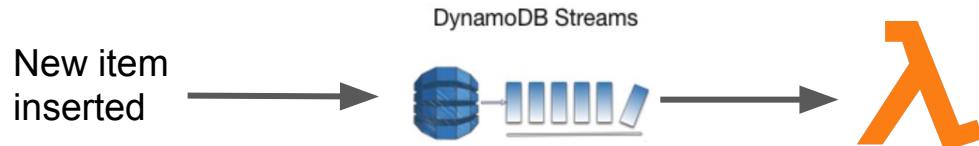
- \$ 1.25 per million write requests, \$ 0.25 per million read requests

# DynamoDB Streams

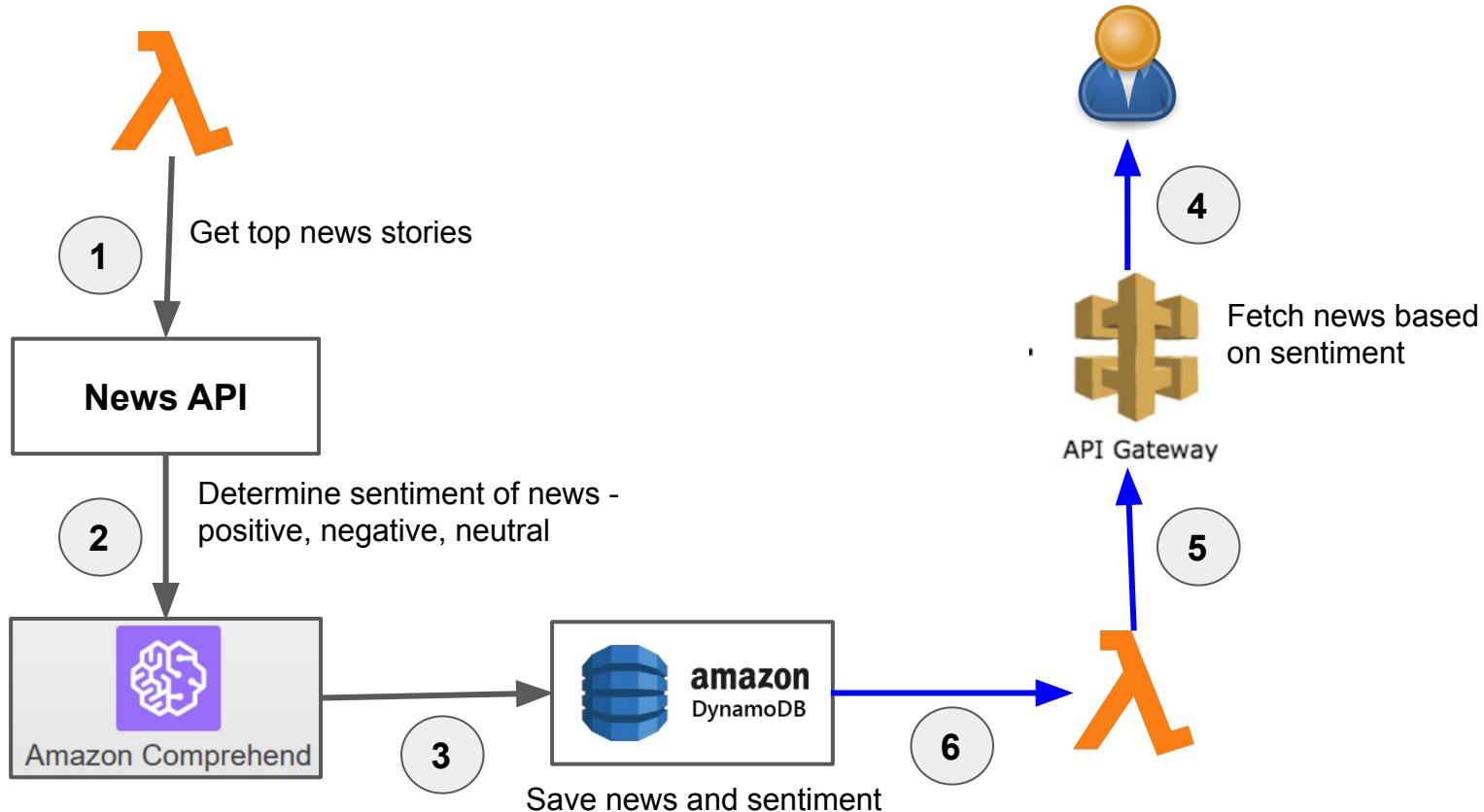
**What is DynamoDB Streams:** An optional feature that captures data modification events in DynamoDB tables. The data about these events appear in the stream in near real time, and in the order that the events occurred.

Events that triggers stream record:

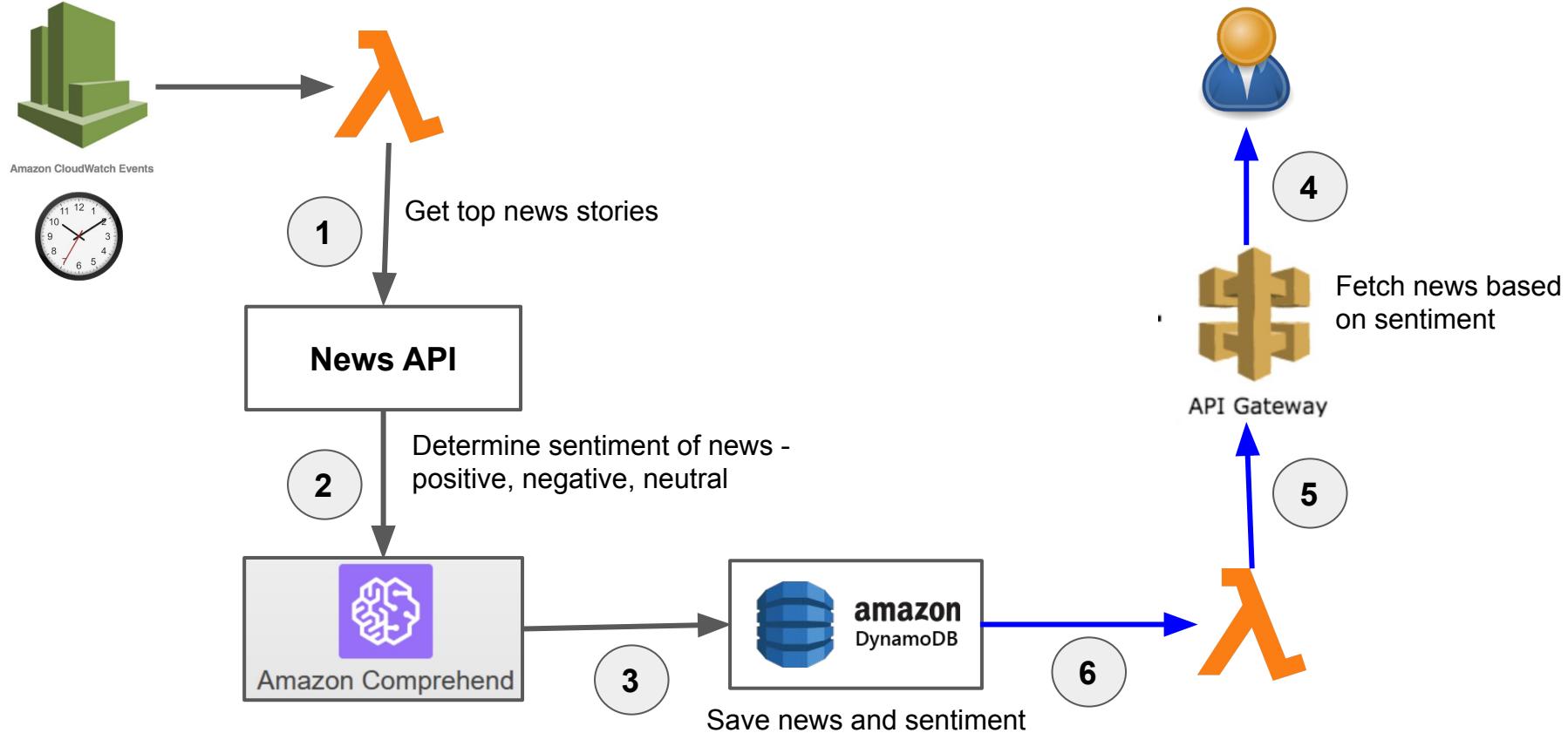
- A new item is added to the table
- An item is updated
- An item is deleted from the table



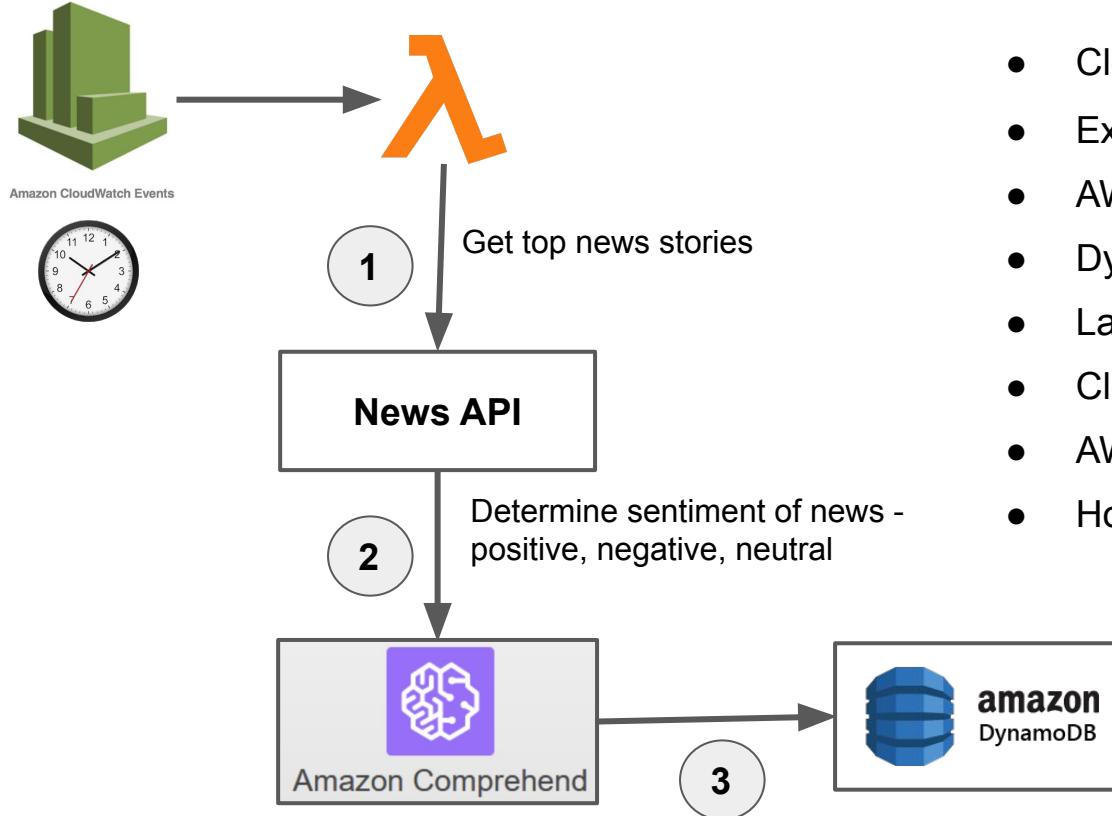
# Lambda Real Life Project



# Lambda Real Life Project



# Lambda Real Life Project - Part 1 Recap



- Cloud9
- External Dependencies
- AWS Role
- DynamoDB
- Lambda Write/Delete into DynamoDB
- Cloudwatch Rules
- AWS Comprehend
- Homework - AWS Secrets Manager

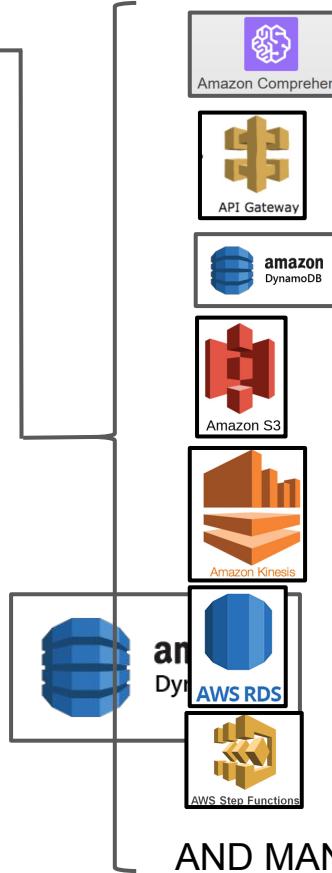
# Lambda Real Life Project - Final Words



Amazon CloudWatch Events



- Easy Integration
- Secure
- Focus on Service, not Servers!
  - Automatic AutoScaling
  - No AMI to maintain
  - No patching
  - Economical



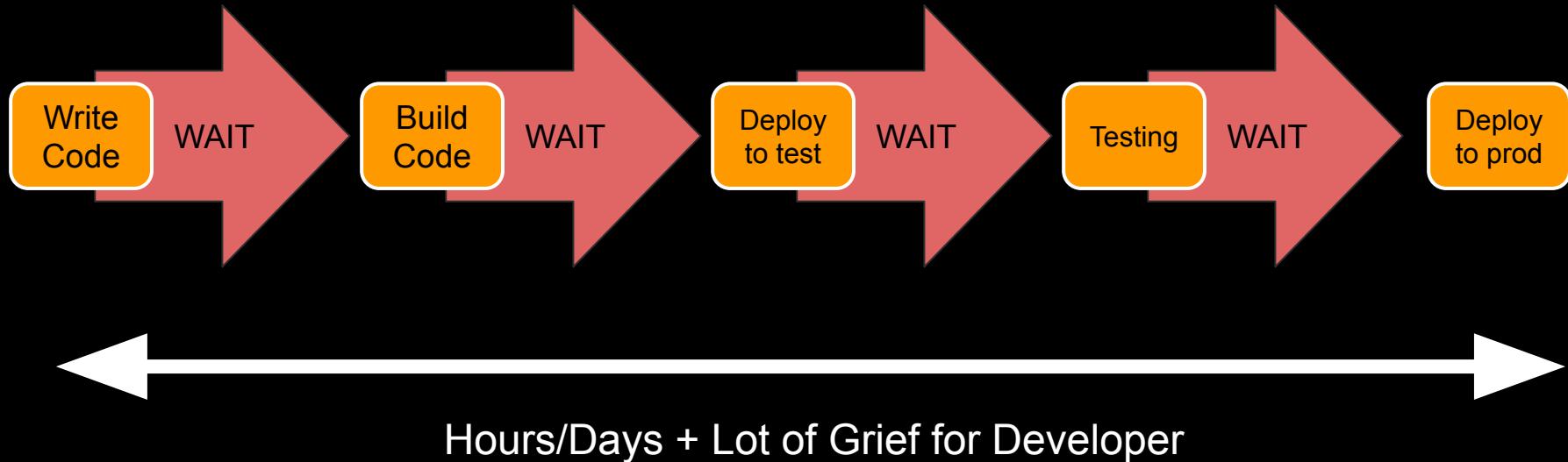
AND MANY MORE

# DevOps for Serverless

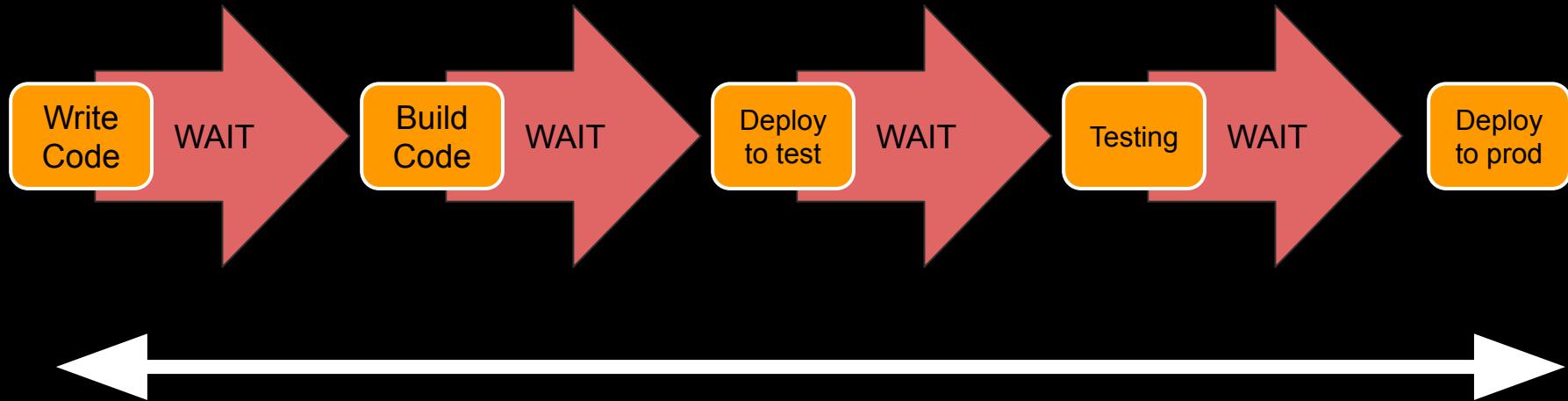


**SOFTWARE MOVES FASTER TODAY!**

# Traditional Software Delivery

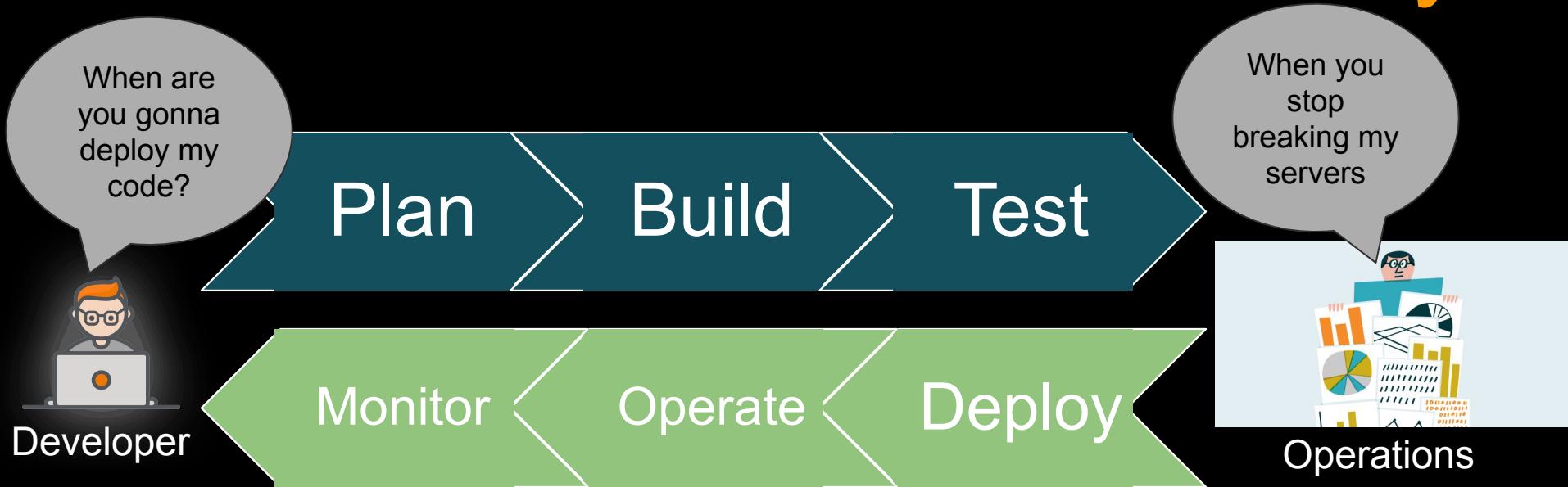


# Traditional Software Delivery



Hours/Days + Lot of Grief for Developer & Operations

# Traditional Software Delivery

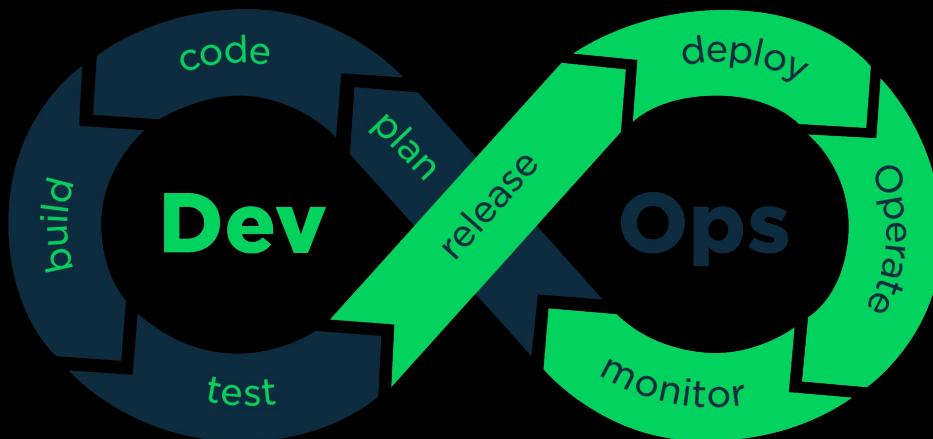






# What is DevOps?

- Word “DevOps” coined in 2009 by Patrick Debois
- Combination of cultural philosophies, practices, and tools
- Development and Operations teams are no longer “siloed”



# Why DevOps?

How long would it take your organization to deploy a change that involves a single line of code?

Can you do this on a repeatable reliable basis?

DevOps Vs Non-Devops organizations:

**4x**

Lower change failure rate

**24x**

Faster recovery times

**200x**

More frequent deployments

**44%**

More time spent on new features and code

*Source: Puppet 2017 State of DevOps Report*

# General DevOps Practices

- Automate everything!
- Deploy frequently rather than one mega deployment in months
- Codify every step - infrastructure, application and more
- Rome was not built in a day!

