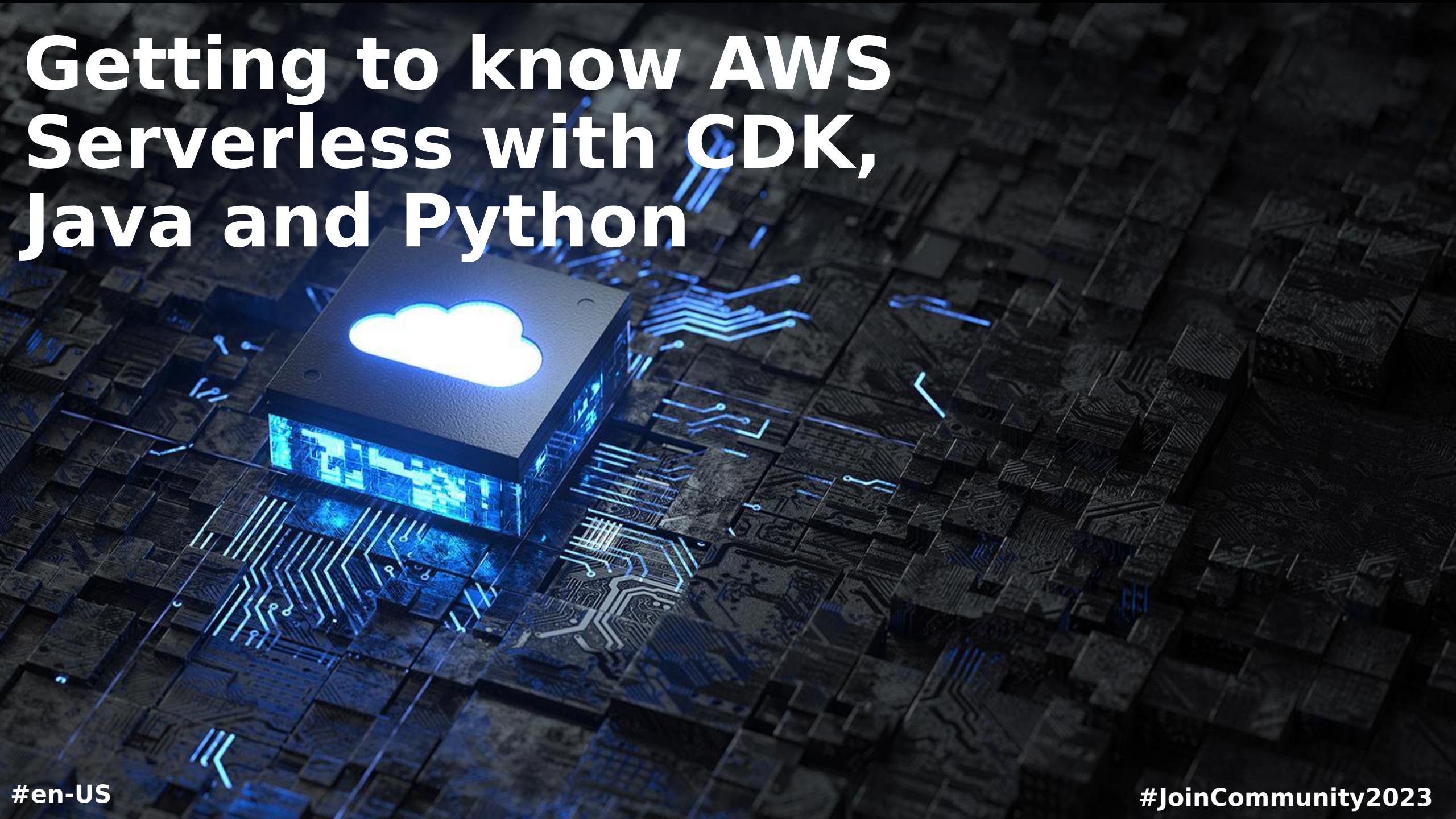


Conhecendo AWS Serverless com CDK, Java e Python



Getting to know AWS Serverless with CDK, Java and Python



About the Speaker

Weder Mariano de Sousa

Specialist Senior - GFT

Technician **System Development**

Graduated **Computer Science**

Post Graduate in **Midias UFG**

Post Graduate in **Information Security**



AWS
community
builder

<https://www.linkedin.com/in/wedermarianodesousa/>

<https://github.com/weder96>

<https://twitter.com/weder96>

<https://dev.to/weder96>



GOJava

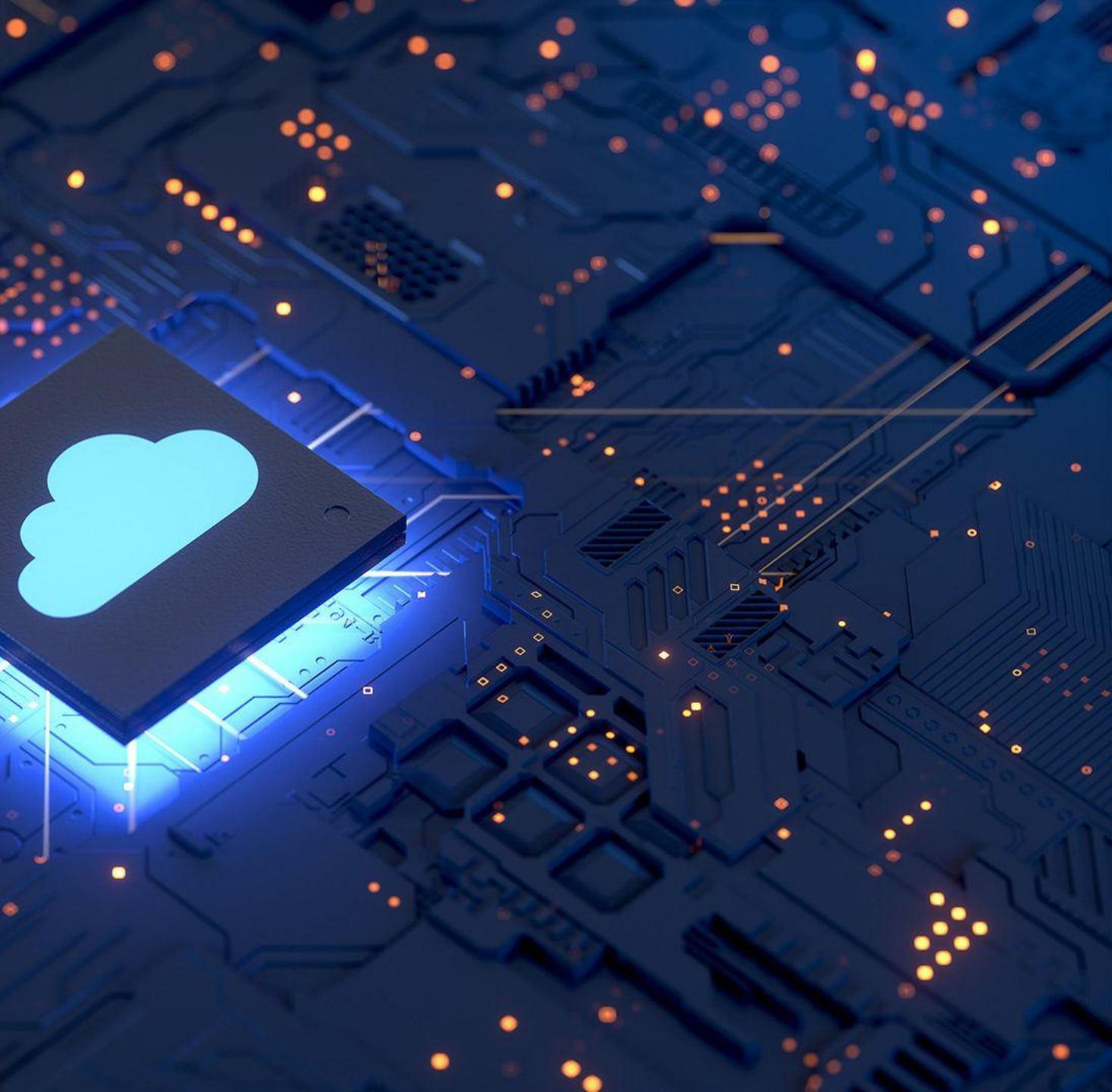


AWS User Group Goiânia



Agenda

1. Question AI
2. Install AWS Command Line Interface (AWS CLI) Linux
3. AWS IDE Toolkits
4. Prerequisites and Tools
5. Serverless
6. AWS Storage
7. AWS Database
8. AWS API Gateway
9. AWS Application Integration
10. AWS Application Orchestration
11. AWS Lambda Overview
12. Invoke Lambda Functions
13. Pattern: The comfortable “REST”
14. Architecture Serverless
15. AWS CDK For Java and Python
16. Next Steps