FASTER DELIVERY

RELIABILITY

SCALE

INNOVATION

IMPROVED COLLABORATION

CULTURAL BENEFITS



# DevOps Phases

# DevOps Phases

Source → Build → Test → Prod

- Check-in source code
  - Compile code
  - Unit tests
  - Create artifacts
- Integration tests with other systems
  - Load testing
  - UI tests
  - Penetration testing
- Deployment to production environments

# DevOps Phases - CI/CD

Source

Build

Test

Prod

Continuous Integration

Continuous Delivery

Continuous Deployment



# Why DevOps on AWS?

- Fully Managed
- Built for Scale
- Inherent Integration with other AWS Services
- Apply Existing Knowledge of AWS SDK
- Secure
- Pay As You Go + Free Tier
- DevOps on AWS job is on high demand

# DevOps On AWS - A Glimpse

Source

Build

Test

Prod



AWS  
CodeCommit



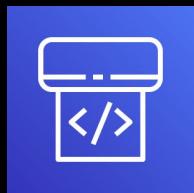
AWS  
CodeBuild



AWS CodeBuild  
+ Third Party Tools



AWS  
CodeDeploy



AWS CodePipeline

# What is AWS CodeCommit

- Code Repository
  - Think of GitHub, GitHub Enterprise, BitBucket on Steroids! (*More on this on next slide*)
- Store Anything
  - Source Code to binaries
- Based on Git
  - Works seamlessly with existing Git tools

# AWS CodeCommit Benefits

- Fully Managed
  - Eliminates need to host, maintain, backup, scale your own source control servers
  - Scales automatically
- Secure
  - Automatic encryption at rest & in transit
  - IAM integration
- High Availability
- Collaboration
- Inherent integration with AWS Services

# Diving Deeper

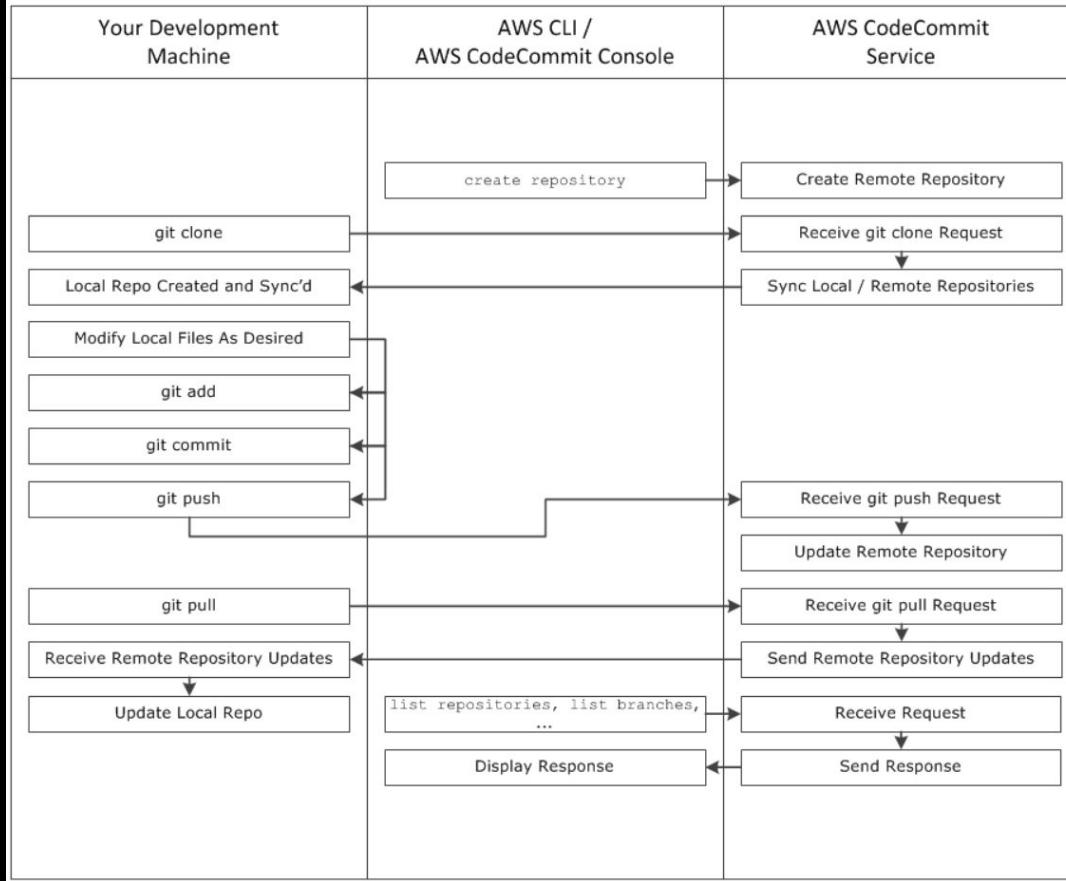
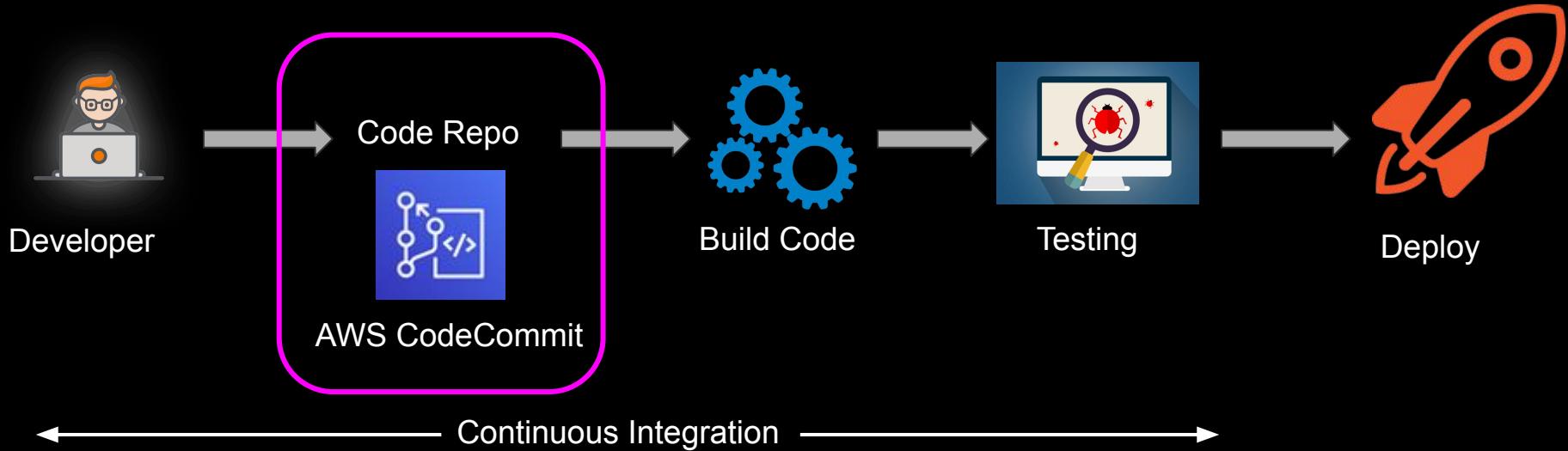


Image: CodeCommit User Guide (<https://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html>)

# Big Picture



# Branch Visualizer

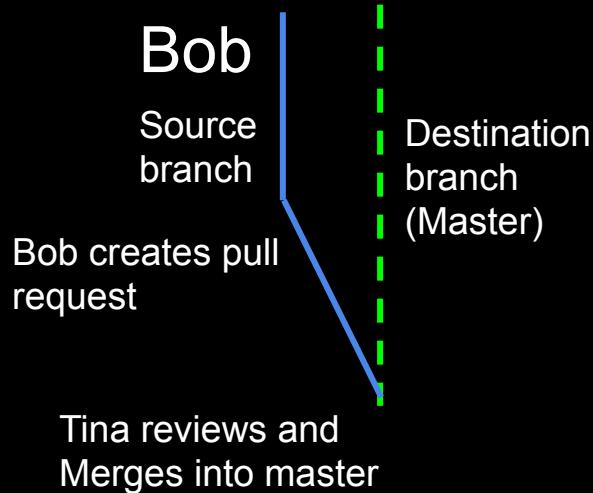
# What's Different in Real World?

- Create a repository from Console
- Clone the repo to Cloud9
- Add a program to the local clone
- Push your first commit back to the repo !
- Code review (Branch, Pull Request)
- Code merged to Master

# In This CodeCommit Demo

- Create a Branch - *Junior Developer*
- Create Pull Request - *Junior Developer*
- Review Pull Request - *Senior Developer*
- Merge into Master - *Senior Developer*
- Controlling Access to Repo - *Junior Vs Senior Developer*

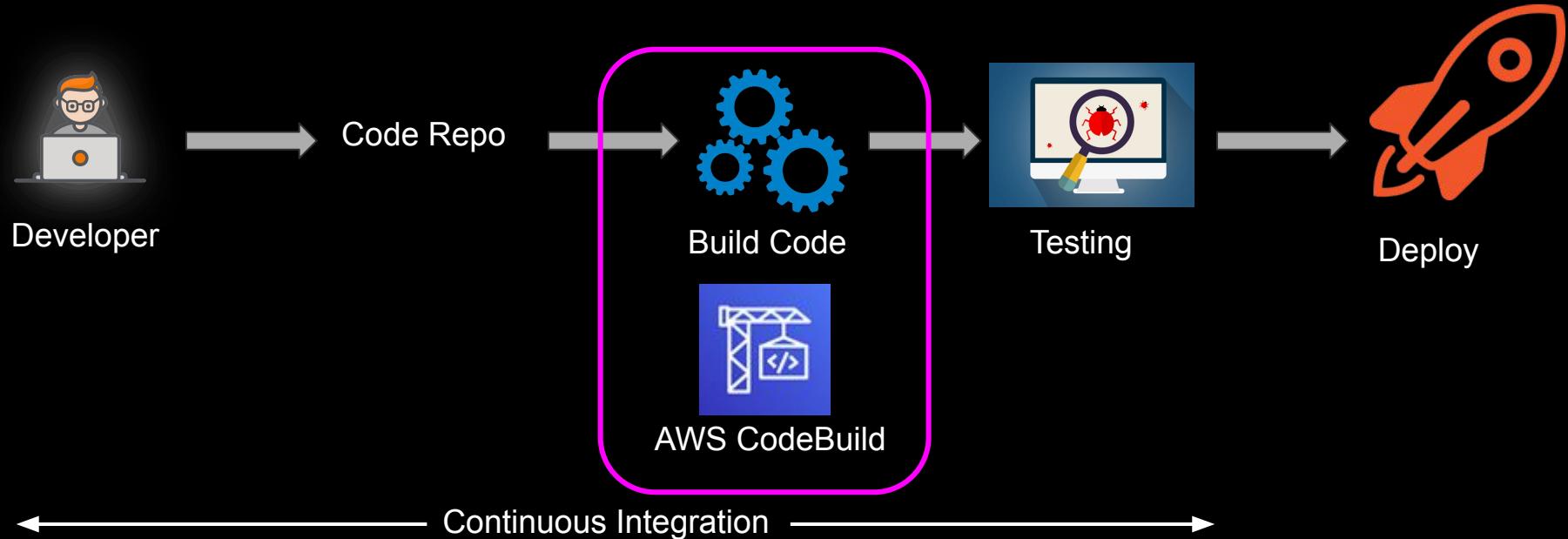
# In This CodeCommit Demo



# What is AWS CodeBuild

- Fully Managed Continuous Integration (CI) Service
  - Compiles source code
  - Runs tests
  - Produces ready to deploy software packages
- No need to provision and manage build servers
  - Scales automatically
  - Processes multiple builds concurrently
  - Use prepackaged or custom build environments
- Pay as you go
- Secure

# Big Picture



# Elephant in the Room - Jenkins

## Jenkins

## CodeBuild

Need to maintain Master and Worker Nodes in VM

Pay for idle resources

You take care of availability and scalability

Very mature plugin ecosystem

Can use CodeBuild as worker node

No need to provision and manage any Server

Pay as you go

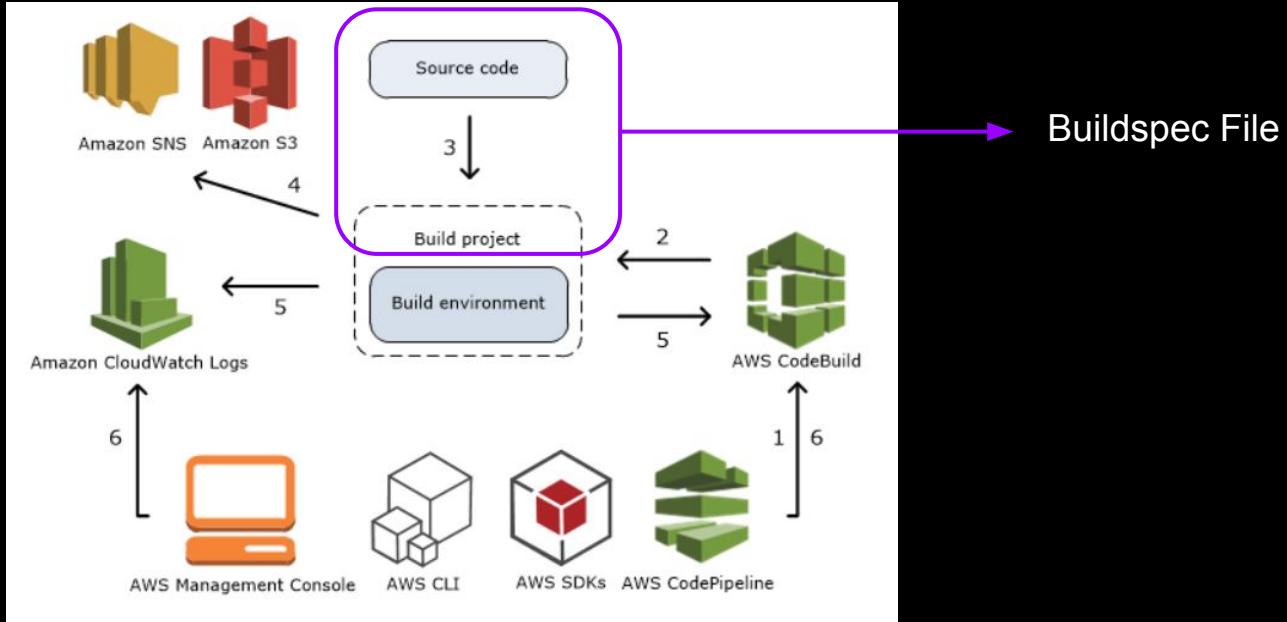
Fully managed and scales automatically

Have some prepackaged environments

Inherent integration with other AWS Services

# CodeBuild Flow

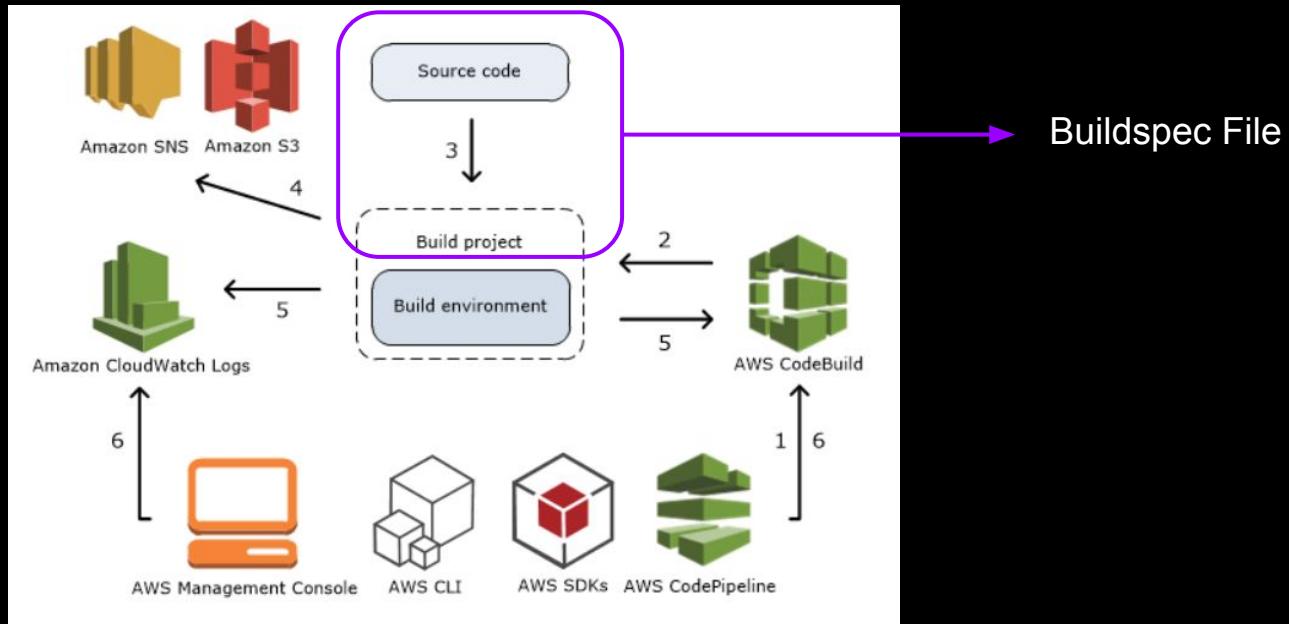
# How CodeBuild Works



Demo will be shown for all these steps

# Buildspec

# Build Specification

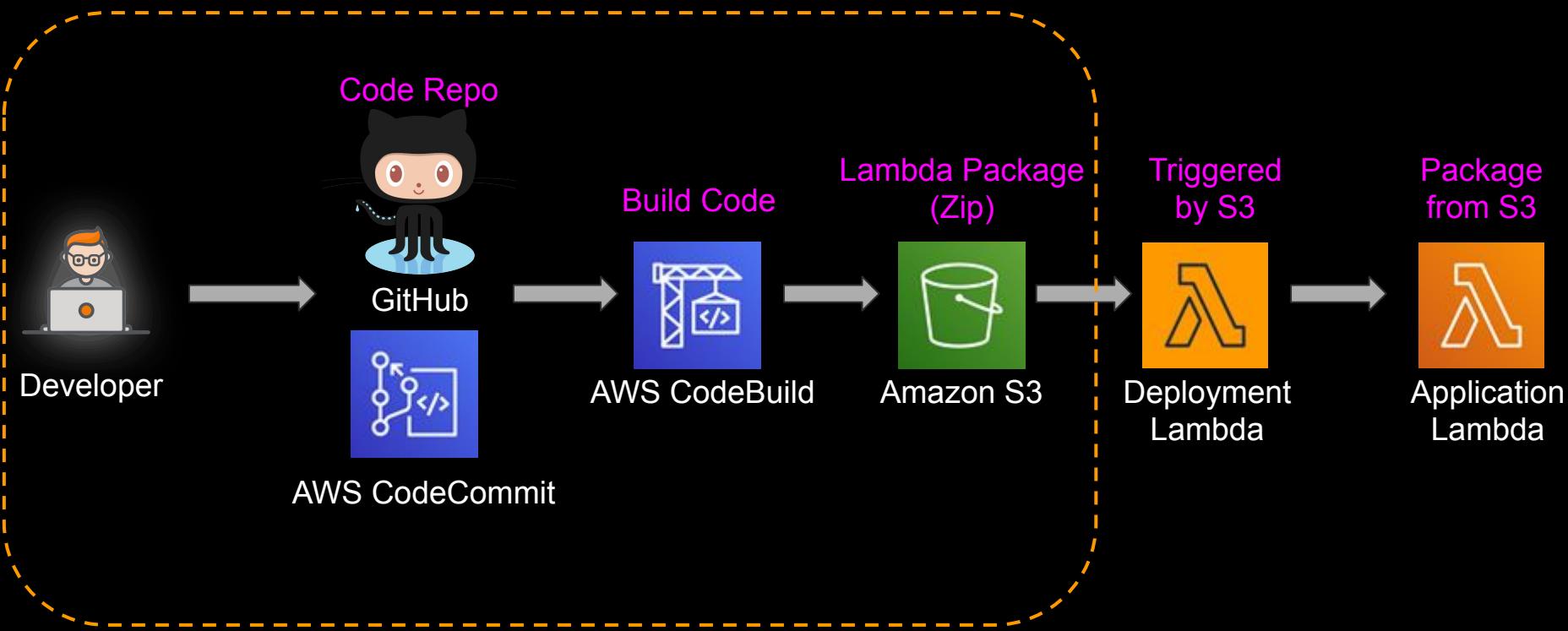


# Buildspec.YML

- Codebuild spins up a docker image and buildspec.yml tells what commands to run for installing packages and building code
- Docker images comes with common programming languages installed
- By default, buildspec file must be named buildspec.yml and placed in the root of your source directory in repo
- Must be in YAML format

# CodeBuild Demo

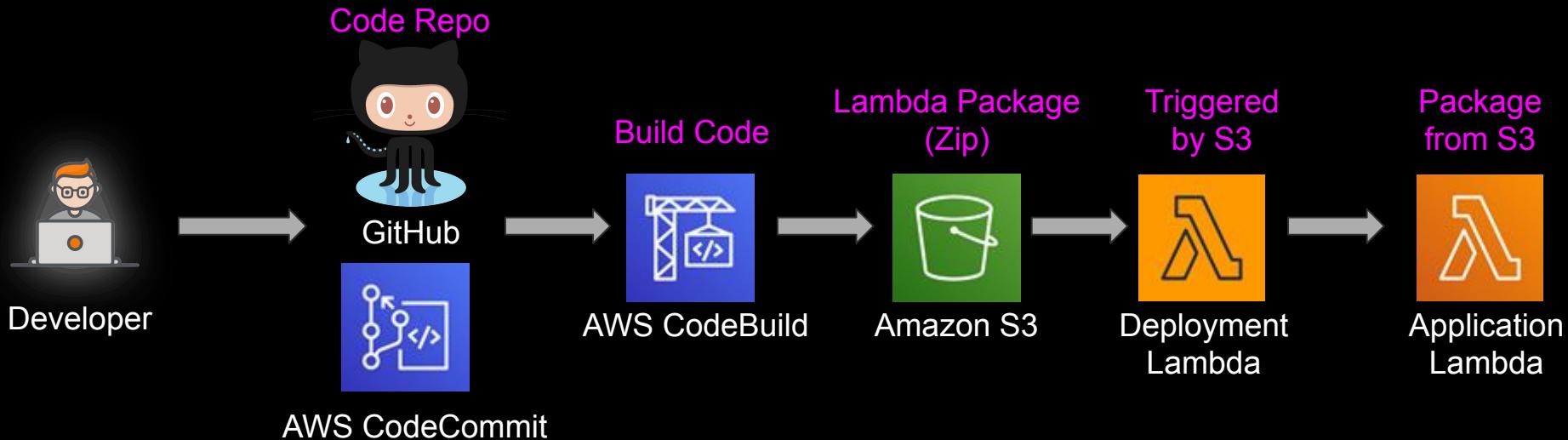
# In this CodeBuild Demo



# Many Ways to Skin a Cat!

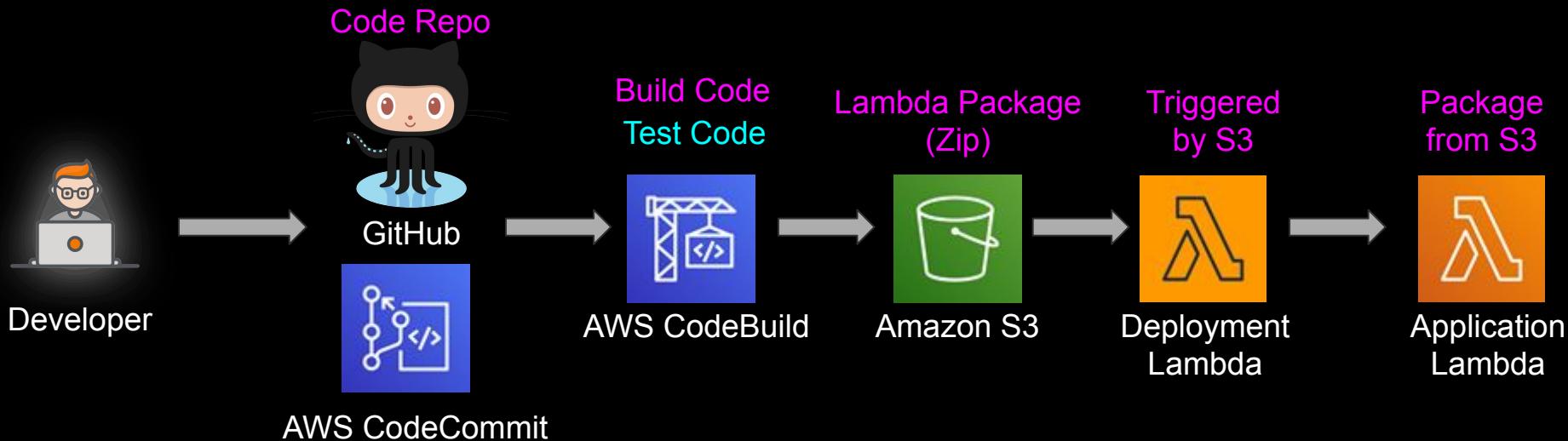
- DevOps is deploying a change on a repeatable, reliable and automated way
- Whichever tools and methods let's you do it in a simple fashion for your organization, is A OK
- No need to have a defined, fancy tool for each DevOps stage
  - Sometimes a Lambda can do the job
  - Sometimes ServiceCatalog API
  - Sometimes CodePipeline
  - Sometimes Jenkin
  - Sometimes Combination of all above
  - Choose based on your requirement and hold your head high!

# CI/CD Flow-1 for Lambda



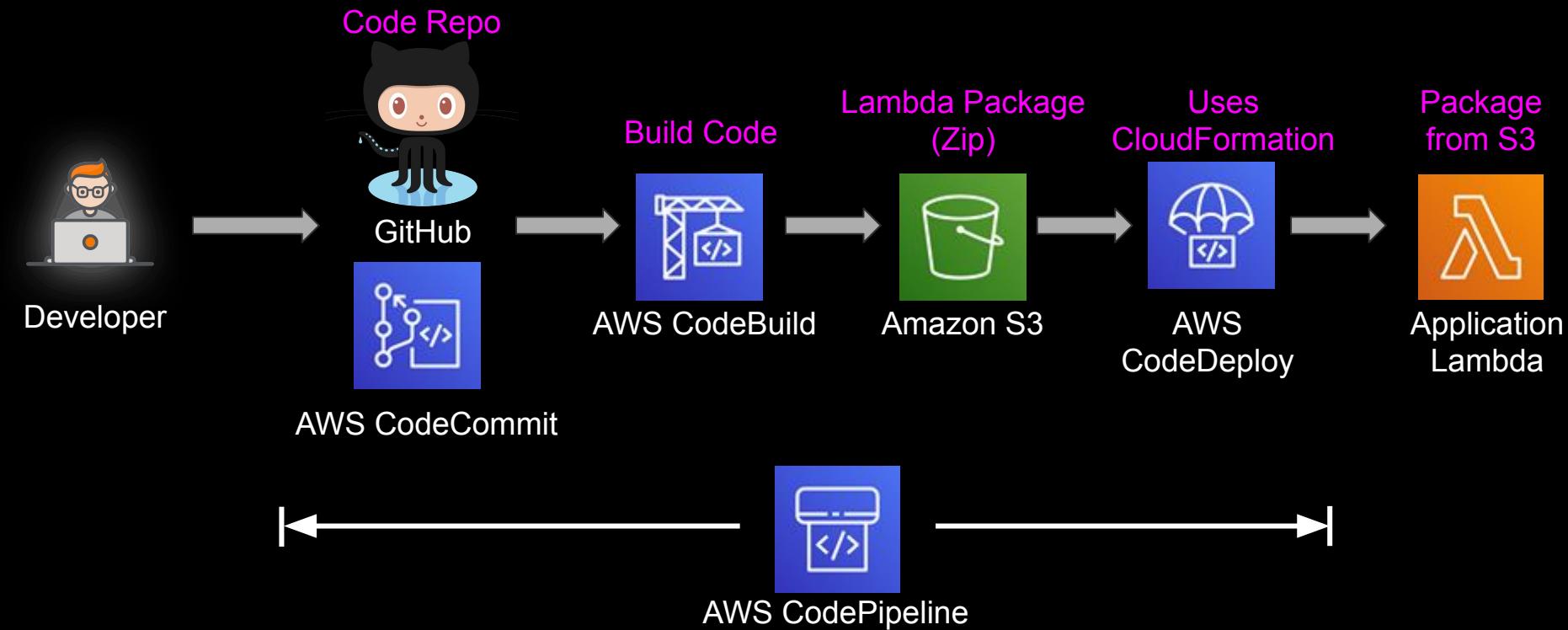
Deploying Lambda using Lambda - That's How Cool We Are!

# CI/CD Flow for Lambda

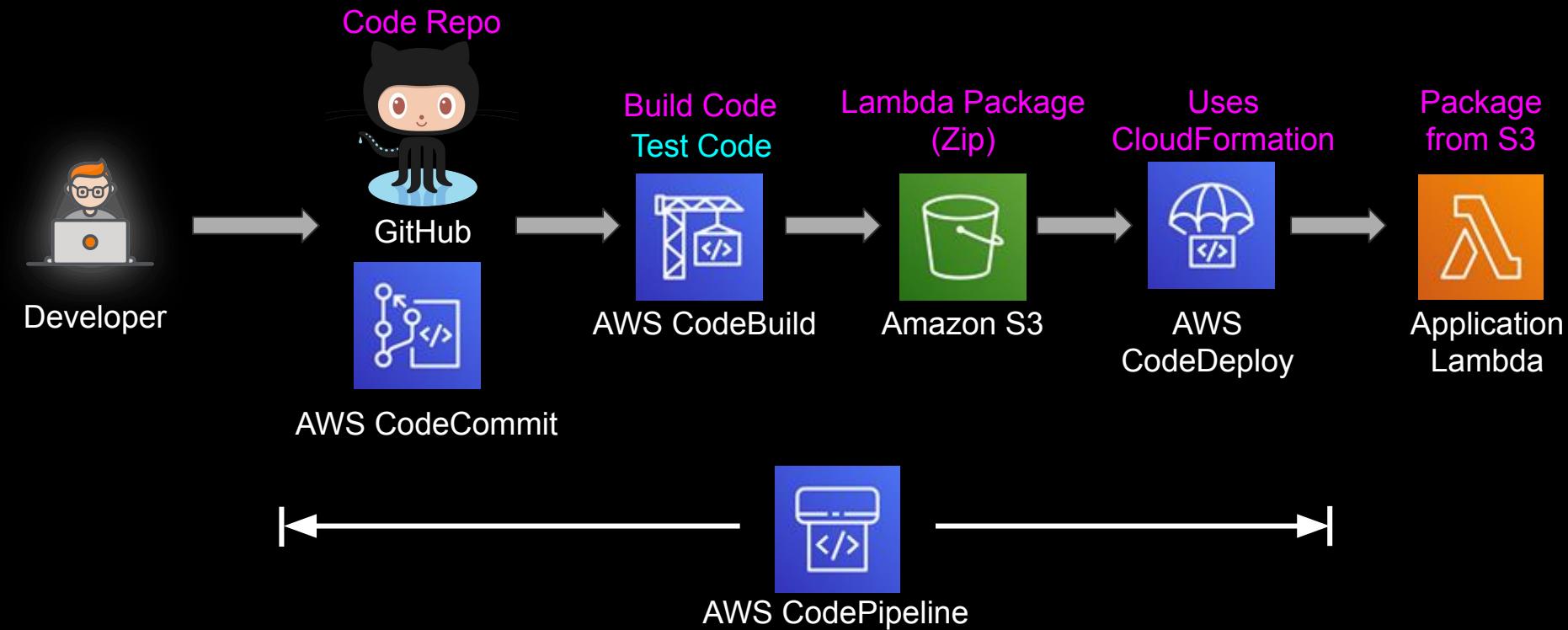


Deploying Lambda using Lambda - That's How Cool We Are!

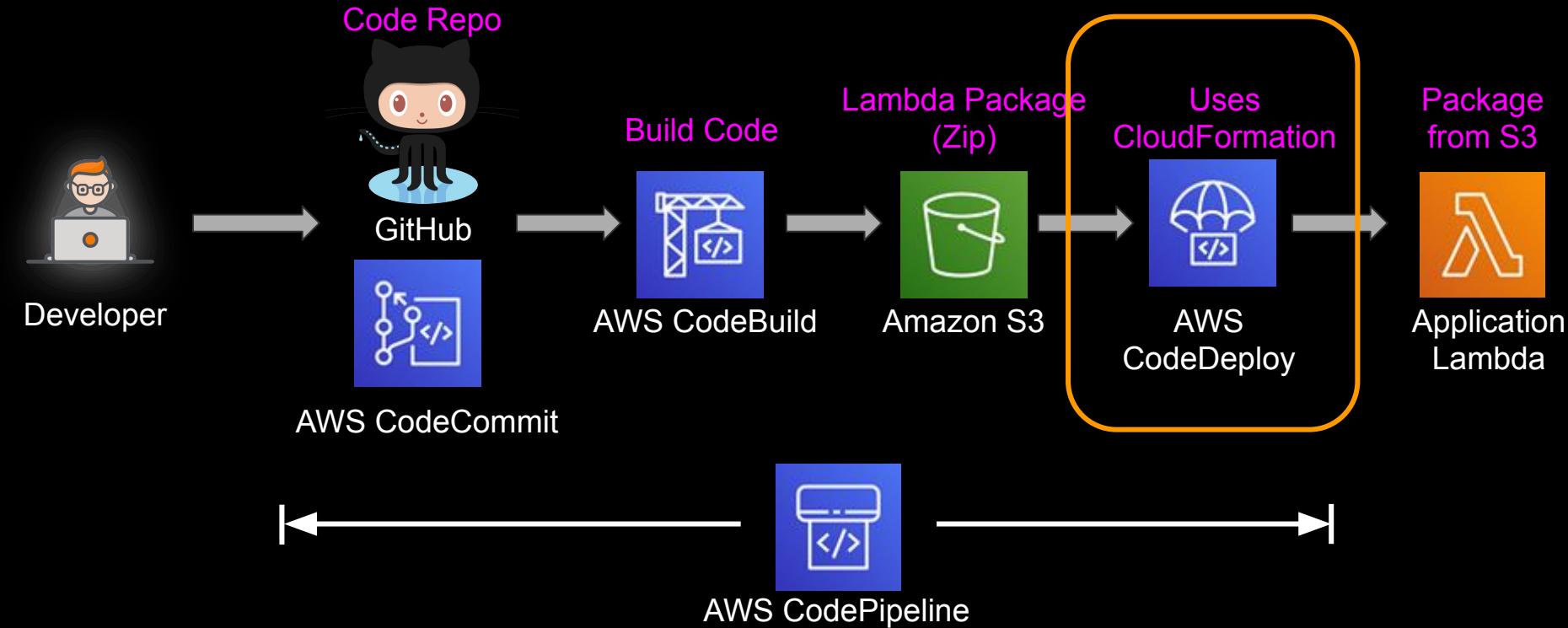
# CI/CD Flow-2 for Lambda



# CI/CD Flow-2 for Lambda



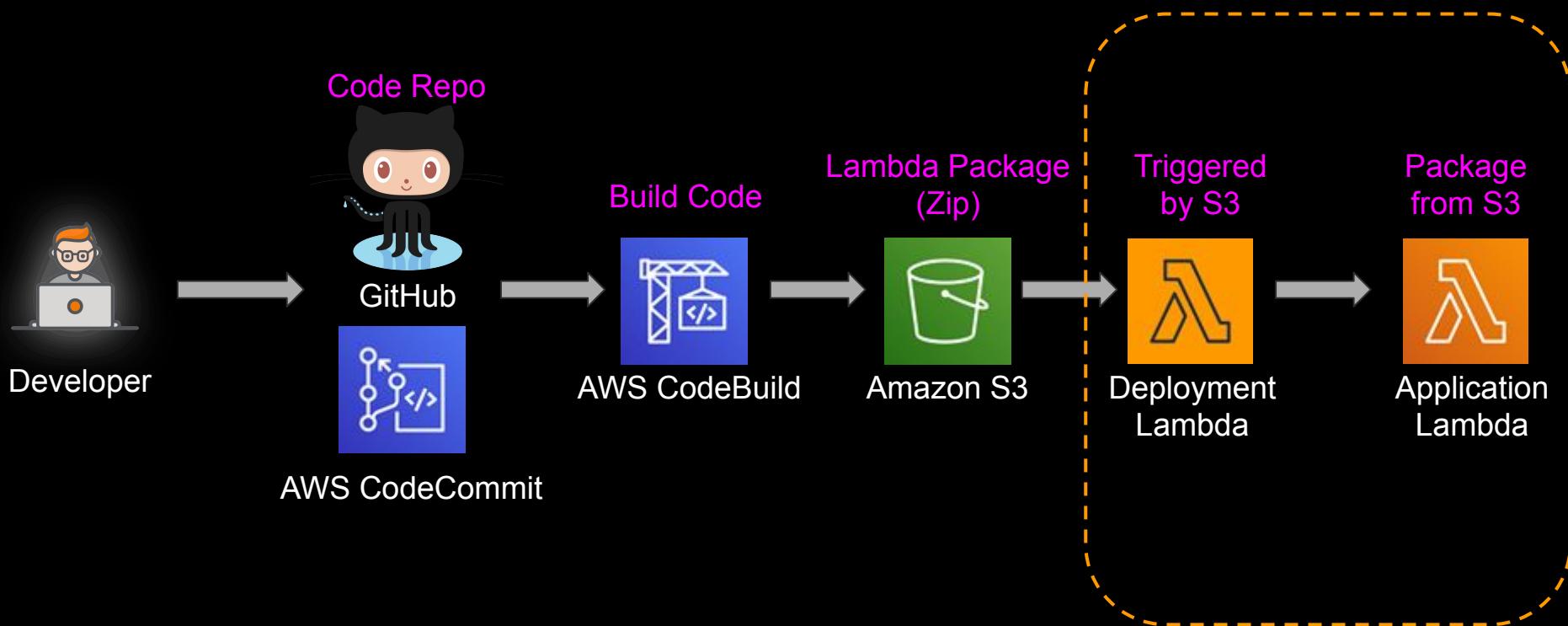
# Power of this Pipeline



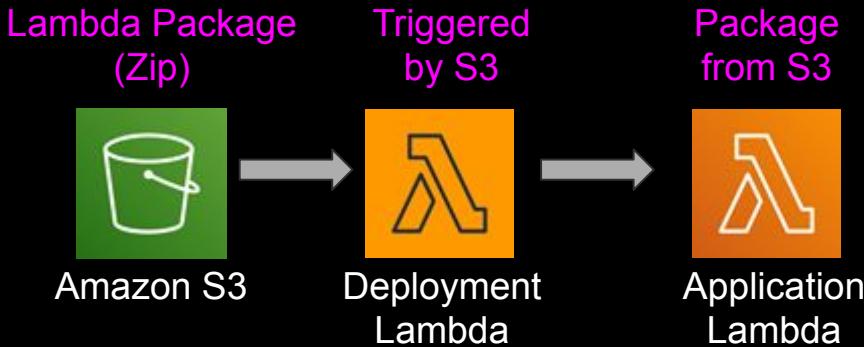
# In This Lecture

- Deployment Lambda
- End to End Flow

# In this CodeBuild Demo



# Deployment Lambda

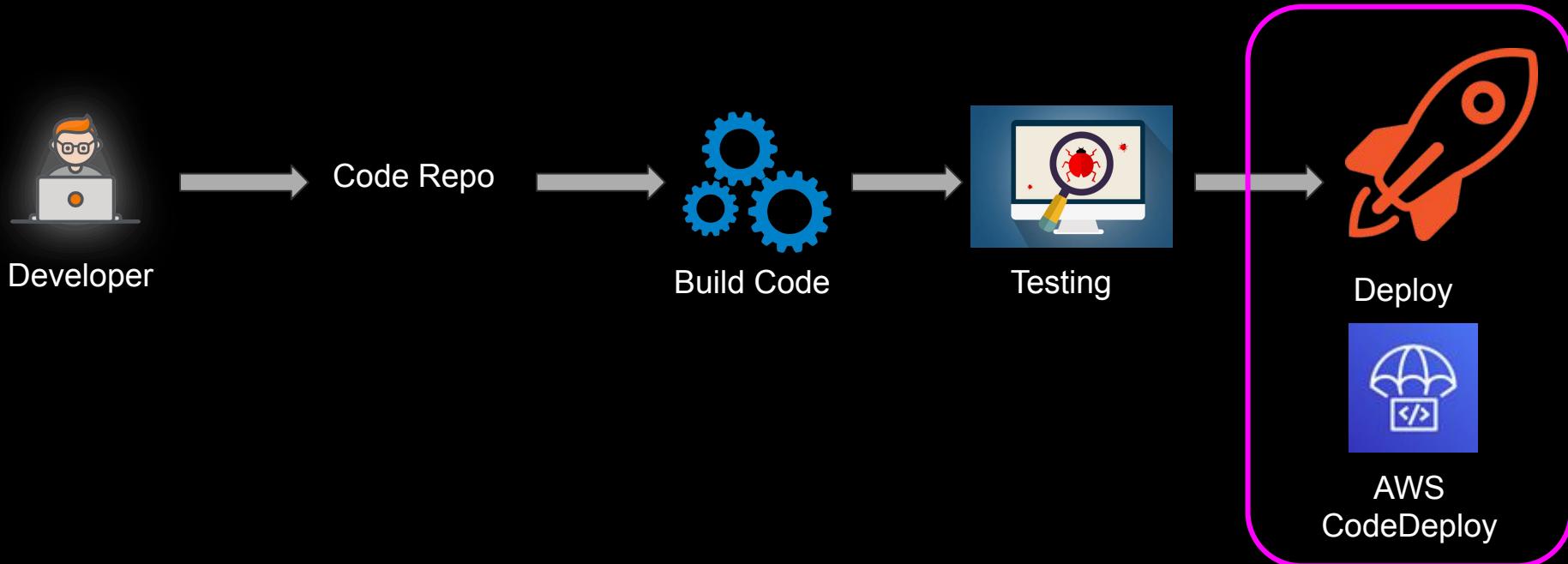


- Grab the Zip file from S3
- Use AWS-SDK to deploy the Lambda
  - Update for existing Lambda
  - Create for new

# What is AWS CodeDeploy

- Deployment service that automates application deployments to
  - EC2 instances, ECS, Lambda, on prem instances
- Can deploy variety of application content
  - Code, Lambda, Web and Config Files, Executables, Packages, Scripts, Multimedia files

# Big Picture



# AWS CodeDeploy Benefits

- Automatically Scales
  - Deploy to one or thousands instances
- Minimize Downtime
  - Rolling updates
  - Blue Green Deployments
- Stop And Roll Back
- Integrate with third party tools
- Concurrent Deployments

# CodeDeploy For Lambda

- CodeDeploy can NOT deploy Lambda code currently  
(As of 08/2019)
- CodeDeploy can switch Lambda traffic from one version to another
  - All at once
  - Canary
- CodePipeline CAN deploy Lambda code
  - CodeDeploy can switch traffic after deploy
  - Both are covered in the course :)

# AppSpec

- AppSpec file tells CodeDeploy what tasks to do in what order
- Think of it as conceptually similar to BuildSpec file in CodeBuild

# AppSpec

- Serverless Students can skip ahead to Lambda part

# AppSpec

- For EC2/On-Premises Compute
  - What source files to pick up from where
  - Where to deploy them
  - What scripts to run
- For Amazon ECS
  - Name of the Amazon ECS service and the container name and port used to direct traffic to the new task set
  - Functions to be used as validation tests

# AppSpec

- For AWS Lambda
  - The Lambda version traffic should be switched to
  - Functions to run as validation tests

# Revision & AppSpec

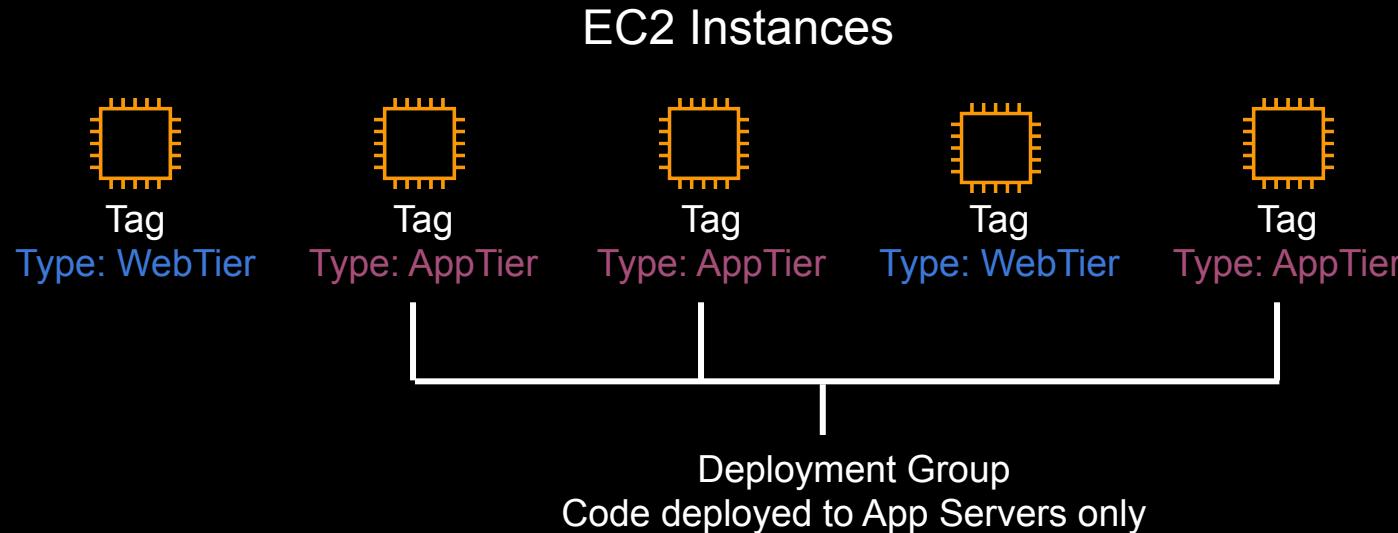
- Revision contains
  - Source files CodeDeploy will deploy to instances
  - Scripts CodeDeploy run on instances
- Revision contains AppSpec file
- For Lambda and ECS, a revision is same as AppSpec File

# Key Terms

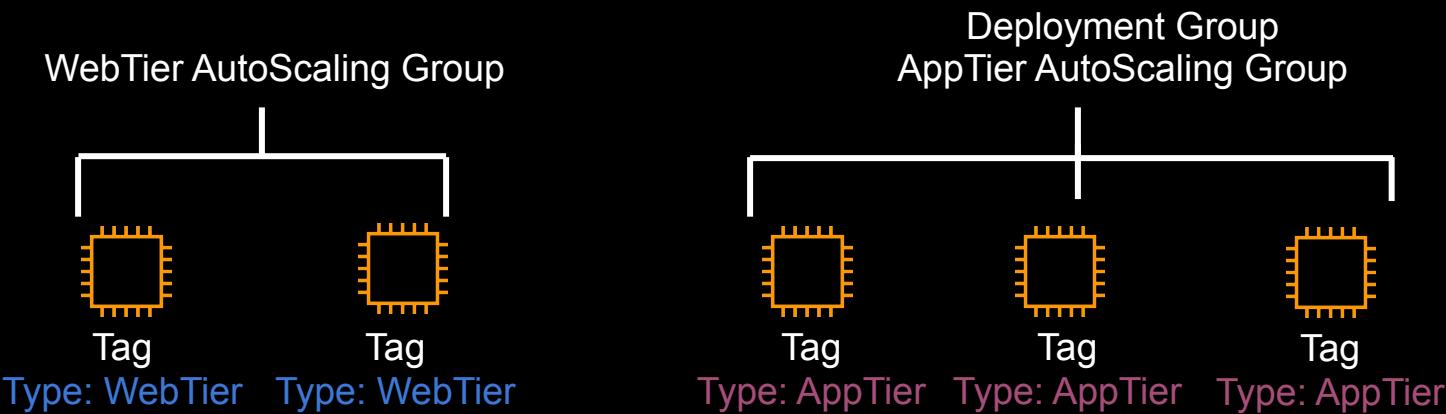
- Deployment Groups
- Deployment Type
- Deployment Configurations

# Deployment Groups

Set of Individual Instances



# Deployment Groups



# Deployment Groups for Lambda

- Set of CodeDeploy Configurations for future deployments
- Easier to understand with demo

# Deployment Type

## Deployment type

Choose how to deploy your application

- In-place

Updates the instances in the deployment group with the latest application revisions. During a deployment, each instance will be briefly taken offline for its update

- Blue/green

Replaces the instances in the deployment group with new instances and deploys the latest application revision to them. After instances in the replacement environment are registered with a load balancer, instances from the original environment are deregistered and can be terminated.