Question AI

Criar uma rotina de estudo focado na aprendizagem de Serverless AWS ?





Semana 1-2: Fundamentos da AWS

Semana 3-4: Conceitos Básicos de Serverless

Semana 5-6: Desenvolvimento Serverless

Semana 7-8: Arquitetura e Segurança Serverless

Semana 9-10: Monitoramento e Depuração

Semana 11-12: Serverless Avançado e Projetos

Nas últimas semanas, vá além dos conceitos básicos:

Serverless Framework: Familiarize-se com o Serverless Framework para facilitar o desenvolvimento e implantação de aplicações Serverless.

Projetos Práticos: Comece a criar projetos práticos para aplicar seus conhecimentos e construir aplicações Serverless reais.

Lembre-se de que a prática é fundamental para o aprendizado.

À medida que você avança na sua rotina de estudo, trabalhe em projetos reais para consolidar seus conhecimentos.

Serverless é uma abordagem poderosa, e dominá-la pode abrir muitas oportunidades no mundo da computação em nuvem.

ChatGPT

Além da Rotina de Estudo:

Participe de comunidades online relacionadas à AWS e Serverless para fazer perguntas, compartilhar conhecimentos e aprender com os outros.

Considere a certificação **AWS Certified Developer - Associate** para validar suas habilidades Serverless.

Mantenha-se atualizado lendo blogs, assistindo a webinars e participando de eventos relacionados à AWS e Serverless.



Bard

Semana 1-2: Introdução a Serverless AWS

Semana 3-4: AWS Cognito, AWS CloudWatch AWS CloudFormation

Semana 5-6: Avanços em Serverless AWS Casos de uso de Serverless AWS Projetos de Serverless

Projetos pessoais: Construção de aplicações Serverless AWS Recursos

Estudo autônomo: Leitura de documentação

Documentação da AWS Tutoriais da AWS

Esta rotina pode ser adaptada de acordo com as necessidades e objetivos do aluno.

Por exemplo, alunos com experiência em desenvolvimento de software podem avançar mais rapidamente no conteúdo.

Alunos com menos experiência podem dedicar mais tempo ao estudo dos conceitos básicos.

Encontre um grupo de estudo. Trabalhar com outras pessoas pode ajudá-lo a aprender e resolver problemas.

Não tenha medo de pedir ajuda. Se você ficar preso em um conceito, não tenha medo de procurar ajuda do seu instrutor, de um amigo ou de um grupo de discussão online.

Estabeleça metas realistas. Não tente aprender tudo de uma vez.

Comece com os conceitos básicos e avance gradualmente.

Crie um cronograma e cumpra-o. Ter um plano o ajudará a se manter motivado e no caminho certo.



Install AWS Command Line Interface (AWS CLI) Linux



Install AWS Linux

You must be able to extract or "unzip" the downloaded package. If your operating system doesn't have the built-in unzip command, use an equivalent.

The AWS CLI uses glibc, groff, and less. These are included by default in most major distributions of **Linux**.

We support the AWS CLI on 64-bit versions of recent distributions of **CentOS**, **Fedora**, **Ubuntu**, **Amazon Linux 1**, **Amazon Linux 2** and **Linux ARM**.

Install or update the AWS CLI

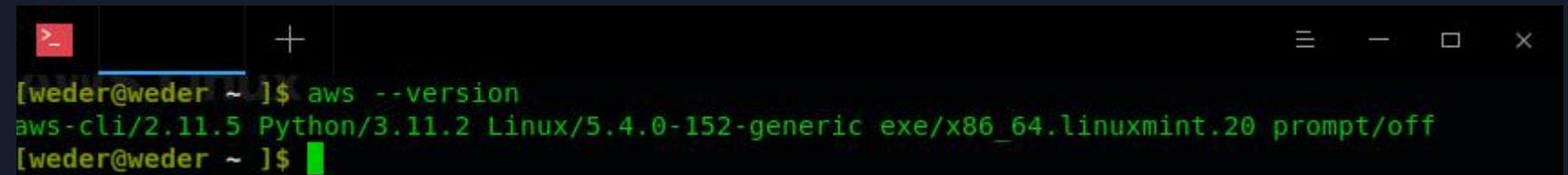
To update your current installation of AWS CLI, download a new installer each time you update to overwrite previous versions. Follow these steps from the command line to install the AWS CLI on Linux.

We provide the steps in one easy to copy and paste group based on whether you use 64-bit Linux or Linux ARM. See the descriptions of each line in the steps that follow.

Linux x86 (64-bit) Linux ARM

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

Because AWS doesn't maintain third-party repositories.
We can't guarantee that they contain the latest version of the AWS CLI.



A screenshot of a terminal window titled 'Terminal' showing the AWS CLI installed and running. The command 'aws --version' is entered, and the output shows the AWS CLI version 2.11.5, Python 3.11.2, and Linux 5.4.0-152-generic. The terminal has a dark theme with white text and a black background.

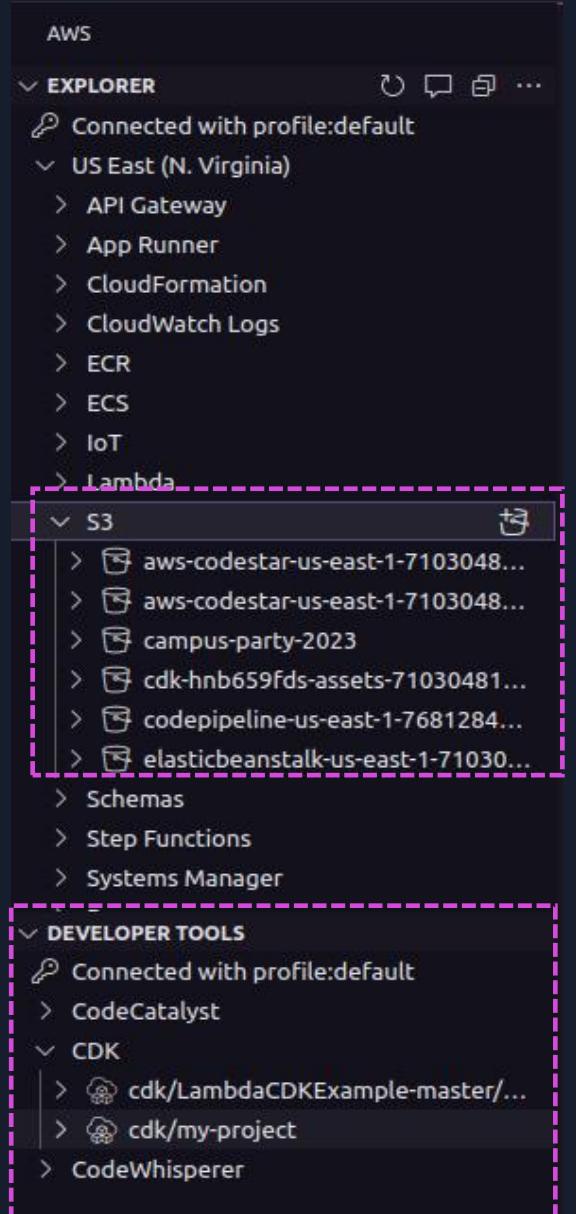
```
[weder@weder ~]$ aws --version
aws-cli/2.11.5 Python/3.11.2 Linux/5.4.0-152-generic exe/x86_64.linuxmint.20 prompt/off
[weder@weder ~]$
```