# Deployment Configurations

A deployment configuration is a set of rules that determines how fast an application will be deployed and the success or failure conditions for a deployment.

<b>CodeDeployDefault. OneAtATime</b>  Compute platform EC2/On-premises  Minimum healthy hosts value 1	<b>CodeDeployDefault. HalfAtATime</b>  Compute platform EC2/On-premises  Minimum healthy hosts value 50%	<b>CodeDeployDefault. AllAtOnce</b>  Compute platform EC2/On-premises  Minimum healthy hosts value 0	<b>CodeDeployDefault. LambdaAllAtOnce</b>  Compute platform AWS Lambda  Configuration type All at once
<b>CodeDeployDefault. LambdaLinear10Per centEvery1Minute</b>  Compute platform AWS Lambda  Configuration type Linear  Step 10%  Interval 1 minutes	<b>CodeDeployDefault. LambdaLinear10Per centEvery2Minutes</b>  Compute platform AWS Lambda  Configuration type Linear  Step 10%  Interval 2 minutes	<b>CodeDeployDefault. LambdaLinear10Per centEvery3Minutes</b>  Compute platform AWS Lambda  Configuration type Linear  Step 10%  Interval 3 minutes	<b>CodeDeployDefault. LambdaLinear10Per centEvery10Minutes</b>  Compute platform AWS Lambda  Configuration type Linear  Step 10%  Interval 10 minutes

# Putting It All Together

## Deployment to EC2

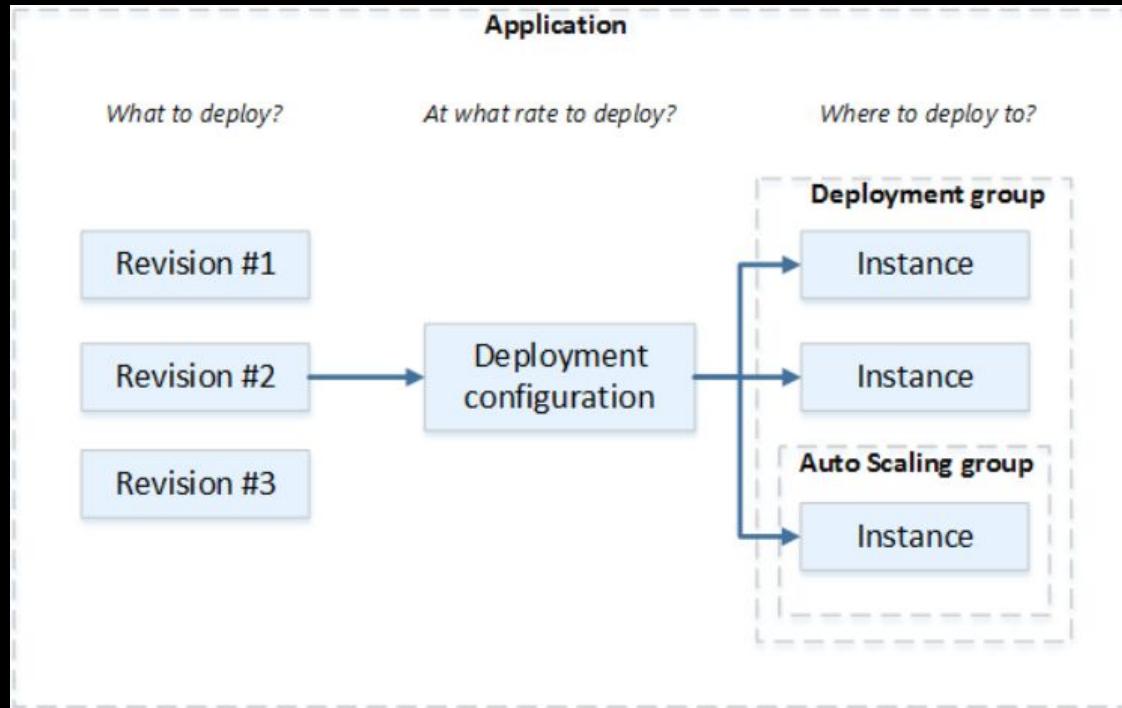


Image: <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-steps.html#deployment-steps-server>

# Putting It All Together

Deployment Components on AWS Lambda

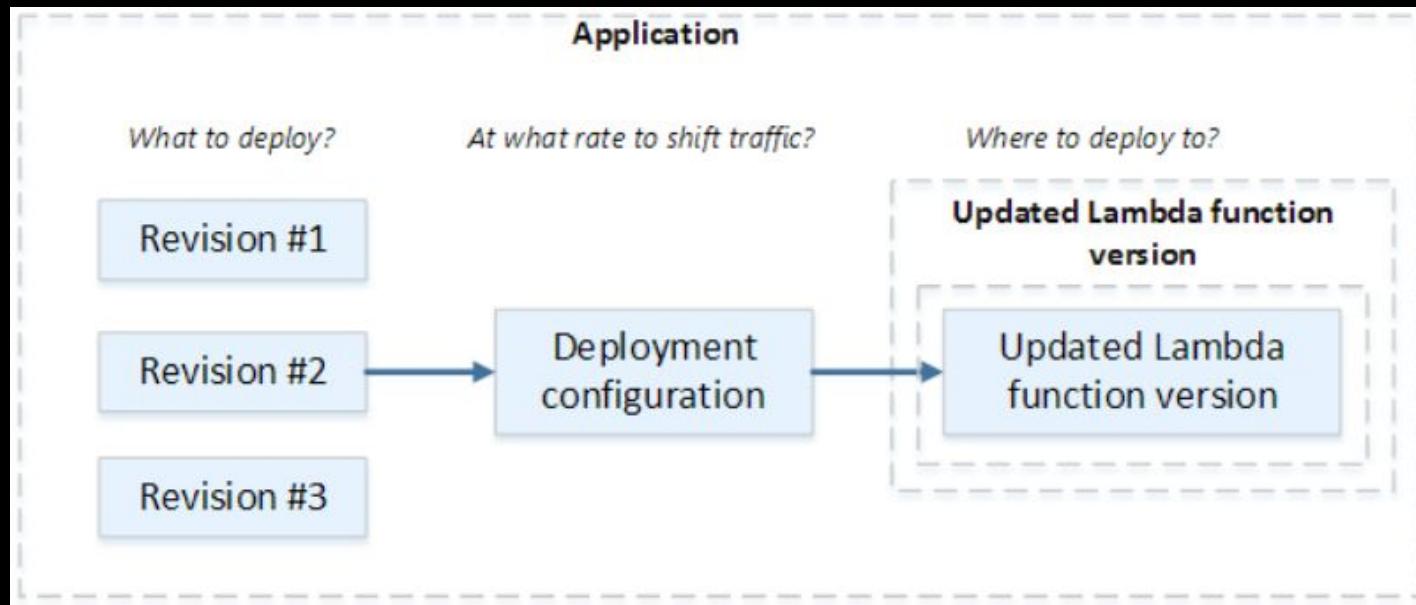


Image: <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-steps.html#deployment-steps-server>

# DEMO TIME!

- Deployment Group
- Deployment Type
- Deployment Configurations

# DEMO TIME!

- CodeDeploy for Lambda
  - Switch traffic to versions
  - Appspec file
- Deploying Lambda will be covered in CodePipeline section

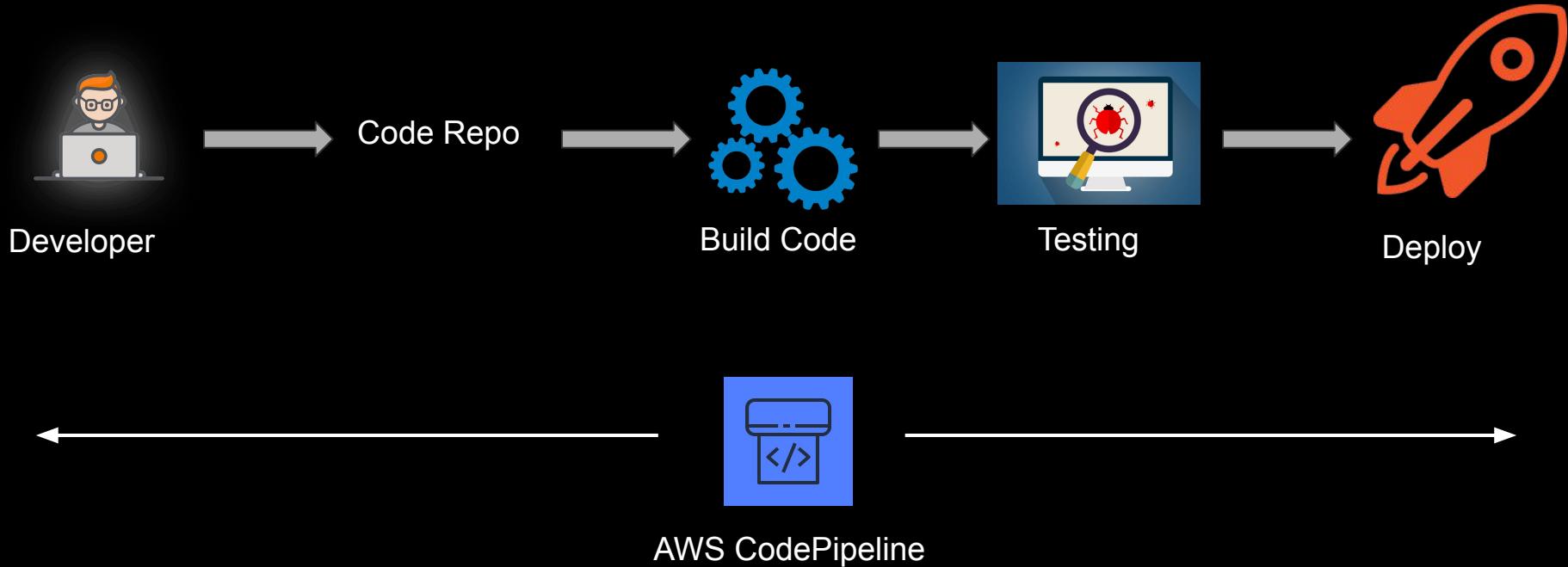
# What is AWS CodePipeline

- End to End CI/CD service
- Model, visualize and automate the steps required to release your software
- Service to orchestrate all the components, that we have covered so far, together

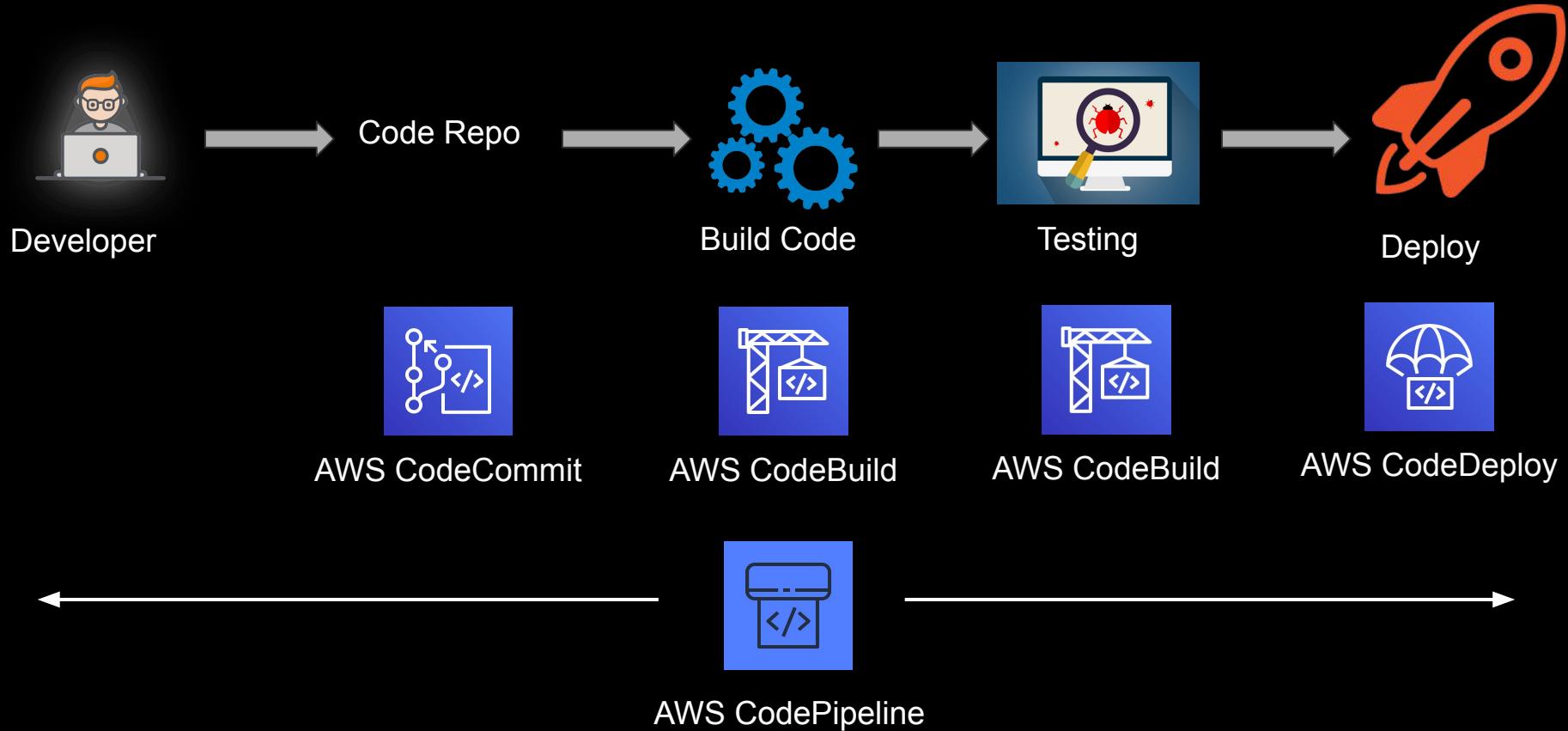
# What is AWS CodePipeline

- End to End CI/CD service
- Model, visualize and automate the steps required to release your software
- Service to orchestrate all the DevOps Components together

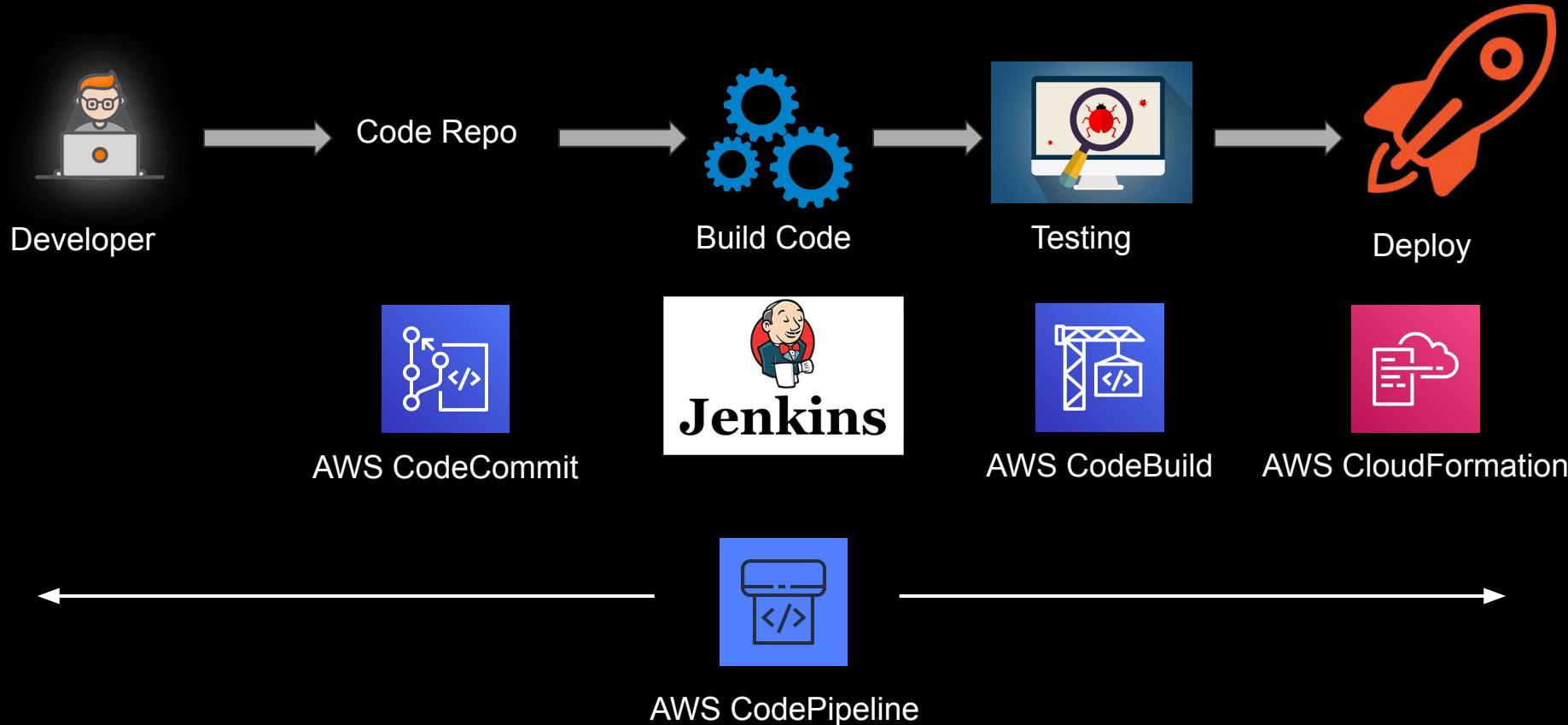
# AWS CodePipeline Big Picture



# AWS CodePipeline Big Picture



# AWS CodePipeline Big Picture



# AWS CodePipeline Benefits

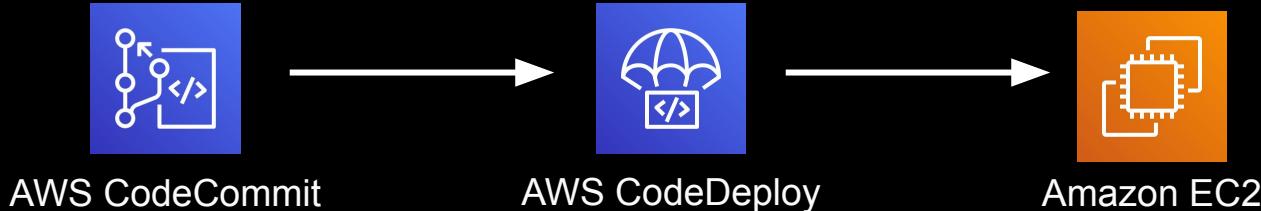
- Rapid Delivery
- Configurable Workflow
- Fully Managed
- Easy Integration
  - With AWS Services
  - Third Party Tools

# DEMO TIME!

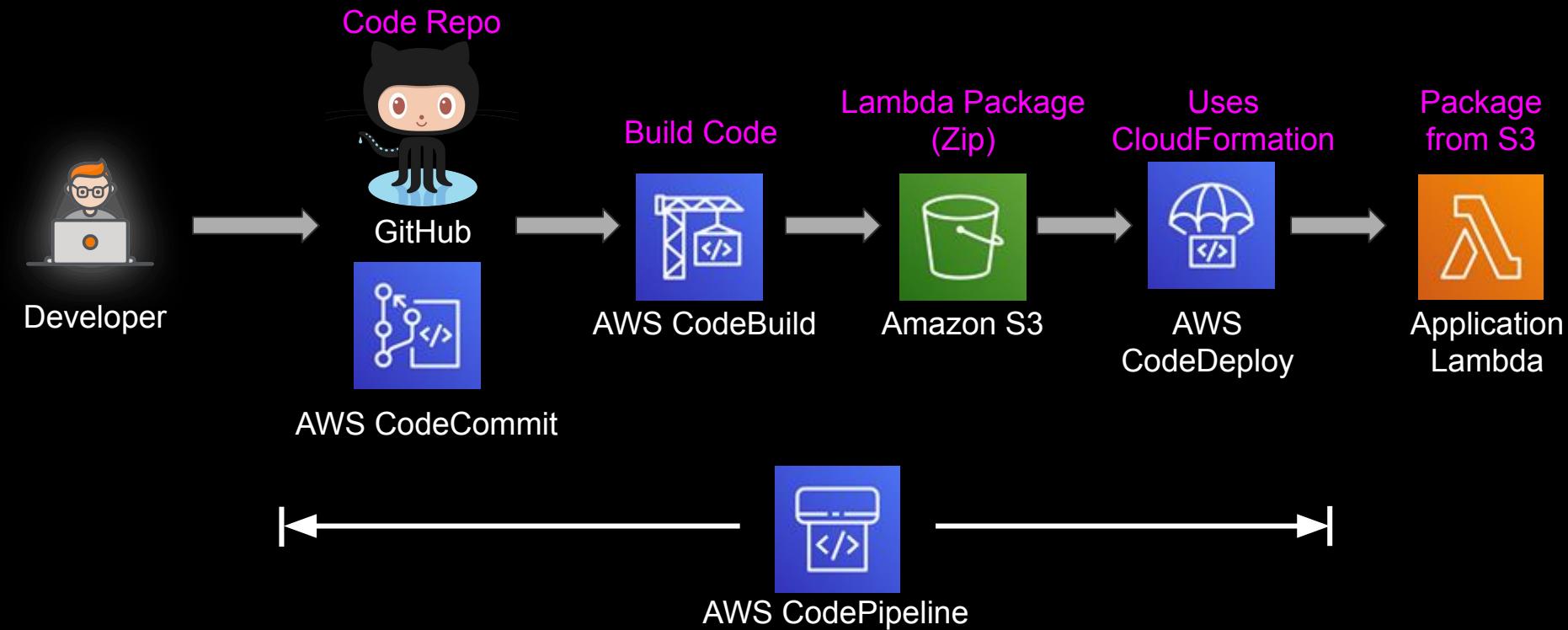
- Overall Look & Feel in Console

# DEMO TIME!

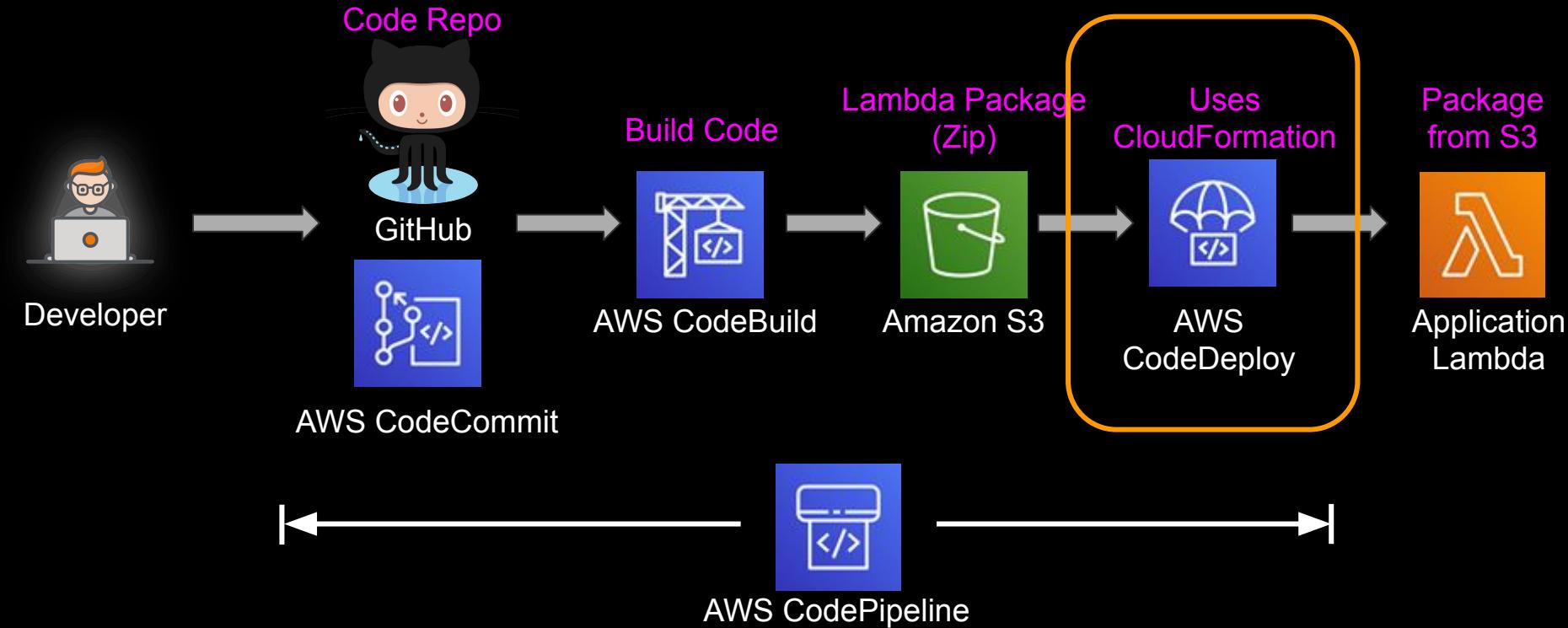
- Create a Simple Pipeline



# CI/CD Flow-2 for Lambda



# Power of this Pipeline



# CodeStar - What and Why



Primary Purpose

Quickly develop. Build and deploy apps on AWS. Includes multiple Serverless patterns

CI/CD

CodePipeline created. Each step is visualized.

Access to Code base

Code can be opened in IDE with one click. Natively integrated with Cloud9.

Track Sprints and Issues

Integrated with Jira and Github Issues. Can be tracked from CodeStar

Manage Project Team

Assign team member and roles from the service

# AWS SAM (Serverless Application Model)

# What is SAM

- Shorthand syntax to create Lambda, API, Database, Event Source Mappings, Layers
  - Can write in plain CloudFormation but will be much simpler in SAM
  - Converted into CloudFormation during deployment

# What is SAM

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: Test Pipeline Lambda
Resources:
  samfunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: lambda_function.lambda_handler
      Runtime: python3.7
      CodeUri: .
      Events:
        MyTimeApi:
          Type: Api
          Properties:
            Path: /Codepipeline
            Method: GET
  Description: ''
```

## Creates

- Lambda
- API in API Gateway
- Put the Lambda as backend for /GET

Plain CloudFormation will be much larger

# What is SAM

- Shorthand syntax to create Lambda, API, Database, Event Source Mappings
  - Can write in plain CloudFormation but will be much simpler in SAM
  - Converted into CloudFormation during deployment
- Local Debugging and Testing
- Deep integration with dev tool - AWS and External
  - IDEs, Jenkins, Stackery toolkit etc.

# AWS SAM Template Concepts

- Declaring Serverless Resources
  - AWS::Serverless::Function
  - AWS::Serverless::API
  - AWS::Serverless::Application
  - AWS::Serverless::LayerVersion
  - AWS::Serverless::SimpleTable

# DEMO

- Install SAM CLI
  - Cloud9 Terminal
  - Local Desktop - Favorite IDE

*AWS Cloud9 is my IDE of choice for rest of the demos*

# DEMO

Author and Deploy SAM

- ❖ Lambda with no External Dependencies

# DEMO

Author and Deploy SAM

- ❖ Lambda with no External Dependencies

# DEMO

## Lambda with External Dependencies

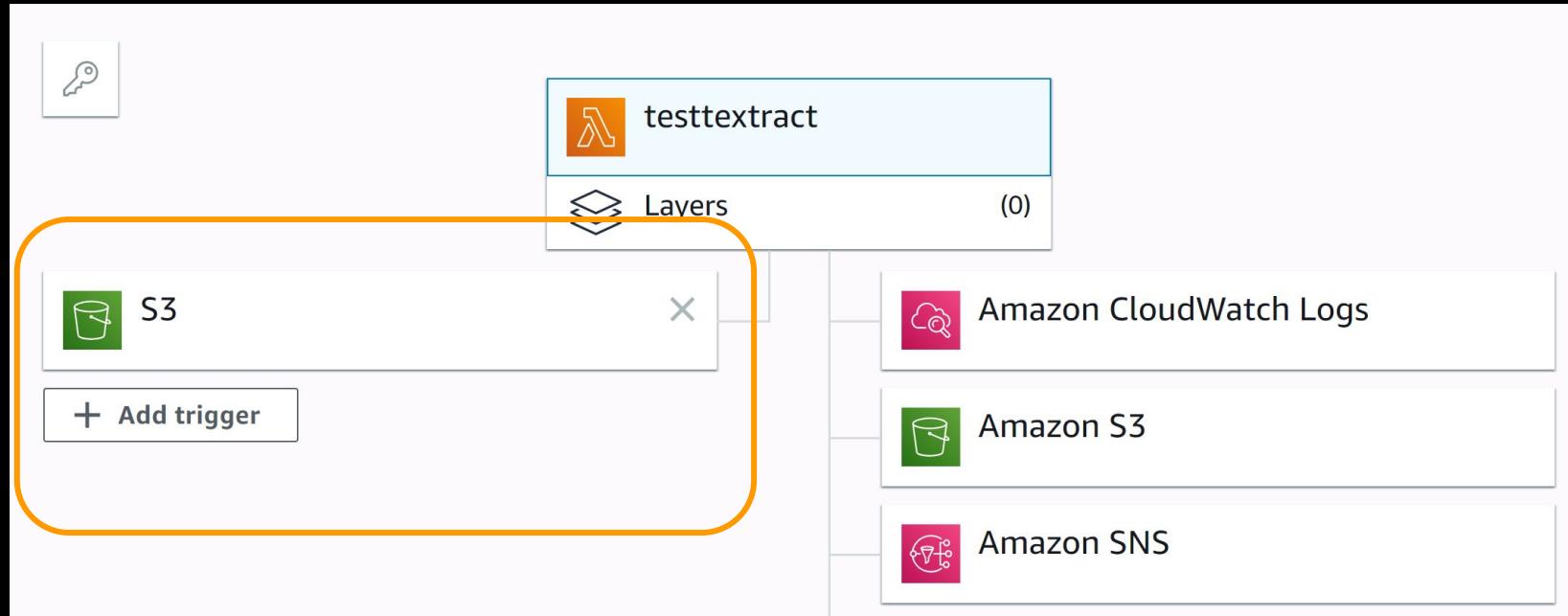
- ❖ Local Testing with SAM
- ❖ Deploy with SAM

# SAM Template

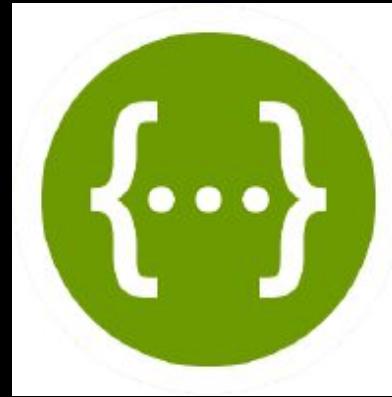
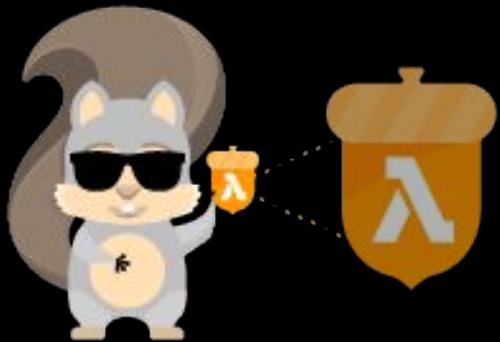
- Mix with regular CloudFormation in same template
  - E.g. VPC, Kinesis, ELB etc.
- Supports use of parameters, mappings, outputs, importvalue etc.
- Supports intrinsic functions
- YAML or JSON

# Lambda Events in SAM

- Events that trigger Lambda
  - E.g. S3 upload, API, Scheduled etc.



# SAM & Swagger



# Defining APIs with SAM

```
AWS::TemplateFormatVersion: '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: nodejs6.10
```

```
Policies: AmazonDynamoDBReadOnlyAccess
```

```
Events:
```

```
GetHtml:
```

```
Type: Api
```



API Endpoint triggering Lambda Function

```
Properties:
```

```
Path: /{proxy+}
```

```
Method: ANY
```

# Using Swagger for complex APIs

GetHtml:

Type: AWS::Serverless::Api

Properties:

StageName: prod

DefinitionUri: swagger.yml

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: nodejs6.10

Policies: AmazonDynamoDBReadOnlyAccess

Events:

GetHtml:

Type: Api

Properties:

RestApId: !Ref GetHtml

# DEMO

- Defining API with SAM and Swagger

# DEMO

- Swagger with already created Lambda

# Serverless Frameworks

# Serverless Frameworks

- Why frameworks?
- Different frameworks
  - Yes, there are many!
- Thinking Cap ON - should I adopt third party serverless Frameworks?
- Detailed Tutorial on frameworks

# Serverless Lifecycle

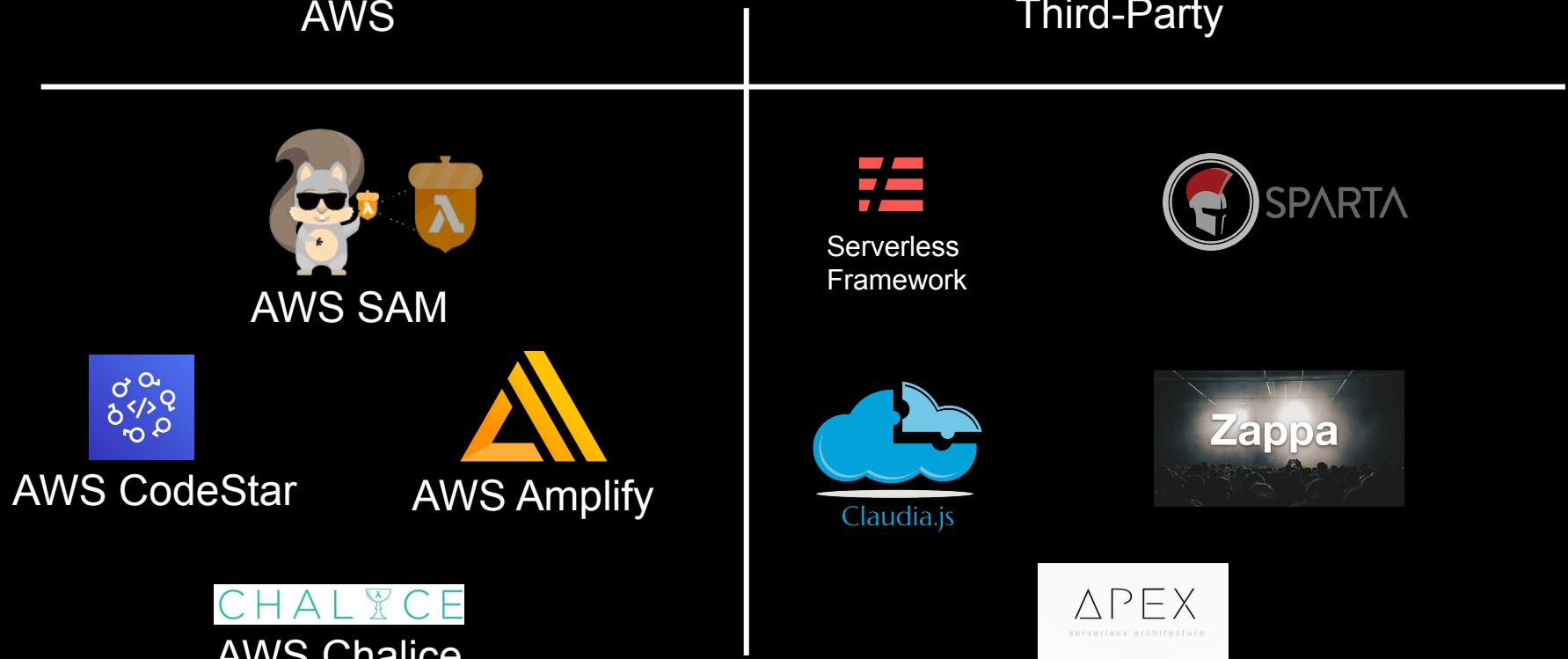
Couple Years Back

- Create Lambda and APIs using lines and lines of CloudFormation
- Create CI/CD pipelines by stitching multiple pieces
- Go multiple places to monitor and troubleshoot

# Serverless Frameworks

- Simplifies development, deployment and monitoring
- Introduce high level of abstraction
  - Few lines can spin up serverless infrastructure
- Single pane to monitor and troubleshoot

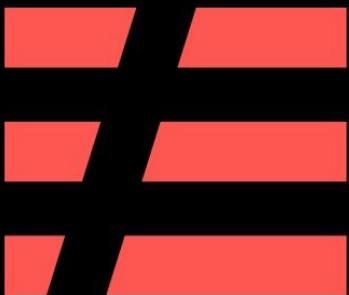
# Different Serverless Frameworks



# Considerations for 3rd Party Frameworks

- Each Enterprise is different
  - Low abstraction vs High abstraction
- License cost down the line?
- Cloud Agnostic approach
- Openness to Open-Source
- How fast do you adopt new AWS services and features
  - 3rd party tools will take some time to integrate new features

# The Serverless Framework



# The Serverless Framework

- Started in 2015 as open source project
- Supports multi-cloud
- Single pane develop, deploy and monitoring
  - Diving Deeper with example
- Thousands of plugins available
- Have open source and pro edition
  - This course will cover both



# AWS SAM



VS

# The Serverless Framework



## AWS SAM



## The Serverless Framework



Template

Highly abstracted template

AWS Integration

Integrates with native AWS tooling - CodeBuild, CodeDeploy, CodePipeline etc.

Dashboard

No dashboard, need to collect data from various services

Additional Code

No additional SDK needed

Multicloud

No MultiCloud capability

Looking ahead

Will evolve, example AWS CDK

Highly abstracted template

Out of box integration not available, some can be done through plugins

Single pane dashboard makes life easier

Serverless SDK required, makes the codebase larger and layered

MultiCloud capability

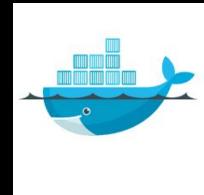
Always play catch up

# SERVERLESS



VS

# CONTAINER



# Let's Start from Beginning

## What is Serverless

- No servers to provision or manage
- Automatically scales with usage
- Never pay for idle
- Highly available

## Serverless Services



Amazon DynamoDB



Amazon API Gateway



AWS Step Functions



Amazon Simple Queue Service

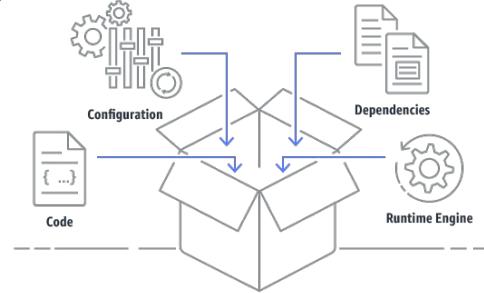


AWS Lambda

A compute service that lets you run code without provisioning or managing servers

## What is Container

Standard unit of software that packages up code and all its dependencies



The application runs quickly and reliably from one computing environment to another

## Container Orchestrators



Kubernetes (K8)



Amazon EKS



Amazon ECS



Docker Swarm

# Environment Difference



## Serverless

Underlying infrastructure managed by Cloud Provider

- Scales automatically
- No Patching headache

Can't install software (e.g. Webserver, Appserver) in underlying environment

- Code libraries can be installed

Easy selection of compute power

- 128 MB to 3 GB memory
- 1 sec to 15 Minutes time limit

No attached hard disk, deployment package size limited

**Superpower - Easier to onboard, focus on solving business problem from get go**



## Container

Users control underlying infrastructure - VM Size, OS, AMI etc.

- Requires management and orchestration
- Need to make master node HA, handle VM failover, AMI rehydration etc.

Install almost any software

- Prepackaged images with different softwares available

Adjustment of VM parameters requires some work

- Think of it as changing EC2 instance type on a running instance

Hard Disks attached to nodes

**Superpower - Complete control of environment, rich ecosystem**

# Use Case Difference



## Serverless

Shines at event driven architectures

- Native integration with other services
- Example - Triggered by S3, Kinesis

Suited when traffic is unpredictable

- Autoscaling
- Pay as you go

Microservices

- API Gateway integration
- Code is modular without software dependencies, e.g - python APIs
- Easier to migrate Cloud native, green fields apps
- Consider VPC Cold Start Latency

**Kryptonite - For brown field monoliths to Lambda, major refactoring needed**



## Container

Faster migration to cloud with other softwares

- Webserver, Appserver
- App requires third party software

Suited when traffic is predictable

- You pay for the underlying VM regardless
- Scales an entire VM

Microservices!

- Easy to move API with dependencies, e.g. Spring Boot with Discovery layer
- Consider cost and complexity for green field

**Kryptonite - Steep learning curve with multitudes of choices along with significant Day 2 operational overhead**

# Scaling of Lambda Vs Container



**Serverless**

Traffic to Lambda



Pay for what you use



**Container**

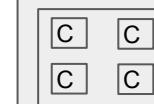


Amazon EKS

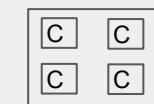
Traffic  
Traffic  
Traffic

K8 Master

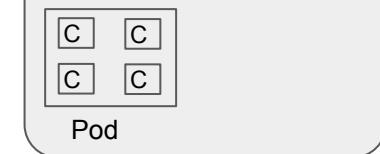
Basically bunch of  
EC2s



K8 Node  
(EC2)



Pod



Pod

[C] = Container

Node is at 50% Utilization  
Charged for entire EC2  
Pay for idle resources

**Wait before you make up your mind!**

# Money Matters



## Serverless

3 million/month, 512 MB Memory, 300 ms execution time,  
unpredictable traffic

- **\$8/month**

90 million/month, 512 MB memory, 250 ms execution time,  
predictable traffic

- **\$206/month**



## Container

Amazon EKS

3 million/month, 512 MB Memory, 300 ms execution time

- \$144/month (Control Plane) + \$14.4 (t3.small worker node) = **\$160/month**
- Cost will increase during higher spike coz of scaling

90 million/month, 512 MB memory, 300 ms execution time,  
predictable traffic

- \$144/month (Control Plane) + \$28.8 (t3.medium worker node) = **\$173/month**
- Predictable traffic makes it possible to select proper VM, higher CPU utilization

One is NOT cheaper or pricier than the other, it all depends on the use case

Parting Words

# SERVERLESS



&

# CONTAINER



“We don’t believe in one tool to rule the world. We want you to use the right tool for the right job.”— Andy Jassy, CEO of AWS

# Fargate - What & Why

- Both ECS and EKS requires managing Cluster and/or some infrastructure
- What if you just want to run your container
- Serverless version of Container - Fargate
- No need to create cluster or determine EC2 size, Fargate scales on-demand
- Pay for what you use
- One size does NOT fit ALL - Can be cheaper or pricier than ECS/EKS based on usage

# What is Docker/Container?

- Docker packages software into standardized units called containers that have everything your software needs to run including libraries, code and runtime
- Lets you quickly deploy and scale applications into any environment



# What is Container Orchestrator?



# How Does Docker Work?

Insert Video here

Draw a pentagon representations of 2 apps, color it different, then say it needs a host to run, so we spin up EC2s, move the pentagons inside EC2.

EC2s are like hyenas, if you see one, other ones are nearby.

To make it Highly Available, you need another in Az

Then comes scaling, put it in ASG, to route traffic you need Load Balancer.

If one task fails then u need to spin up

# Tasks Associated with Containers

- Deployment of Containers
- Redundancy and availability of Containers
- Scaling up or down of Containers
- Load Balancing
- Health Monitoring of Containers and Hosts
- Service Discovery
- And More...

# Container Orchestrator



# Say Hello to Container Orchestrators

- Docker Swarm



- Apache Mesos



- Cattle, Nomad, Empire

- AWS ECS (Elastic Container Service)



- Kubernetes



- EKS (Elastic Container Service for Kubernetes)



- AWS Fargate



# Battle of Orchestrators

So it begins...

# Control Plane - ECS & Kubernetes

Control Plane - Main entry point of Orchestrator. Interface to launch an application, query the state or shut down

## ECS

- Fully managed, highly available, highly scalable Control Plane
- You do NOT pay for Control Plane

## Kubernetes

- Fully Open-Source, created by Google, run on any cloud or on-prem
- Run your own Control Plane on Host (such as EC2)
  - You need to take care of selecting EC2 and scaling etc.
- AWS can manage Kubernetes Control Plane for you - **EKS**
- Pay for the Control Plane - either for underlying EC2 or AWS Managed EKS Control Plane

# Why Fargate?

- Both ECS and EKS requires managing Cluster and/or some infrastructure
- What if you just want to run your container
- Serverless version of Container - Fargate
- No need to create cluster or determine EC2 size, Fargate scales on-demand
- Pay for what you use
- One size does NOT fit ALL - Can be cheaper or pricier than ECS/EKS based on usage

# Fargate - What & Why

- Both ECS and EKS requires managing Cluster and/or some infrastructure
- What if you just want to run your container
- Serverless version of Container - Fargate
- No need to create cluster or determine EC2 size, Fargate scales on-demand
- Pay for what you use
- One size does NOT fit ALL - Can be cheaper or pricier than ECS/EKS based on usage

# Money Matters

	ECS	EKS	Fargate
1 task cost per month			
Control Plane	\$0.00	\$144.00	\$0.00
EC2 worker node (m5.large)	\$70.28	\$70.28	\$0.00
Task (0.5 vCPU, 1 GB Mem)	\$0.00	\$0.00	\$27.36 (\$0.0506 vCPU/hr X 0.5 vCPU X 24 hr X 30 Days + \$0.0127 GB/hr X 1 GB X 24 hr X 30 Days)
Total	\$70.28	\$214.28	\$27.36
24 tasks cost per month			
Control Plane	\$0.00	\$144.00	\$0.00
EC2 worker node (m5.large X 4)	\$421.68 (4 EC2s = \$70.28 X 4 )	\$421.68 (4 EC2s = \$70.28 X 4 )	\$0.00
Task (0.5 vCPU, 1 GB Mem)	\$0.00	\$0.00	\$656.64 (\$27.36 X 24)
Total	\$421.68	\$565.68	\$656.64

# The Good Old Comparison Chart

ECS



Container Orchestration, created by AWS

Requires creating cluster

Control Plane Costs Zero, pay for worker nodes

Deeper integration with other AWS Services such as IAM, ALB etc.

Good for cloud native container architectures

EKS



Managed Kubernetes (Open Source) platform by AWS

Requires creating cluster

Control Plane Costs 144\$ (Dec 2018), pay for worker nodes

AWS actively working on the integrations

Good for cloud native container architectures, easier to move on-prem Kubernetes to AWS EKS

Fargate



Containers on-demand

No cluster is required

Only pay for tasks based on CPU and Memory

Fargate currently runs on ECS, early days, more to come

Good for workload which runs for a duration, Fargate is expensive if high CPU/Memory tasks runs all the time

# Serverless Architectures & Advanced Optimization Techniques

# Optimizing Your Lambda



# Couple Words Before We Begin

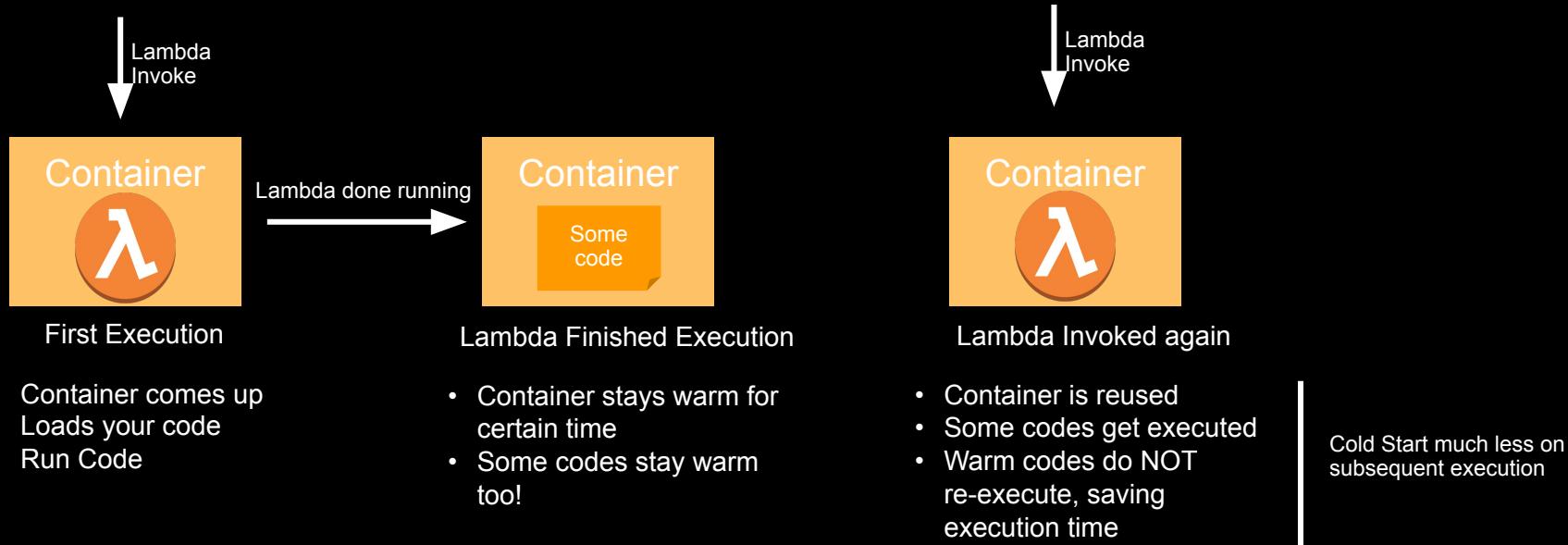
- Battle hardened real world tips
- Advanced lecture (rewatch parts if needed)

# Places Where You can Optimize

- Your Function
  - Actual Code
- Execution Environment
  - The execution settings of the Lambda

# Optimizing Your Function Code

# Lambda Code Execution Under The Hood



What Codes Stay Warm?

# A Typical Lambda function

## Handler() function

Function to be executed upon invocation

## Event object

Data sent during Lambda function Invocation

## Context object

Methods available to interact with runtime information (request ID, log group, etc.)

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello World!')
    }
```