AWS IDE Toolkits

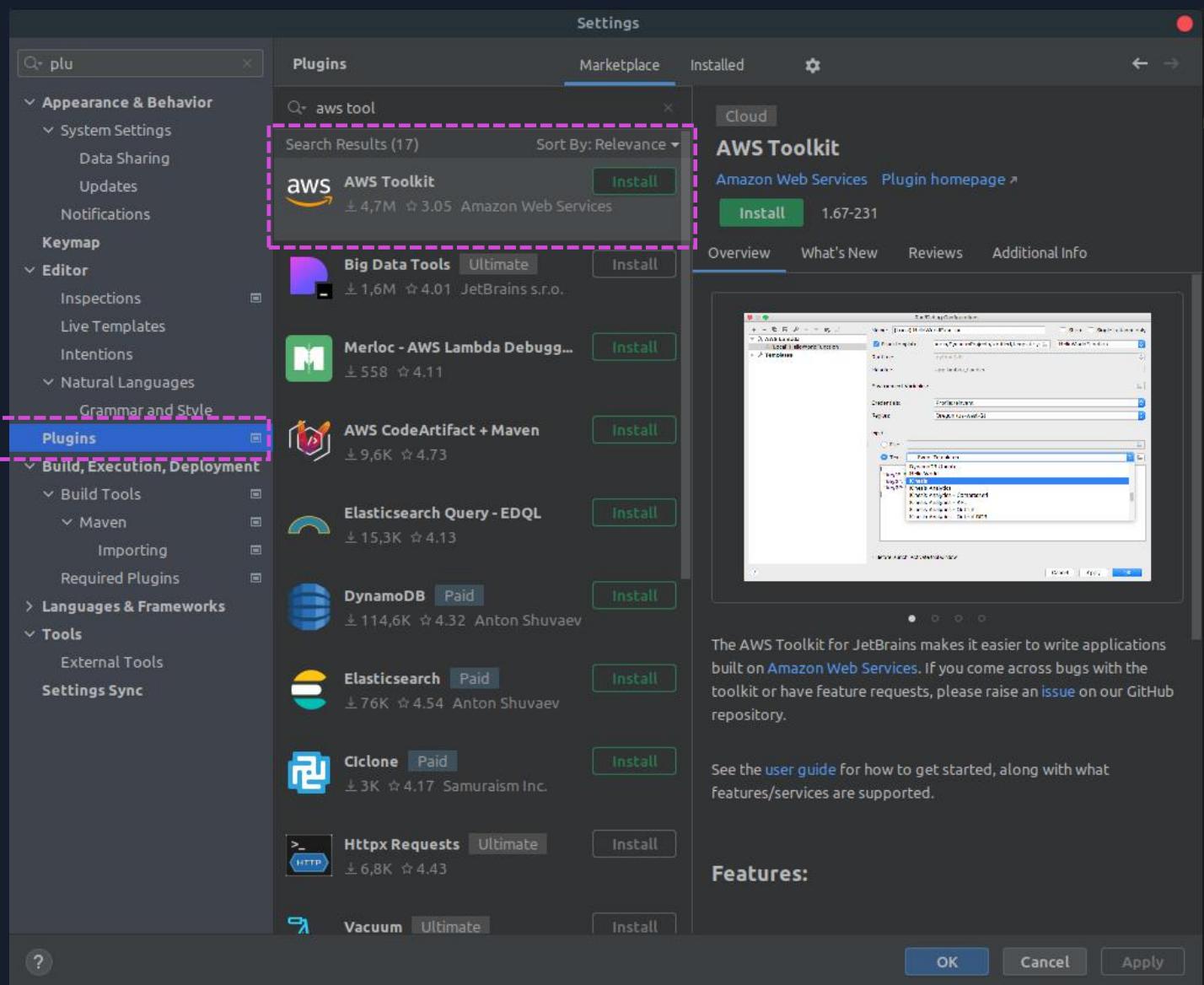


AWS Toolkit for Visual Studio Code



https://aws.amazon.com/serverless/?nc1=h_ls
<https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.aws-toolkit-vscode>

What is Toolkits de IDE da AWS ?



What is Toolkits de IDE da AWS ?

Amazon S3 > Buckets > campus-party-2023

campus-party-2023 Info

Objects Properties Permissions Metrics Management Access Points

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to

C Copy S3 URI Copy URL Download Open Delete Actions ▾

Find objects by prefix

Name	Type	Last modified
AB177.jpg	jpg	May 31, 2023, 15:25:43 (UTC-03:00)
AB188.jpg	jpg	May 31, 2023, 15:25:44 (UTC-03:00)
AB199.jpg	jpg	May 31, 2023, 15:25:45 (UTC-03:00)

The screenshot shows the IntelliJ IDEA interface with the AWS Toolkit plugin installed. The left sidebar displays the AWS Toolkit navigation pane, which includes sections for Project, AWS Toolkit, default (selected), us-east-1, Explorer, and Developer Tools. Under the AWS Toolkit section, the S3 category is expanded, showing the 'campus-party-2023' bucket. The right side of the interface shows the main code editor area with a file named 'AB199.jpg' open. A dashed purple rectangle highlights the S3 entry in the AWS Toolkit sidebar, and another dashed purple rectangle highlights the list of objects in the code editor's file tree.





Prerequisites and Tools



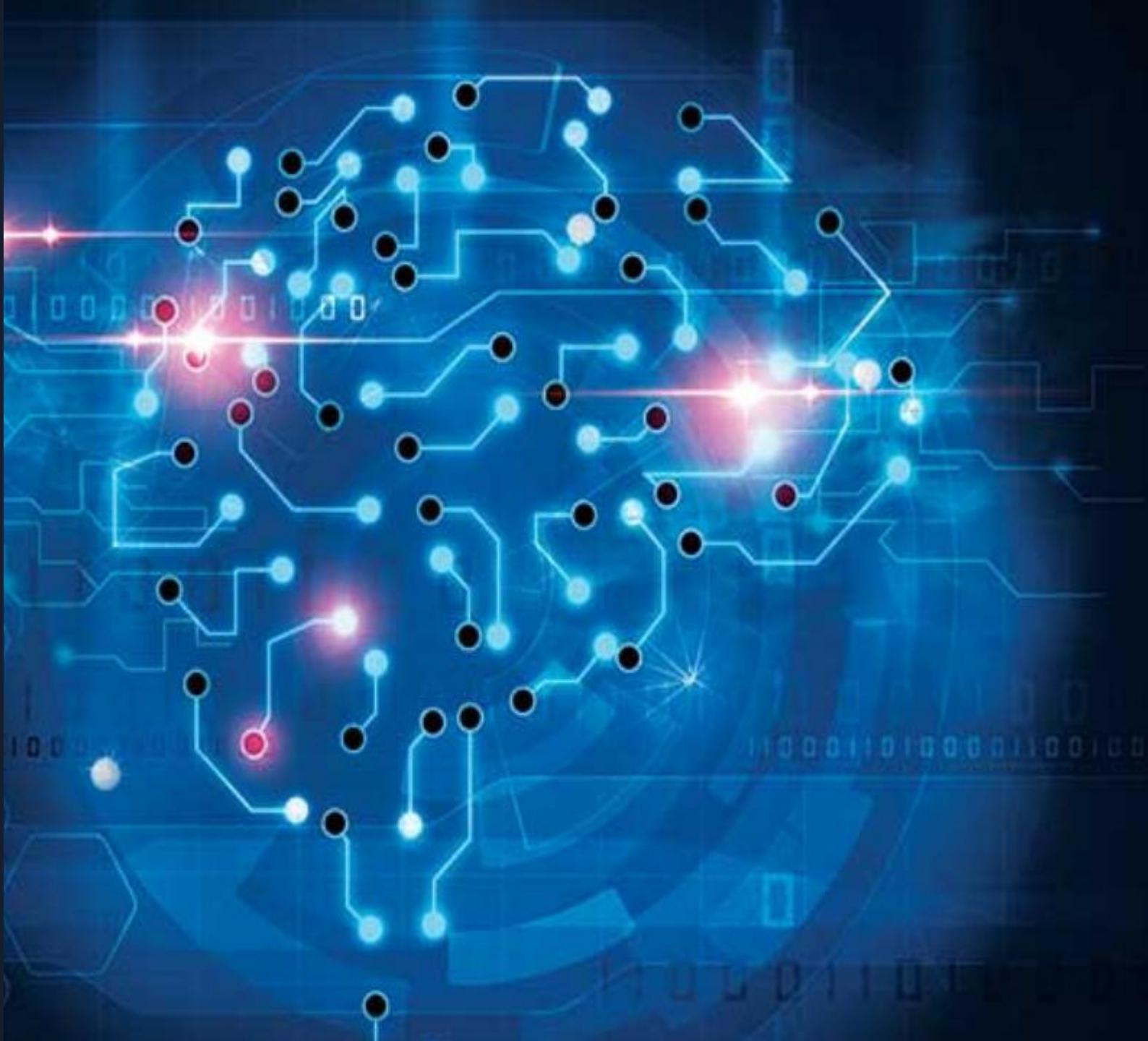
Prerequisites and Tools

6 main Prerequisites;

1. AWS Account and User
2. AWS CLI
3. Python 3.10
4. IDE for your programming language = Visual Studio Code
5. IDE for your programming language = IntelliJ IDE
6. Postman



Serverless



What is Serverless ?

Serverless is a way to describe the services that enable you to build and run applications without thinking about servers. AWS has serverless services for all layers of your stack.

In many public forums, like our marketing website, **we list these services under the Serverless heading** because it **improves discoverability**.

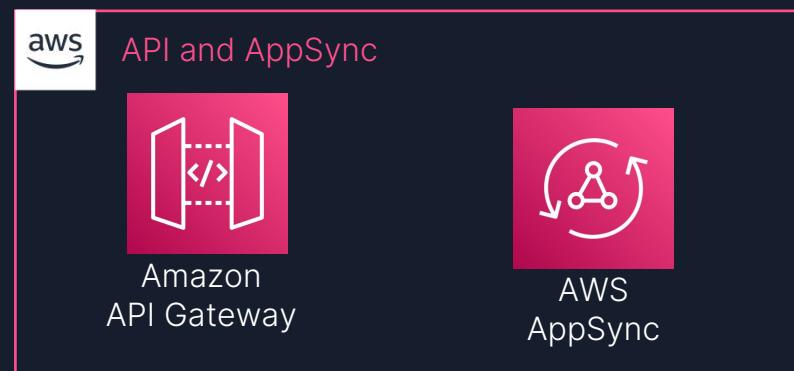
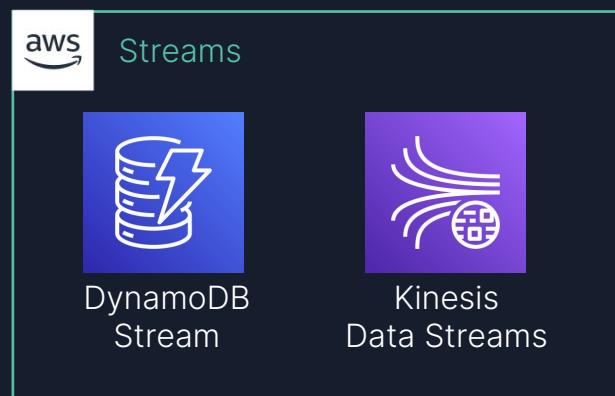
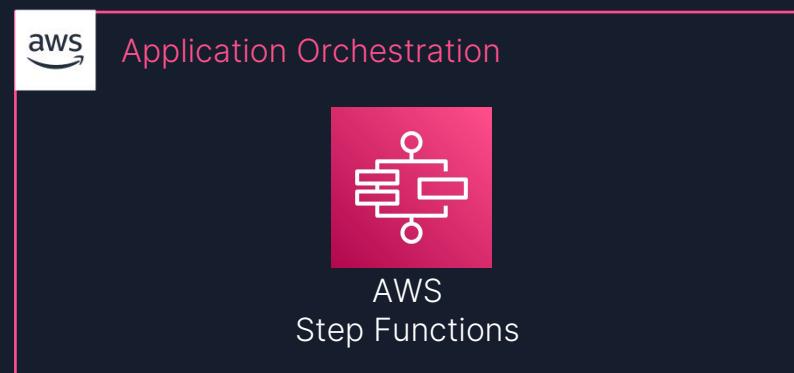
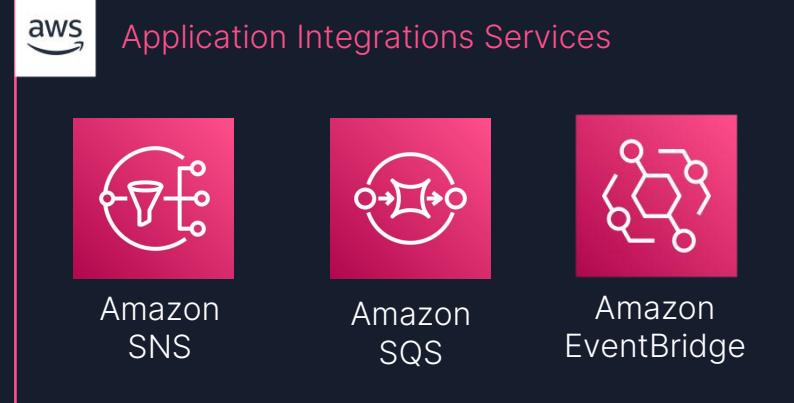
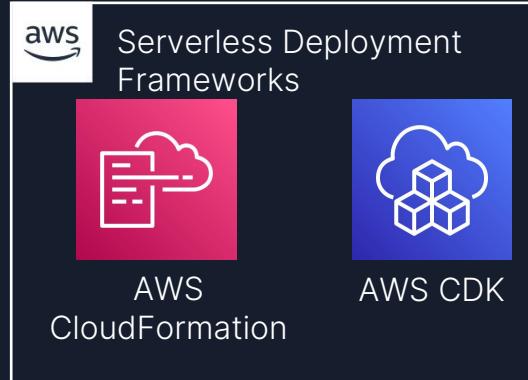
When it comes to building an architecture diagram, **it's more intuitive to use a functional categorization**. Please find the architecture icon for each serverless service in its functional category.

You can use the serverless category icon to describe serverless applications or architectures.



https://aws.amazon.com/serverless/?nc1=h_ls

AWS Services Serverless



How to build, run and deploy?