# Lambda Code Execution Under The Hood

```
lambda_function x +  
1 import boto3  
2 import os  
3 import datetime  
4  
5 textract = boto3.client('textract')  
6 sns = boto3.client('sns')  
7 s3client = boto3.client('s3')  
8  
9 badimagesns=os.environ['bad_image_sns']  
10 circuitname=os.environ['circuit_name']  
11 s3resultbucket=os.environ['s3_result_bucket']  
12  
13 def lambda_handler(event, context):  
14     #print(event)  
15     for item in event['Records']:  
16         s3SourceBucketName=item['s3']['bucket']['name']  
17         documentName=item['s3']['object']['key']  
18  
19         connect_to_database()  
20         execute_business_logic()  
21
```

Executed during first execution and stay warm

Reruns during each execution

# Lambda Code Execution Under The Hood

```
lambda_function x +  
1 import boto3  
2 import os  
3 import datetime  
4  
5 textract = boto3.client('textract')  
6 sns = boto3.client('sns')  
7 s3client = boto3.client('s3')  
8  
9 badimagesns=os.environ['bad_image_sns']  
10 circuitname=os.environ['circuit_name']  
11 s3resultbucket=os.environ['s3_result_bucket']  
12  
13 def lambda_handler(event, context):  
14     #print(event)  
15     for item in event['Records']:  
16         s3SourceBucketName=item['s3']['bucket']['name']  
17         documentName=item['s3']['object']['key']  
18  
19         connect_to_database() ←  
20         execute_business_logic()  
21
```

Is this function executing every time?

If Yes and expensive, move to global scope

# Lambda Code Execution Under The Hood

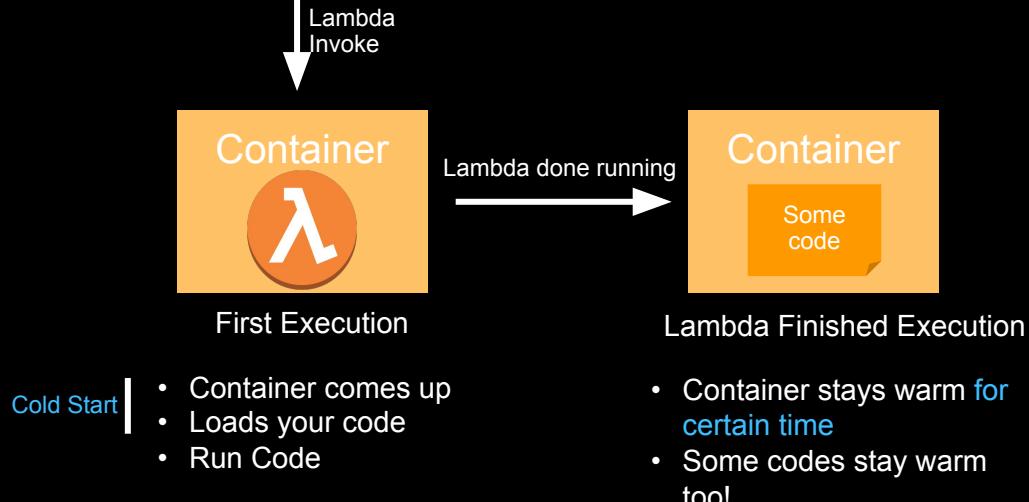
lambda\_function x +

```
1 import boto3
2 import os
3 import datetime
4
5 textract = boto3.client('textract')
6 sns = boto3.client('sns')
7 s3client = boto3.client('s3')
8
9 badimagesns=os.environ['bad_image_sns']
10 circuitname=os.environ['circuit_name']
11 s3resultbucket=os.environ['s3_result_bucket']
12
13 connect_to_database() ←
14
15 def lambda_handler(event, context):
16     #print(event)
17     for item in event['Records']:
18         s3SourceBucketName=item['s3']['bucket']['name']
19         documentName=item['s3']['object']['key']
20
21
22     execute_business_logic()
```

Moved to global scope

connect\_to\_database() will stay warm through out subsequent execution

# So Why Not Put EVRYTHING in Global Scope?



- The more code you put on global scope, cold start on first execution will be longer
- Container does go down if lambda not invoked subsequently for certain time
- It's a balance (will go over tools to help)

# Anti-Pattern

lambda\_function 

```
import json
import account_validation as validationlayer
import http-lib

sendRegularoffer()

def lambda_handler(event, context):
    if validationlayer.validateAcct(event['AcctNo']) == 'PASS':
        #lots of logic here but only 5% qualifies
        import superspecial-library
        sendSuperSpecialOffer()
    else:
        sendRegularoffer()
```

This does not  
save anything

# Concise Function Logic

```
lambda_function x +  
1 import boto3  
2 import os  
3 import datetime  
4 import kitchen-sink  
5 from http import request  
6  
7 textract = boto3.client('textract')  
8 sns = boto3.client('sns')  
9 s3client = boto3.client('s3')  
10  
11 badimagesns=os.environ['bad_image_sns']  
12 circuitname=os.environ['circuit_name']  
13 s3resultbucket=os.environ['s3_result_bucket']  
14  
15 connect_to_database()  
16  
17 def lambda_handler(event, context):  
18     #print(event)  
19     for item in event['Records']:  
20         s3SourceBucketName=item['s3']['bucket']['name']  
21         documentName=item['s3']['object']['key']  
22
```

Don't load something that you don't need

Import required libraries from package rather than loading the entire package

# Lazy Loading

lambda\_function ×

```
import json
import account_validation as validationlayer
import http-lib

sendRegularOffer()

def lambda_handler(event, context):
    if validationlayer.validateAcct(event['AcctNo']) == 'PASS':
        # lots of logic here but only 5% qualifies
        import superspecial-library
        sendSuperSpecialOffer()
    else:
        sendRegularOffer()
```



Libraries can be  
lazy loaded too!

# Lambda Environment Variables

- Key-value pairs that you can dynamically pass to your function without making code changes
- Available via standard environment variable APIs
- Can be encrypted via AWS Key Management Service (AWS KMS)
- Useful for creating environments per stage (i.e., dev, testing, production)

# Lambda Environment Variables

```
import boto3
import os
import datetime

textract = boto3.client('textract')
sns = boto3.client('sns')
s3client = boto3.client('s3')

badimagesns=os.environ['bad_image_sns']
circuitname=os.environ['circuit_name']
s3resultbucket=os.environ['s3_result_bucket']

def lambda_handler(event, context):
    print(event)
    for item in event['Records']:
```

## 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](#)

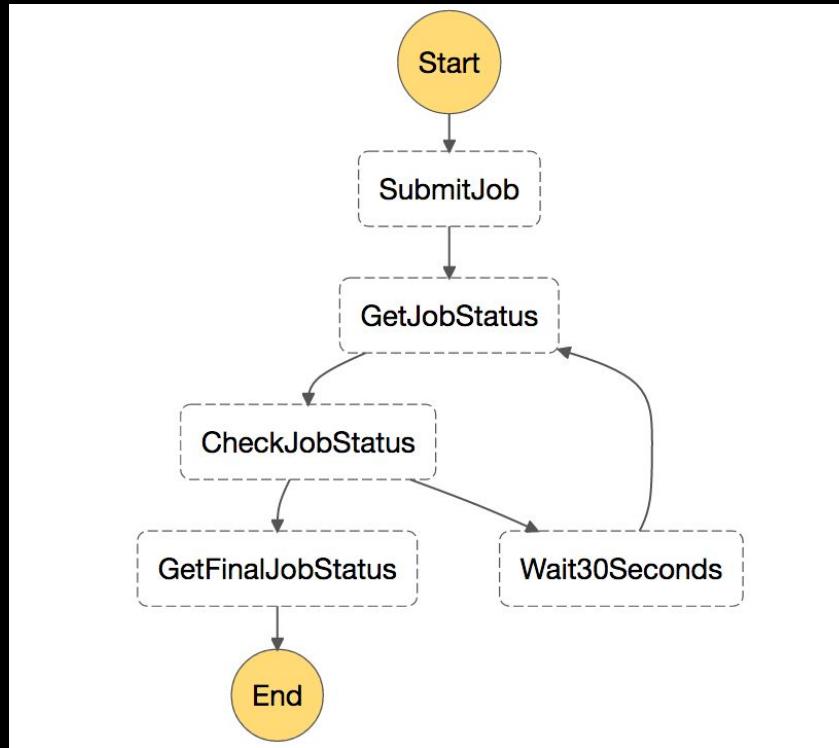
bad_image_sns	arn:aws:sns:us-east-1:453986252919:drcft1252-Ba	<button>Remove</button>
circuit_name	Kumo Torakku	<button>Remove</button>
s3_result_bucket	drcft1252-s3bucketforresult-bc9x5qr6cx9y	<button>Remove</button>
Key	Value	<button>Remove</button>

▶ [Encryption configuration](#)

# Concise function logic

- Separate Lambda handler (entry point) from core logic
  - Use Lambda Layers for duplicated logic
- Use functions to **TRANSFORM**, not **TRANSPORT**
  - API directly integrate with AWS Services
  - SNS > Lambda > S3? Do SNS > S3
- Dynamic logic via configuration
  - Per function – Environment variables
  - Cross function – Amazon Parameter Store/Secrets Manager
- Read only what you need. For example:
  - Properly indexed databases
  - Use Views instead of compute
  - Use Amazon S3 Select

# Keep Orchestration out of the code



# Optimizing Lambda Code Recap

- Use pre-handler logic strategically
- Think about how re-use impacts variables, connections, and dependency usage
- Minimize dependencies
- Share secrets based on application scope:
  - Single function: Env-Vars
  - Multi Function/shared environment: Secrets Manager
- Concise logic
- Push orchestration up to Step Functions
- Use Lambda Layers to simplify deploys

# Optimizing Execution Environment

A photograph of a woman with dark hair, wearing a black t-shirt and blue jeans, working on a car engine in a garage. She is holding a wrench and looking directly at the camera. The garage is filled with various tools and equipment, including a workbench labeled "SATA 95111". In the background, there are shelves with boxes and containers. The lighting is dramatic, with strong highlights and shadows.

# HOW MUCH MEMORY AND TIME NEEDED FOR LAMBDA???

## Memory (MB) [Info](#)

Your function is allocated CPU proportional to the memory configured.



## Timeout [Info](#)

7 min 45 sec

# Knowing is Half The Battle

- “In God we trust, all others bring Data” - Adam Breckler
- X-Ray gives you the data, and areas to optimize
- X-Ray Integrates with Lambda and API-Gateway

AWS X-Ray [Info](#)

Enable active tracing to record timing and error information for a subset of invocations.

Active tracing

[View traces in X-Ray](#)

Logs/Tracing Stage Variables SDK Generation Export Deployment History Documentation History Canary

Configure logging and tracing settings for the stage.

CloudWatch Settings

Enable CloudWatch Logs [?](#)

Enable Detailed CloudWatch Metrics [?](#)

Custom Access Logging

Enable Access Logging [?](#)

X-Ray Tracing [Learn more](#)

Enable X-Ray Tracing [?](#) Set X-Ray Sampling Rules

# X-Ray Trace Example

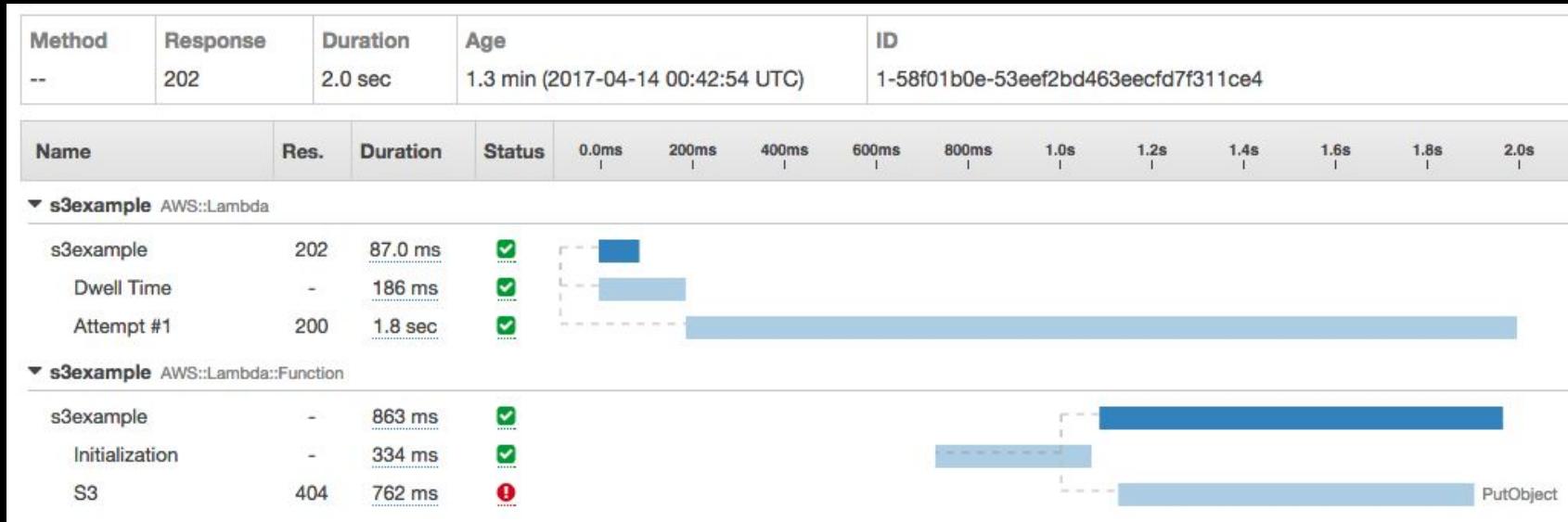


Image: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html>

# Seeing a cold start in AWS X-Ray

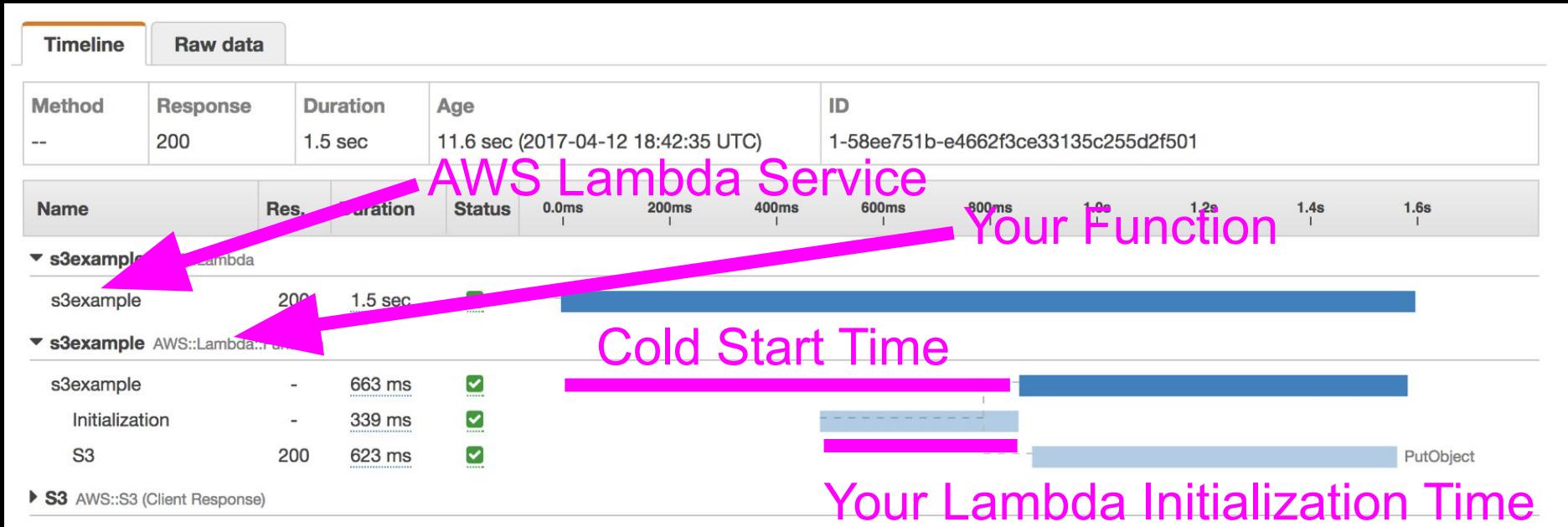


Image: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html>

# Third Party Tools

Lambda performance already available from:

- Datadog
- Epsagon
- NodeSource
- IOPipe
- Thundra

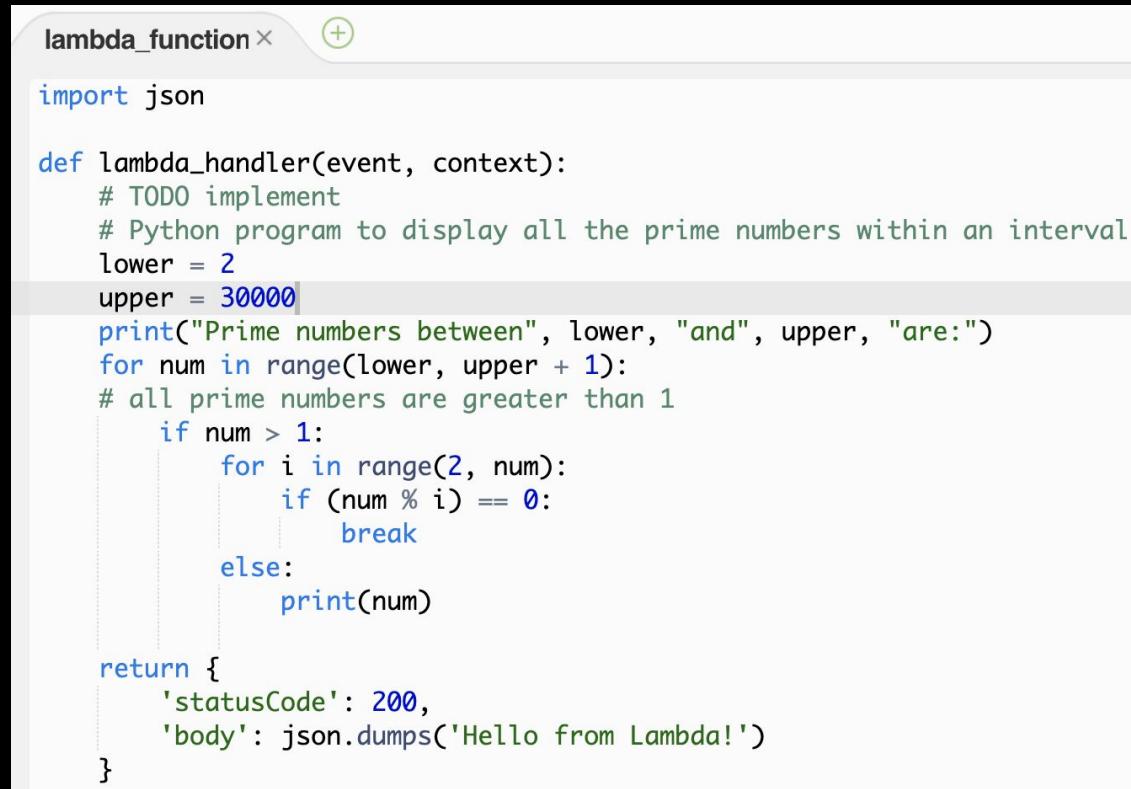
# Lambda Memory and Compute Goes Together



Lambda exposes only a memory control, with CPU allocated to a function proportionally

# Our Very Own Test

Stats for Lambda function that calculates all prime numbers <= 30000 & 3000 times a month



The image shows a screenshot of a code editor with a tab titled "lambda\_function". The code is written in Python and defines a Lambda function handler. The function prints prime numbers between two specified values and returns a JSON response.

```
lambda_function × +  
import json  
  
def lambda_handler(event, context):  
    # TODO implement  
    # Python program to display all the prime numbers within an interval  
    lower = 2  
    upper = 30000  
    print("Prime numbers between", lower, "and", upper, "are:")  
    for num in range(lower, upper + 1):  
        # all prime numbers are greater than 1  
        if num > 1:  
            for i in range(2, num):  
                if (num % i) == 0:  
                    break  
                else:  
                    print(num)  
  
    return {  
        'statusCode': 200,  
        'body': json.dumps('Hello from Lambda!')  
    }
```

# Execution Time

Stats for Lambda function that calculates all prime numbers <= 30000 & 3000 times a month

**128 MB**      56.13735sec

**256 MB**      26.43806sec

**512 MB**      12.68250sec

**1024 MB**      6.28947sec

**Green**==Best

**Red**==Worst

## AWS Lambda Pricing Calculator

**Number of Executions**

3000

Enter the number of times your Lambda function will be called per month

**Allocated Memory (MB)**

128



Enter the allocated memory for your function

**Estimated Execution Time (ms)**

56137.35

Enter how long you expect the average execution will take in milliseconds

**Include Free Tier**

Yes  No

**TOTAL COSTS**

Request Costs: \$0.00

Execution Costs: \$0.35

---

\$0.35/month

<https://s3.amazonaws.com/lambda-tools/pricing-calculator.html>

# Memory Vs Monthly Cost

Stats for Lambda function that calculates all prime numbers  $\leq 30000$  &  
**3000 times a month**

**128 MB**    56.13735sec    \$0.35

**256 MB**    26.43806sec    \$0.33

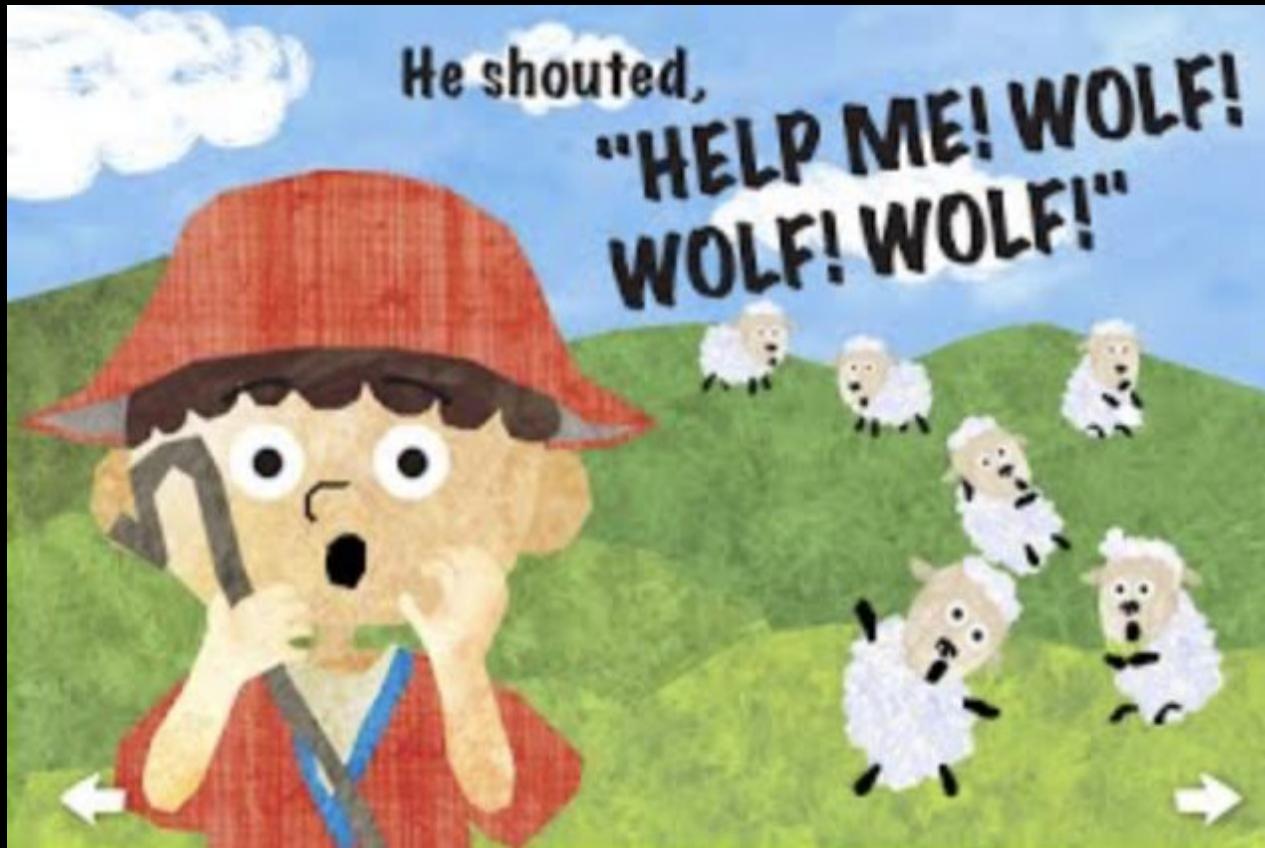
**512 MB**    12.68250sec    \$0.16

**1024 MB**    6.28947sec    \$0.08

**Green**==Best

**Red**==Worst

# Moral of The Story (Yes, there's always one!)



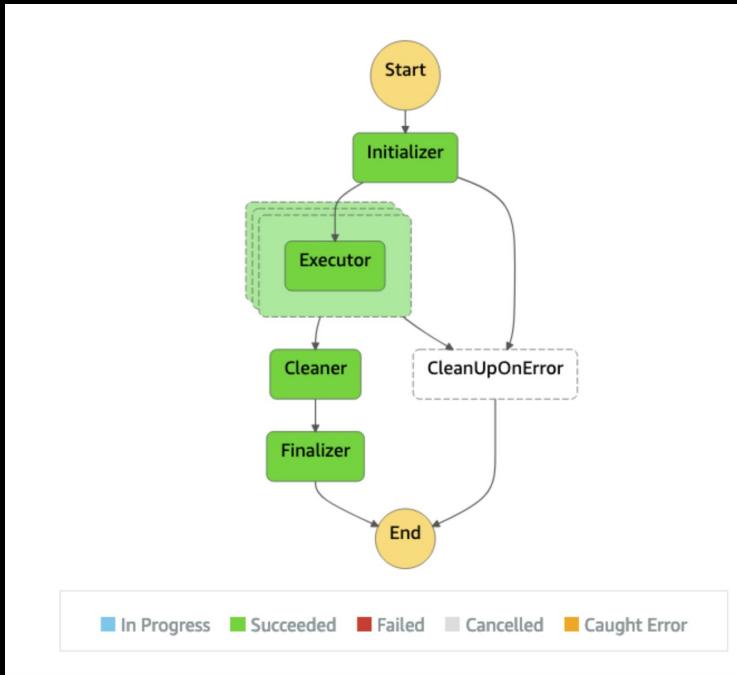
# Moral of The Story (No animals get killed in this one)

- Sometimes even when you crank up memory, cost comes down
- Sometimes with increased memory, cost might increase
  - If Performance increases significantly and cost increases slightly, worth it
  - You define what's acceptable for your case
- Worth testing out for time consuming Lambda

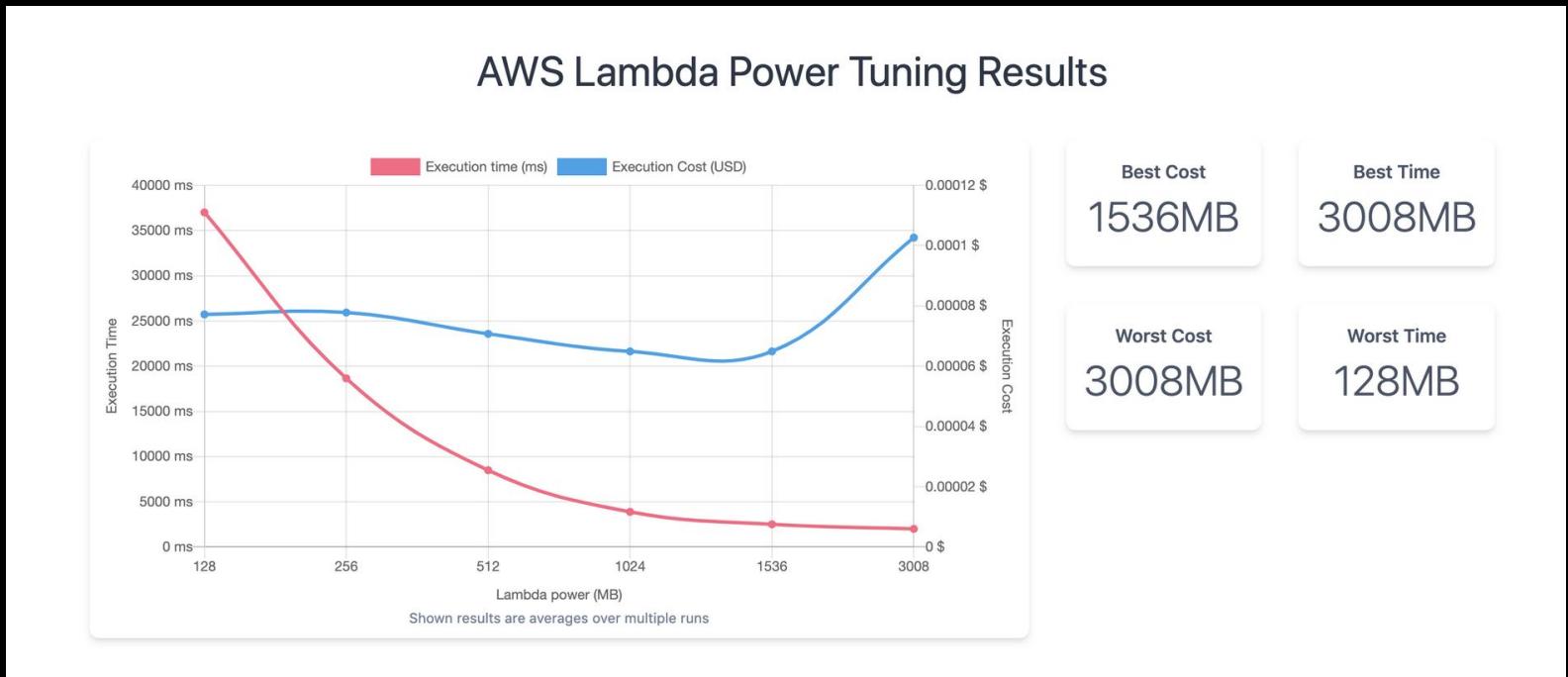
# Lambda Power Tuning

Project: <https://github.com/alexcasalboni/aws-lambda-power-tuning>

Analyze execution logs and suggest you the best configuration to minimize cost or maximize performance



# Lambda Power Tuning



# CloudWatch Insights

The screenshot shows the CloudWatch Insights interface. On the left, a sidebar lists various CloudWatch services: CloudWatch, Dashboards, Alarms (with 1 ALARM), INSUFFICIENT (0), OK (3), Billing, Events, Rules, Event Buses, Logs, Insights (selected), Metrics, Alpine, Settings, and Favorites. The 'Logs' section displays a chart with a Y-axis from 0 to 100 and an X-axis from 0 to 02 PM, with a single data point at 100 at 01:15. Below the chart is the text "Run a query to see the related events". Above the chart is a query editor with a dropdown for "Select log group(s)" and a time range selector (15m, 30m, 1h, 6h, 12h, 1d, custom). The main query text is:

```
fields @timestamp, @message  
| sort @timestamp desc  
| limit 20
```

Below the query editor are "Run query" and "Actions" buttons, and a "Sample queries" dropdown menu. The "Logs" tab is selected in the visualization section. The "Sample queries" menu is open, showing:

- Lambda queries > View latency statistics for 5-minute intervals
- VPC flow log queries > Determine the amount of overprovisioned memory
- CloudTrail queries > Find the most expensive requests
- Common queries >
- Route 53 queries >
- AWS AppSync queries >

On the right, a "Query help" sidebar lists "Commands": fields, filter, stats, sort. It also has a "Search for a field" input field and a note "No log group(s) selected".

# Do You Need Sync Everywhere?

**Separate Sync, Async components**

Example – GET Sync, POST Async

**Sending data to another system**

Do I need to insert realtime, is near realtime okay?

Do I need response in the same call?

API Vs topic/queue/stream

# Do I need to put my functions in an Amazon VPC?

Lambda Needs to Access resource in VPC  
Example – RDS in VPC

## Restrict Outbound Access to Internet

### Security groups

Choose the VPC security groups for Lambda to use to set up your VPC configuration. Format: "sg-id (sg-name) | name-tag". The table below shows the inbound and outbound rules for the security groups that you chose.



⚠ Choose at least 1 security group.

- i When you enable a VPC, your Lambda function loses default internet access. **If you require external internet access for your function, make sure that your security group allows outbound connections and that your VPC has a NAT gateway.**

# Lambda execution models

Synchronous



Amazon API Gateway

GET, POST



AWS Lambda  
function

WebSocket



Amazon API Gateway

Bidirectional



AWS Lambda  
function

Asynchronous



Amazon SNS



Amazon S3

Events



AWS Lambda  
function

Poll-based



Amazon  
Kinesis

changes



AWS Lambda  
service



function

# Do You Need Sync Everywhere?

**Separate Sync, Async components**

Example – GET Sync, POST Async

**Sending data to another system**

Do I need to insert realtime, is near realtime okay?

Do I need response in the same call?

API Vs topic/queue/stream

# Lambda Dead Letter Queues

“By default, a failed Lambda function invoked asynchronously is retried twice, and then the event is discarded.” –

<https://docs.aws.amazon.com/lambda/latest/dg/dlq.html>

- Turn this on! (for async use cases) - It's Free till it's used

## Error handling

### DLQ resource [Info](#)

Choose the AWS service to send the event payload to after maximum retries are exceeded.

None

## Error handling

None

Amazon SNS

Amazon SQS

None

# Do You Need APIs to Expose lambda?

Is Lambda only getting called from internal systems?

Maybe you don't need API

You can call Lambda like any other AWS Service

Lambda can be called from EC2, Lambda etc.

Like Lambda uses Boto3 to call other Service, Lambda can be called using Boto3 as well!

# Events - Don't Fire Lambda Unnecessarily

## Discard Unnecessary Events

S3 - Use Specific Prefix

SNS - Message Filtering

Amazon SNS > Subscriptions > Create subscription

### Create subscription

**Details**

Topic ARN

Protocol  
The type of endpoint to subscribe

Endpoint  
An AWS Lambda function that can receive notifications from Amazon SNS.

▼ **Subscription filter policy - optional**

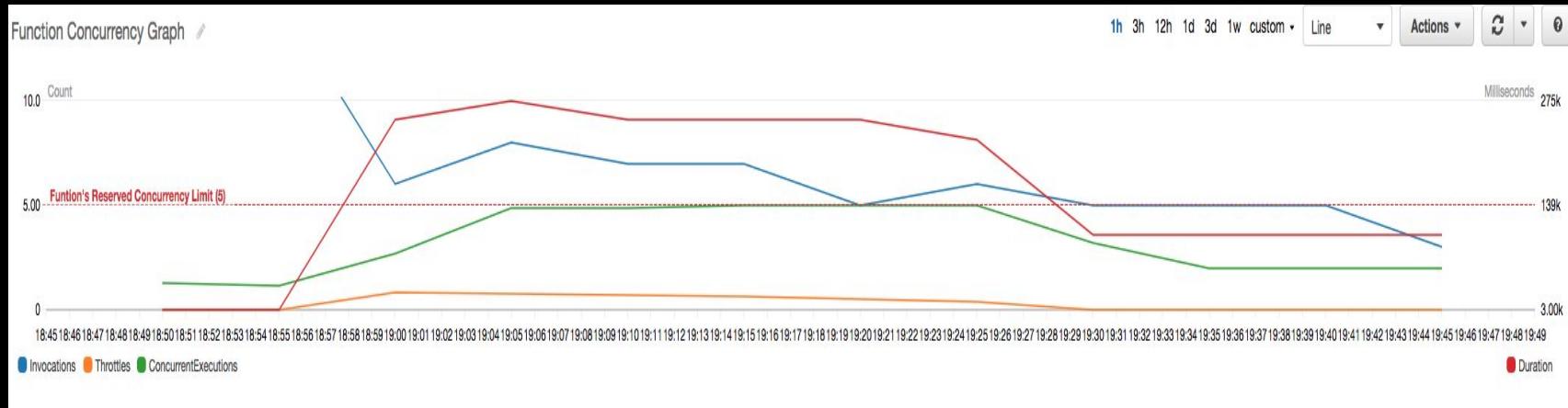
This policy filters the messages that a subscriber receives. [Info](#)

**JSON editor**

```
1 {
  "anyMandatoryKey": [
    "any",
    "of",
    "these"
  ],
  "anyOtherOptionalKey": [
    "any",
    "of",
    "these"
  ]
}
```

# Lambda Per Function Concurrency Controls

- Concurrency a shared pool by default
- Separate using per function concurrency settings
  - Acts as reservation
- Also acts as max concurrency per function
  - Especially critical for data sources like Amazon RDS
- “Kill switch” – set per function concurrency to zero



# Optimizing Execution Env

## Recap

- More memory == More CPU and I/O (proportionally)
  - Can also be lower cost
- Use AWS X-Ray to profile your workload
- Think deeply about your execution model and invocation source needs
  - Not everything needs to be an API
- Understand the various aspects to queues, topics, streams when using them
- VPC has certain benefits but isn't necessary for security

# AWS Well-Architected Framework - Five Pillars

- Operational excellence
- Security
- Reliability
- Performance efficiency
- Cost optimization

**Reference:** <https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf>

# Serverless - General Design Principles

- Speedy, simple, singular
- Share nothing
- Orchestrate your application with state machines
- Design for failures and duplicates
- Use events to trigger transactions

# Serverless - Design Components

## Compute Layer

- Manages requests from external systems
- Controlling access and ensuring requests are appropriately authorized
- Contains the runtime environment that your business logic will be deployed and executed

**Examples** - Lambda, API Gateway, Step Functions

## Data Layer

- Persistent storage
- Secure mechanism to store states that your business logic will need
- Provides a mechanism to trigger events in response to data changes

**Examples** - DynamoDB, S3, Aurora Serverless

# Serverless - Design Components

## Messaging and Streaming Layer

- Manages communications between components
- Manages real-time analysis and processing of streaming data

**Examples** - Kinesis, SNS

## User Management and Identity Layer

- provides identity, authentication, and authorization for both external and internal customers of your workload's interfaces

**Examples** - Cognito

## Systems Monitoring and Deployment

- system visibility through metrics
- defines how your workload changes are promoted