When starting designing, and building serverless function, you might wonder how to deploy your function into the cloud. With AWS, there are some ways we can deploy, test and invoke your function:



Using the AWS Console Management: we can create lambda function, upload code, add triggers, and test your Lambda function manually. You might use this way when first.



AWS CLI: you also can use AWS Lambda CLI to create, deploy, invoke, manage, monitor your Lambda function. You can use existing commands to deploy and test your Lambda function automatically without manual process. But this isn't good for production and large project.



AWS Serverless Application Model (AWS SAM) is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings.



AWS Cloud Development Kit (AWS CDK) is an open source software development framework to define your cloud application resources using familiar programming languages.

How to build, run and deploy?



[Serverless Framework](#) - The Serverless Framework consists of an open source CLI and a hosted dashboard. Together, they provide you with full serverless application lifecycle management.



[Chalice](#) is a framework for writing serverless apps in Python. It allows you to quickly create and deploy applications that use AWS Lambda.



[Arc.codes](#) provides everything you need to build massively scalable serverless apps with low code, clear and terse config, and zero ceremony.



[Claudia.js](#) makes it easy to deploy Node.js projects to AWS Lambda and API Gateway.



AWS Storage



What Is AWS S3?

Cloud storage is a web service where data can be stored, accessed, and quickly backed up by users on the internet. It is more reliable, scalable, and secure than traditional on-premises storage systems.

Amazon (Simple Storage Service) S3 stands for Amazon Simple Storage Service is a Object storage built to retrieve any amount of data from anywhere. Designed for durability.



Provides object storage, which is built for **storing** and **recovering** any amount of data from anywhere over the internet.

Enables users to store and retrieve any amount of data at any time or place, giving developers access to highly scalable, reliable, fast and inexpensive data storage.



Amazon S3

Use Amazon S3 to store and protect any amount of data for a range of use cases, like data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

Amazon S3 Benefits and Features

Storage Classes; S3 offers a range of storage classes designed for different use cases. S3 Standard, S3 Standard-IA, S3 Galcier.

Storage Management; S3 has storage management features that we can use to manage costs, meet regulatory requirements, reduce latency.

Access Management; S3 provides features for auditing and managing access to buckets and objects.

Data Processing; To transform data and trigger workflows to automate a variety of other processing activities at scale.

Storage logging and monitoring; S3 provides logging and monitoring tools that you can use to monitor and control how your Amazon S3 resources are being used. S3 also offers features to gain visibility into your storage usage.

Strong Consistency; S3 provides strong read-after-write consistency for PUT and DELETE requests of objects in your Amazon S3 bucket in all AWS Regions.



Amazon S3

Amazon S3 CLI Commands

joinCommunity2023 / learning / 09_S3 / README.md

Preview

Code

Blame

200 lines (144 loc) · 4.59 KB

Code 55% faster with GitHub Copilot

Create S3 Bucket

```
aws s3 mb s3://s3-join-community-2023-files
```



Create S3 bucket in specific AWS region

```
aws s3 mb s3://s3-join-community-2023-files-us-ea-2 --region us-east-2
```



put version at bucket

```
aws s3api put-bucket-versioning --bucket s3-join-community-2023-files-us-ea-2 --versioning-configuration Status=Enat
```



check bucket status version

```
aws s3api get-bucket-versioning --bucket s3-join-community-2023-files-us-ea-2
```



Suspending versioning

```
aws s3api put-bucket-versioning --bucket s3-join-community-2023-files-us-ea-2 --versioning-configuration Status=Susp
```



Copy files to S3 bucket

```
aws s3 cp images/ s3://s3-join-community-2023-files --recursive --include "*.jpg"
```



List S3 buckets

```
aws s3 ls
```



Amazon S3



AWS Database



What Is Amazon DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.

Serverless, key-value NoSQL database designed to run high-performance applications at any scale.

Create **database tables** that can store and retrieve any amount of data and serve any level of request traffic.

Scale up or down the throughput of your tables without downtime or performance degradation. DynamoDB provides on-demand backup capability.

High Availability and Durability

DynamoDB automatically spreads data and traffic for your tables across enough servers to meet your throughput.



DynamoDB Primary Key, Partition Key and Sort Key

A primary key **uniquely identifies** each item in the table, so no two items can have the same key. DynamoDB supports two different kinds of primary keys:

Partition key

Partition key and sort key

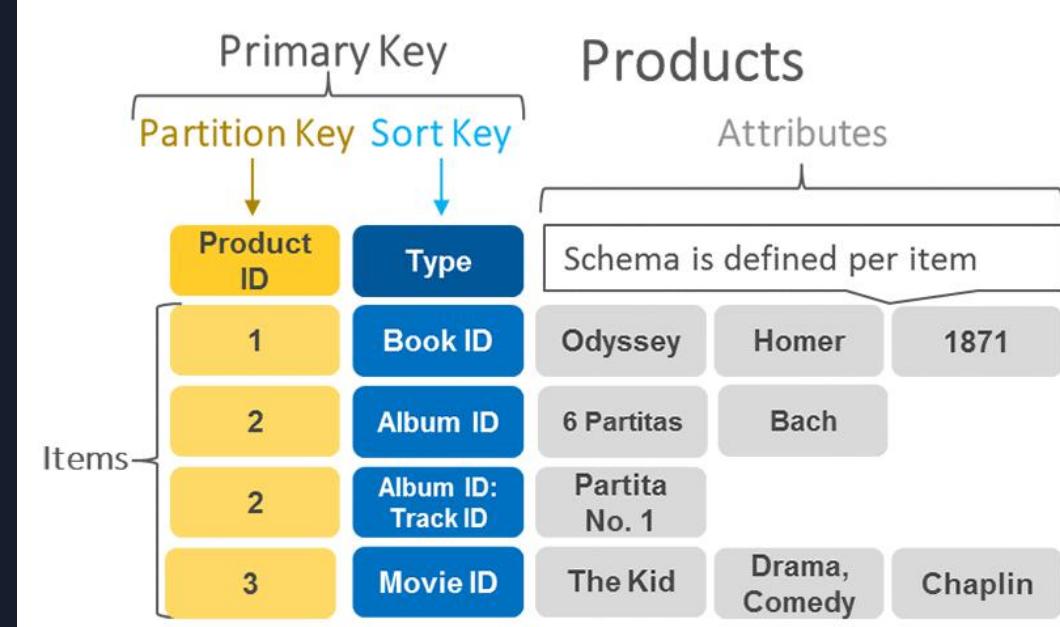
Partition key

A simple primary key, composed of one attribute known as the partition key.

Partition key and Sort Key

It is 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.

DynamoDB uses the partition key value as input to an internal hash function. A composite primary key gives you additional flexibility when **querying data**



General information

Partition key
orderId (String)

Sort key
orderDate (String)

Alarms
 No active alarms

Point-in-time recovery (PITR) [Info](#)
 Off

► Additional info



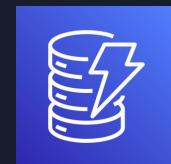
DynamoDB Python(Boto3)

Files

weder96 learning serverless with cli commands

Code Blame 39 lines (32 loc) · 1004 Bytes Code 55% faster with GitHub Copilot

```
1 import json
2 import boto3
3 import uuid
4
5 client = boto3.resource('dynamodb')
6
7 def lambda_handler(event, context):
8     print('Starting input lambda function call')
9     print(event['body'])
10    table = client.Table('orders')
11    myuuid = uuid.uuid4()
12
13    print('Your UUID is: ' + str(myuuid))
14
15    order=json.loads(event['body'])
16
17    data = table.put_item(
18        Item={
19            'orderId': str(myuuid),
20            'orderDate': order['orderDate'],
21            'status': order['status'],
22            'desc': order['desc'],
23            'updateOrderDate': order['updateOrderDate'],
24            'Name': order['Name'],
25            'Email': order['Email']
26        }
27    )
28
```



https://github.com/weder96/joinCommunity2023/tree/main/learning/08_dynamodb

DynamoDB CLI Commands

JoinCommunity2023 / learning / 08_dynamodb /

Name	Last commit message
...	
crud	🚀 add boto3 dynamodb with lambda
README.md	🚀 add boto3 dynamodb with lambda

README.md

Dynamodb

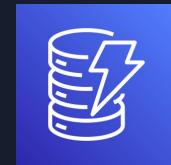
CLI

Create table DynamoDB

```
aws dynamodb create-table \  
  --table-name orders \  
  --attribute-definitions \  
    AttributeName=id,AttributeType=S \  
    AttributeName=status,AttributeType=S \  
  --key-schema \  
    AttributeName=id,KeyType=HASH \  
    AttributeName=status,KeyType=RANGE \  
  --provisioned-throughput \  
    ReadCapacityUnits=5,WriteCapacityUnits=5 \  
  --table-class STANDARD
```

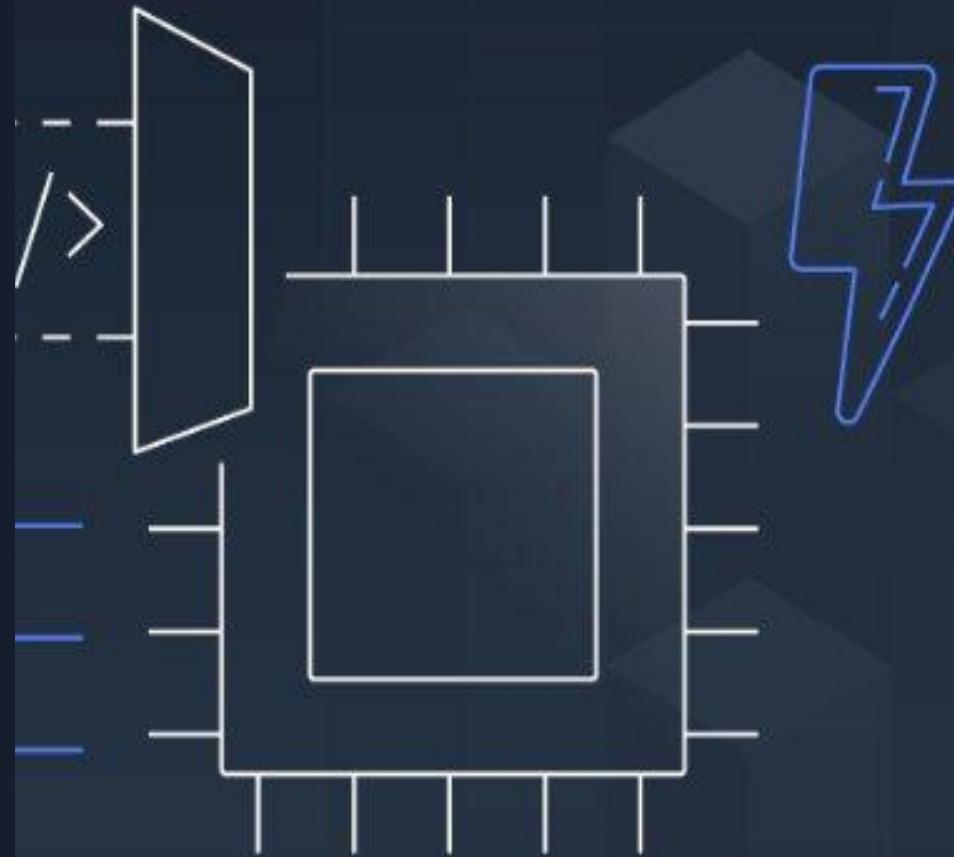
Describe table DynamoDB

```
aws dynamodb describe-table --table-name Order
```





AWS API Gateway



What Is Amazon API Gateway ?

Enables developers to create, publish, maintain, monitor, and secure APIs at any scale.

This is a HIPAA eligible service.

Allows creating, deploying, and managing a RESTful API to expose backend HTTP endpoints, Lambda functions, or other AWS services.



Together with Lambda, API Gateway forms the app-facing part of the AWS serverless infrastructure.



Amazon
API Gateway

Amazon API Gateway Concepts

API Deployment - a point-in-time snapshot of your API Gateway API resources and methods. To be available for clients to use, the deployment must be associated with one or more API stages.

API endpoints - host names APIs in API Gateway, which are deployed to a specific region and of the format: rest-api-id.execute-api.region.amazonaws.com

API key - An alphanumeric string that API Gateway uses to identify an app developer who uses your API.

API stage - A logical reference to a lifecycle state of your API. API stages are identified by API ID and stage name.

Model - Data schema specifying the data structure of a request or response payload.

Private API - An API that is exposed through interface VPC endpoints and isolated from the public internet

Private integration - An API Gateway integration type for a client to access resources inside a customer's VPC through a private API endpoint without exposing the resources to the public internet.

Proxy integration - You can set up a proxy integration as an HTTP proxy integration type or a Lambda proxy integration type.

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.



Amazon
API Gateway

Amazon API Gateway Endpoint Types

Edge-optimized API endpoint: The default host name of an API Gateway API that is deployed to the specified region while using a CloudFront distribution to facilitate client access typically from across AWS regions. API requests are routed to the nearest CloudFront Point of Presence.

Regional API endpoint: The host name of an API that is deployed to the specified region and intended to serve clients, such as EC2 instances, in the same AWS region. API requests are targeted directly to the region-specific API Gateway without going through any CloudFront distribution.

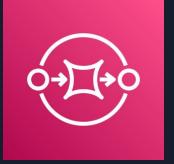
Private API endpoint: Allows a client to securely access private API resources inside a VPC. Private APIs are isolated from the public Internet, and they can only be accessed using VPC endpoints for API Gateway that have been granted access.



Amazon
API Gateway



SNS

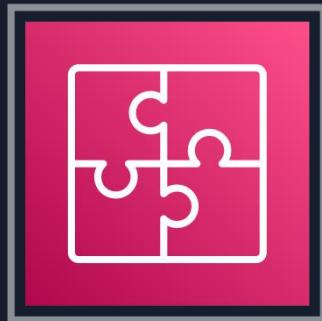


SQS



EVENTBRIDGE

AWS Application Integration





Amazon SNS



Amazon SNS: Fully Managed Pub/Sub Messaging



Application
Integration

Application integration

The Fanout scenario is when a message published to an SNS topic is replicated and pushed to multiple endpoints.