**Examples** - CloudWatch, X-Ray

# Serverless - Design Components

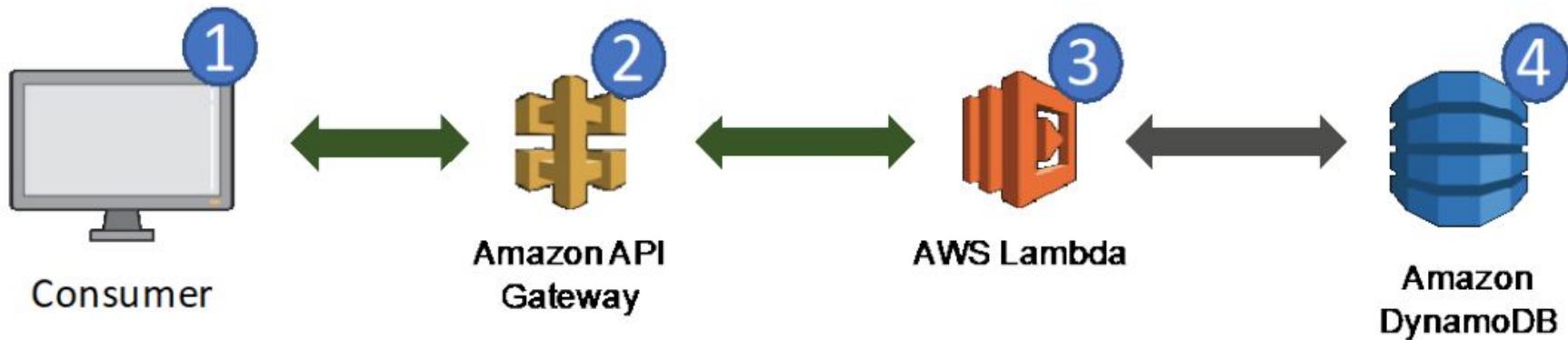
## Edge Layer

- Manages the presentation layer and connectivity to external customers
- Provides an efficient delivery method to external customers residing in distinct geographical locations

**Examples** - CloudFront

# Serverless - Design Scenarios

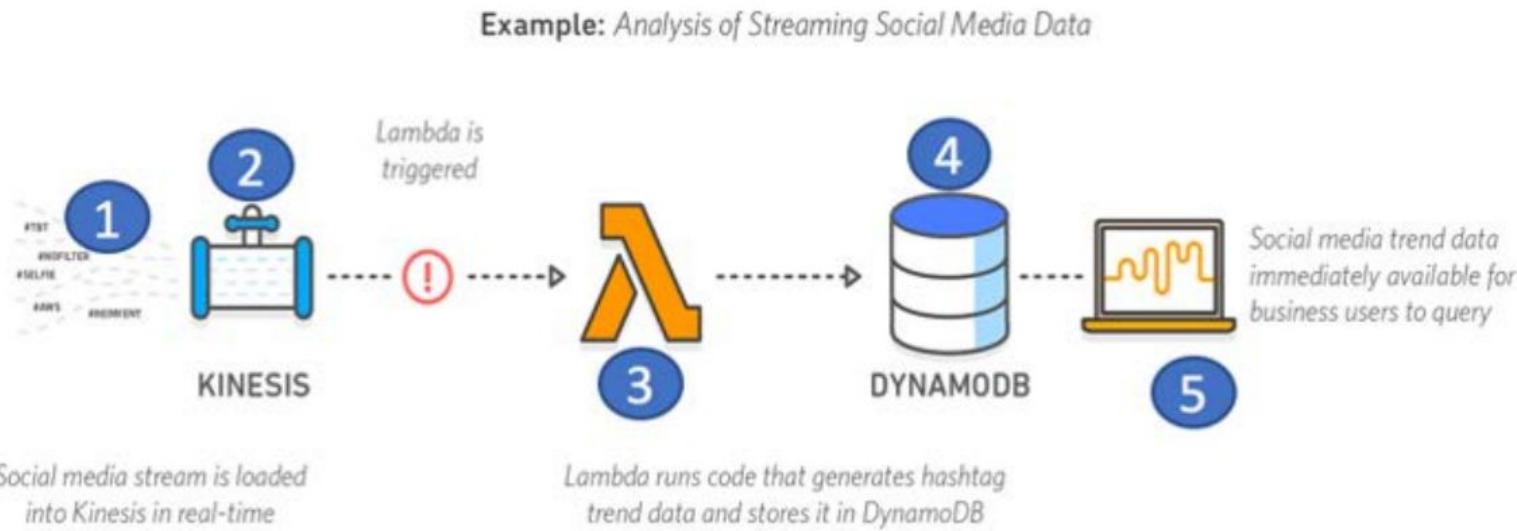
RESTful Microservices



**Figure 1: Reference architecture for RESTful microservices**

# Serverless - Design Scenarios

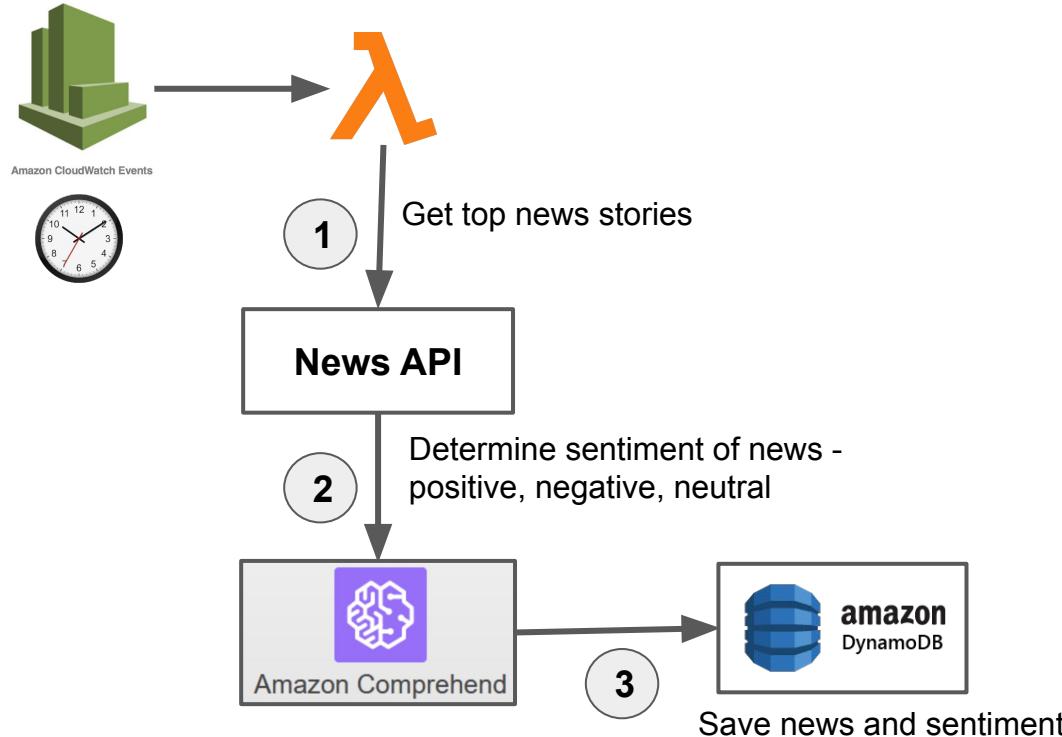
## Stream Processing



**Image from:** <https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf>

# Serverless - Design Scenarios

## Periodic Job



# Serverless - Design Scenarios

## Web Application

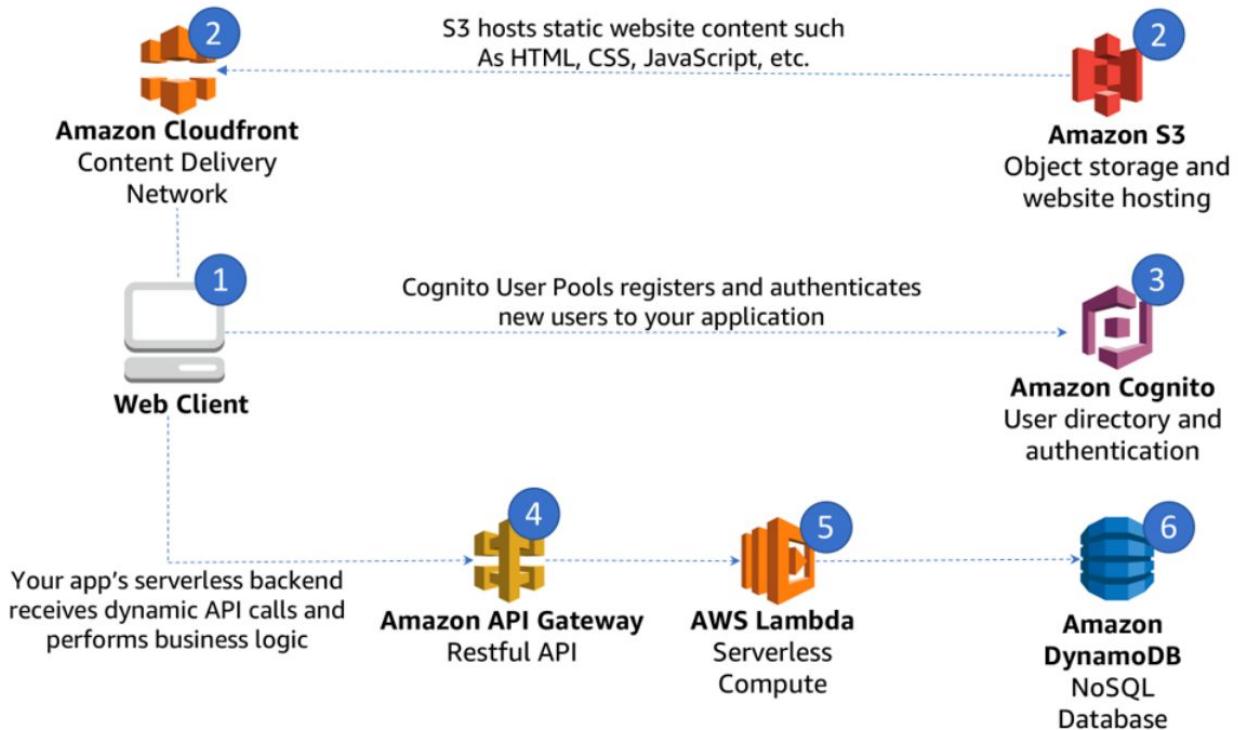


Image from: <https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf>

# When NOT to Use Lambda

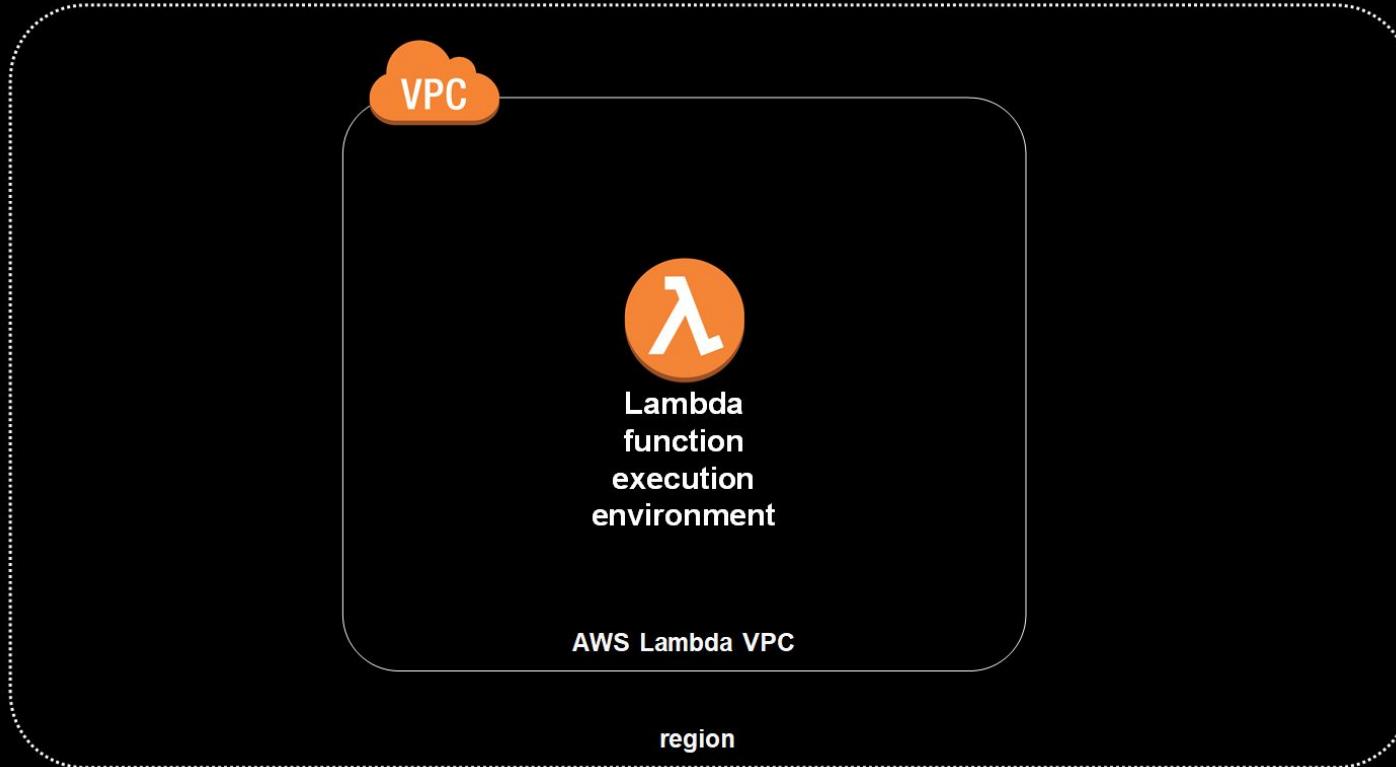
- Always compare cost with EC2
  - Price varies based on traffic, memory and time
- Gorilla in the room - Cold Start in VPC
  - Based on traffic pattern, cold starts of Lambda could occur
  - If Lambda in VPC, Cold Starts attaches ENI (Elastic Network Interface) to Lambda which takes seconds
  - Can try to keep lambda warm using CloudWatch pings and algorithms
  - For corner cases, ultra low SLA of APIs not achievable
- Super heavy computing exceeding Lambda Memory/Time limit

**Important:** Always weigh your options unbiasedly, Lambda is not suppose to be all end all for all design problems

# Lambda VPC ENI Cold Start - No More!



# Lambda VPC Networking

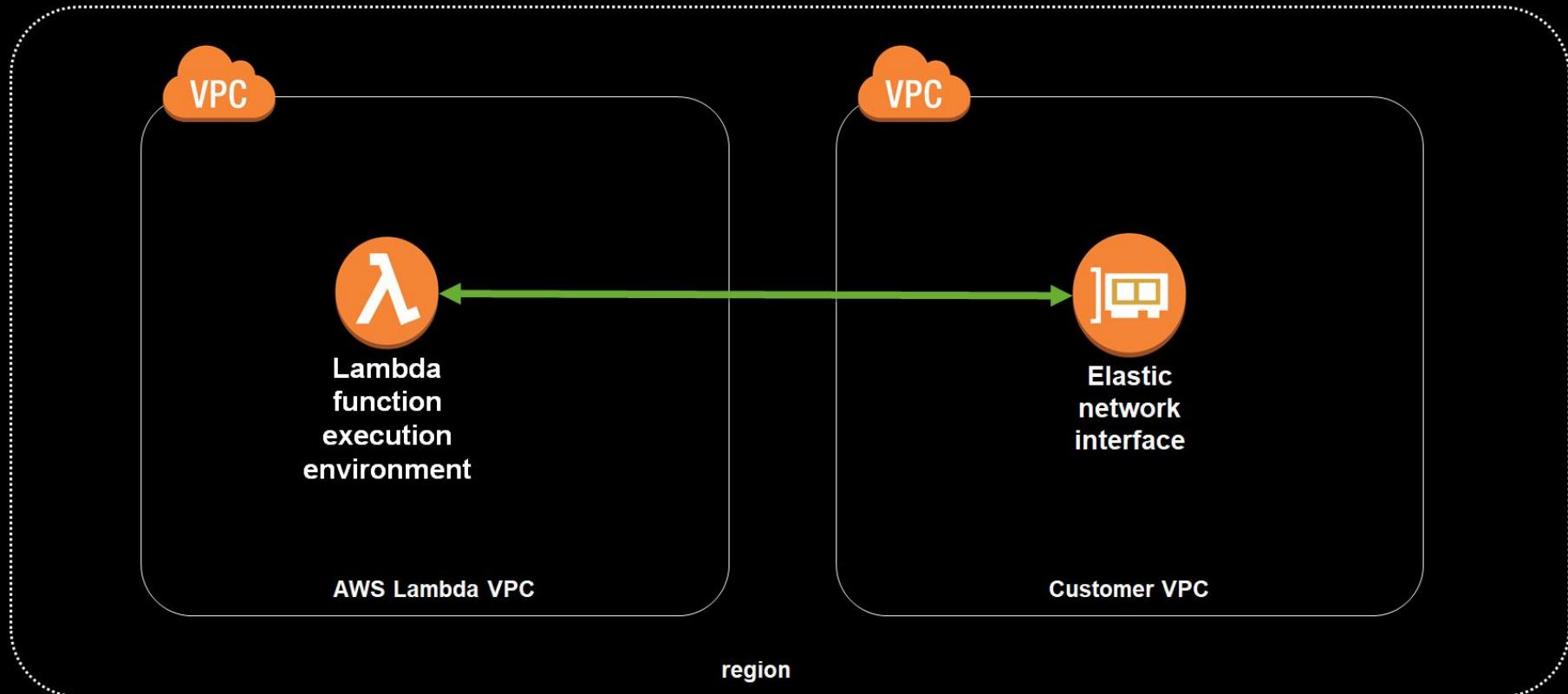


Invocations can  
only come in via  
the AWS Lambda  
API

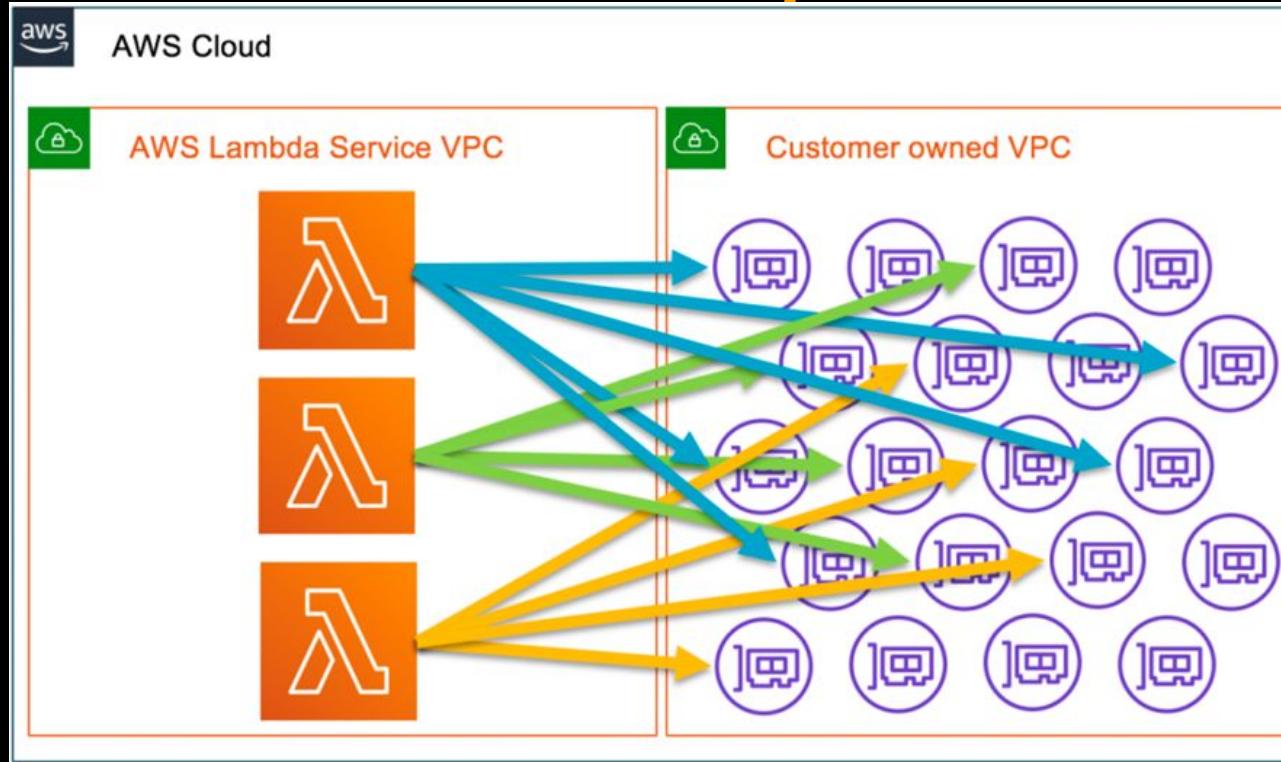


Region

# Lambda Networking with your VPC

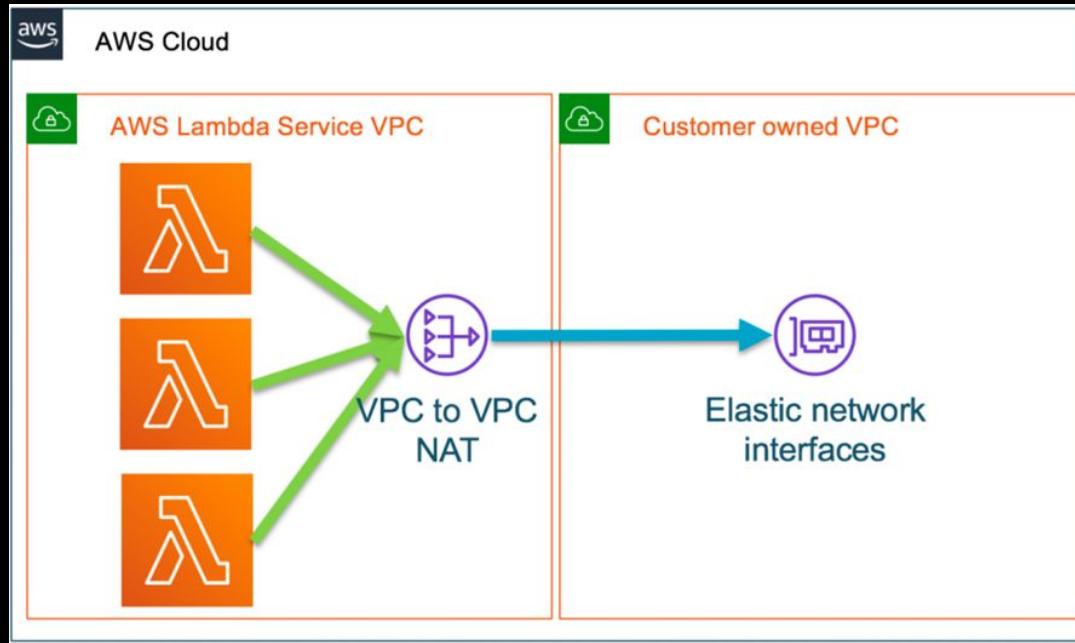


# Prior to VPC Improvements



**ENIs created and attached during function execution**  
**Concurrent Lambdas used to consume large amount of IP Addresses**

# VPC Improvements (Live Now)



ENIs attached during function creation, NOT execution  
Lambdas share an ENI (same subnet, security group combo)

# Lambda VPC Improvements