Application alerts

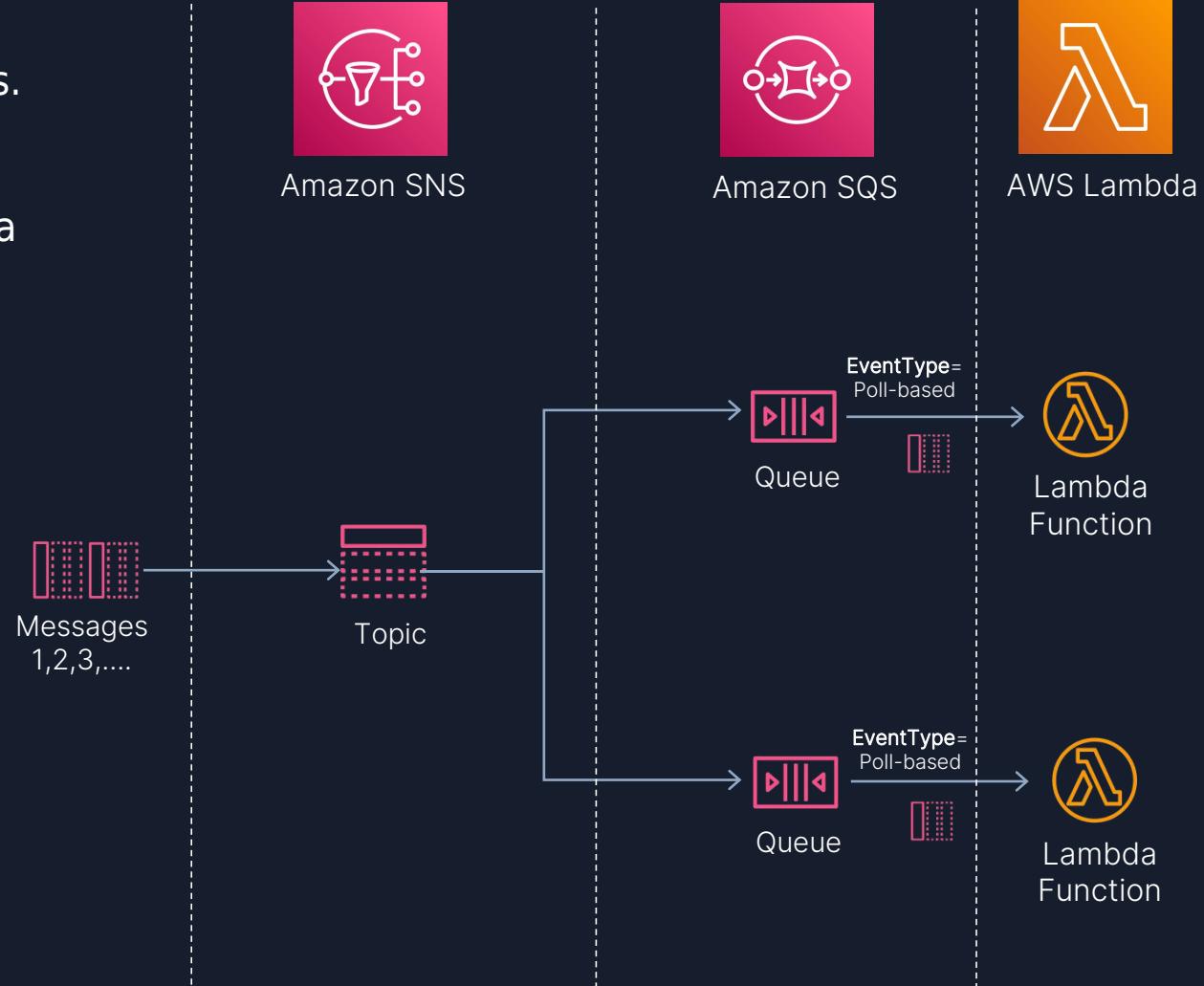
Amazon SNS can send notifications to specified users via SMS and email.

User notifications

Amazon SNS can send push email messages and text messages to individuals or groups.

Mobile push notifications

Mobile push notifications enable you to send messages directly to mobile apps.



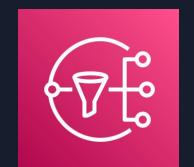
Amazon SNS: Fully Managed Pub/Sub Messaging



Application
Integration

Amazon SNS Actions

- Create a topic
- Delete a subscription
- Delete a topic
- List topics
- Publish an SMS text message
- Publish to a topic
- Set a dead-letter queue for a subscription
- Set a filter policy
- Set the default settings for sending SMS messages
- Set topic attributes
- Subscribe a Lambda function to a topic
- Subscribe a mobile application to a topic



Amazon SNS

https://aws.amazon.com/sns/?nc1=h_ls

Amazon SNS CLI Commands and (Python) Boto3

joinCommunity2023 / learning / 15_lambda sns / README.md 

 weder96 🚀 topics cli

Preview Code Blame 106 lines (75 loc) · 3.5 KB  Code 55% faster with GitHub Copilot

Lambda and SNS with Python

Zip file Linux

```
zip function.zip app.py
```



Create Lambda Function

```
aws lambda create-function \
--function-name snsTopic \
--runtime python3.10 \
--zip-file fileb://function.zip \
--handler app.lambda_handler \
--role arn:aws:iam::710304818543:role/lambda-role-py
```



Create a topic

To create a topic, use the sns create-topic command and specify the name to assign to the topic.

```
aws sns create-topic --name my-topic
```



Amazon SNS

https://github.com/weder96/joinCommunity2023/blob/main/learning/15_lambda sns/README.md





Amazon SQS



What is Amazon SQS ?



Application
Integration

Amazon SQS stands for **Simple Queue Service** is **fully managed message queues** for microservices, distributed systems, and serverless applications.

Enables you to **decouple** and **scale microservices**, distributed systems, and serverless applications.

Eliminates the **complexity** and overhead associated with managing and operating message-oriented middleware.

Send, store, and receive messages between software components at any volume.

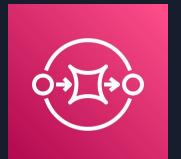
Two types of message queues.

Standard queues offer maximum throughput, best-effort ordering, and at-least-once delivery.

FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.

Integrate and **decouple** distributed software systems and components.

Provides a **generic web services API** that you can access using any programming language that the **AWS SDK** supports.



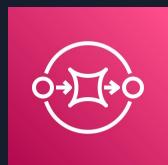
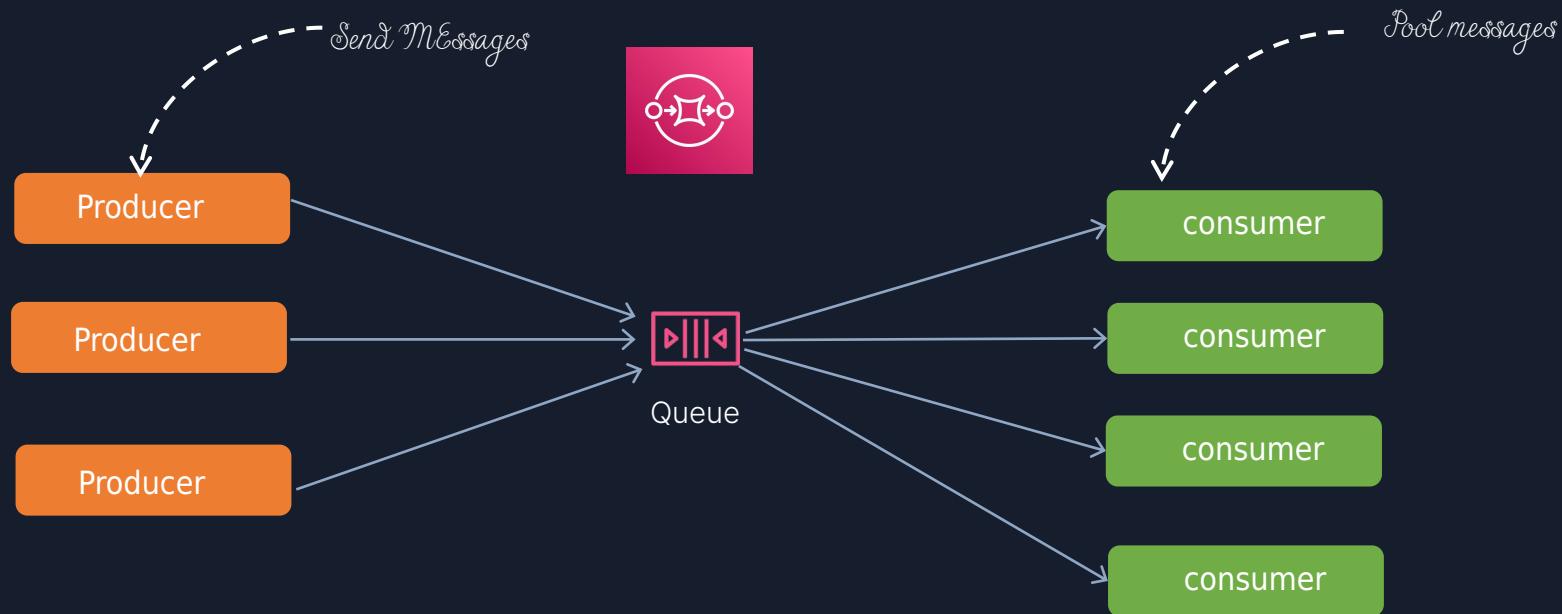
Amazon SQS

https://aws.amazon.com/sns/?nc1=h_ls

Amazon SQS What's a queue?



Application
Integration



Amazon SQS

https://aws.amazon.com/sns/?nc1=h_ls

Benefits of Amazon SQS



Application
Integration

Eliminate administrative overhead

AWS manages all ongoing operations and underlying infrastructure needed to provide a highly available and scalable message queuing service. SQS queues are dynamically created and scale automatically.

Durability and Reliability deliver messages

Amazon SQS stores them on multiple servers. Standard queues support at-least-once message delivery, and FIFO queues support exactly-once message processing. SQS locks your messages during processing, so that multiple producers can send and multiple consumers can receive messages at the same time.

Scalability and Availability and cost-effectively

SQS scales elastically with your application so you don't have to worry about capacity planning and pre-provisioning. There is no limit to the number of messages per queue, and standard queues provide nearly unlimited throughput.

Security - Keep sensitive data secure

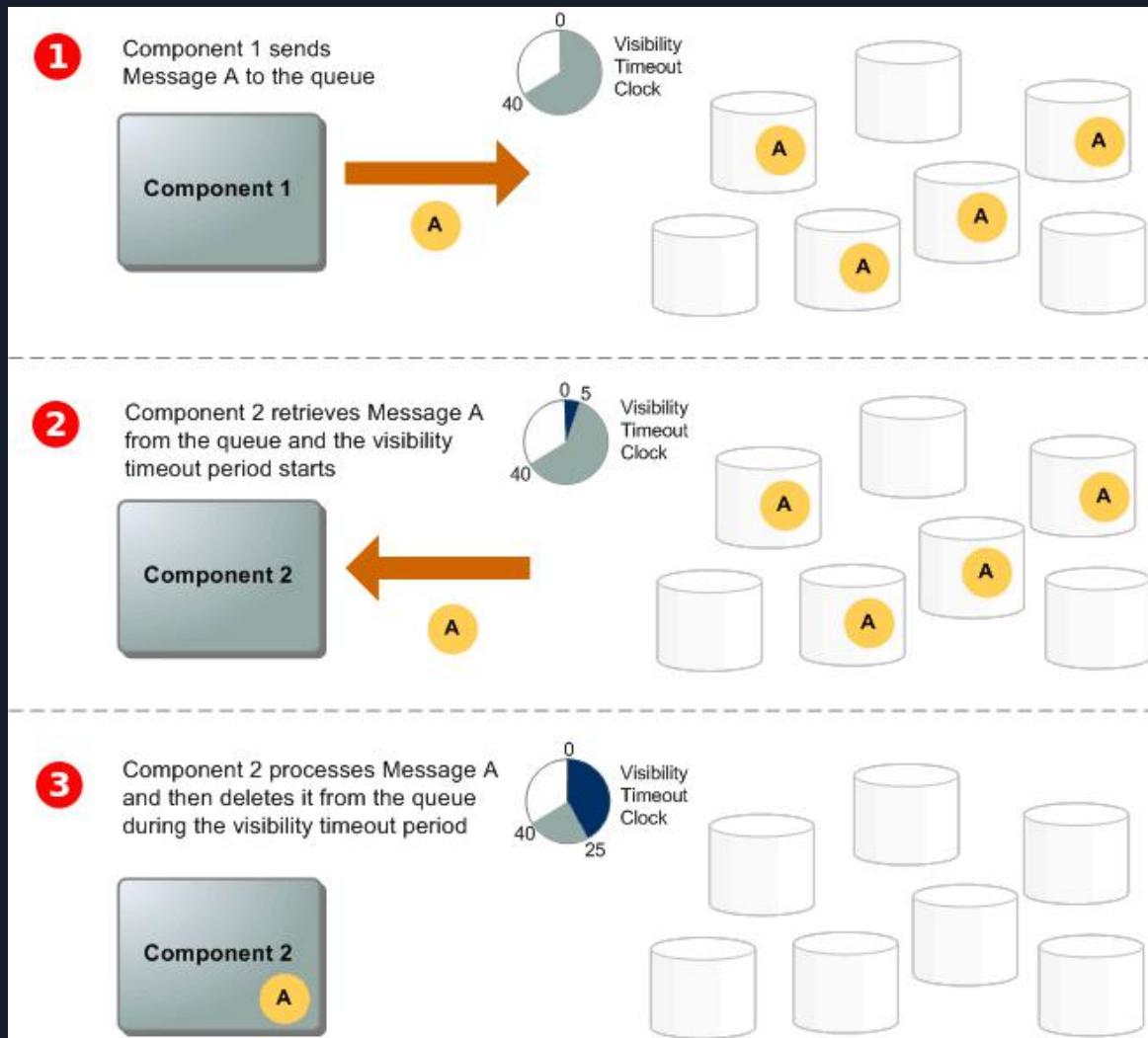
Amazon SQS exchanges sensitive data between applications using server-side encryption (SSE) to encrypt each message body.



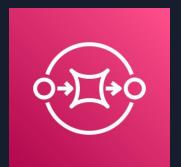
Amazon SQS

https://aws.amazon.com/sns/?nc1=h_ls

The lifecycle of an Amazon SQS message



- ① A producer (component 1) sends message **A** to a queue, and the message is distributed across the Amazon SQS servers redundantly.
- ② When a consumer (component 2) is ready to process messages, it consumes messages from the queue, and message **A** is returned. While message **A** is being processed, it remains in the queue and isn't returned to subsequent receive requests for the duration of the **visibility timeout**.
- ③ The consumer (component 2) deletes message **A** from the queue to prevent the message from being received and processed again when the visibility timeout expires.



Amazon SQS CLI Commands and (Python) Boto3



Application
Integration

joinCommunity2023 / learning / 16_lambda_sqs / README.md

Preview | Code | Blame 237 Lines (170 loc) · 6.14 KB ⚡ Code 55% faster with GitHub Copilot

Command Line Interface

```
aws sqs create-queue --queue-name cli-queue-attr --attributes file://q-attributes.json
```

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/710304818543/cli-queue-attr --attribute-
```

Sending Messages to our AWS SQS Queue

```
aws sqs send-message --queue-url https://sqs.us-east-1.amazonaws.com/710304818543/cli-queue-attr --message-body "IOT-"
```

Reading Messages from the AWS SQS Queue

```
aws sqs --region us-east-1 receive-message --queue-url https://sqs.us-east-1.amazonaws.com/710304818543/cli-queue-attr
```

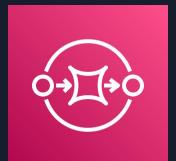
Deletion of a Processed Message from the AWS SQS Queue

```
aws sqs --region us-east-1 delete-message --queue-url https://sqs.us-east-1.amazonaws.com/710304818543/cli-queue-attr
```

Cleaning Up

```
aws sqs --region us-east-1 delete-queue --queue-url https://sqs.us-east-1.amazonaws.com/710304818543/cli-queue-attr
```

Get message with long polling



Amazon SQS

https://github.com/weder96/joinCommunity2023/blob/main/learning/16_lambda_sqs/README.md



Amazon EventBridge



What is Amazon EventBridge ?

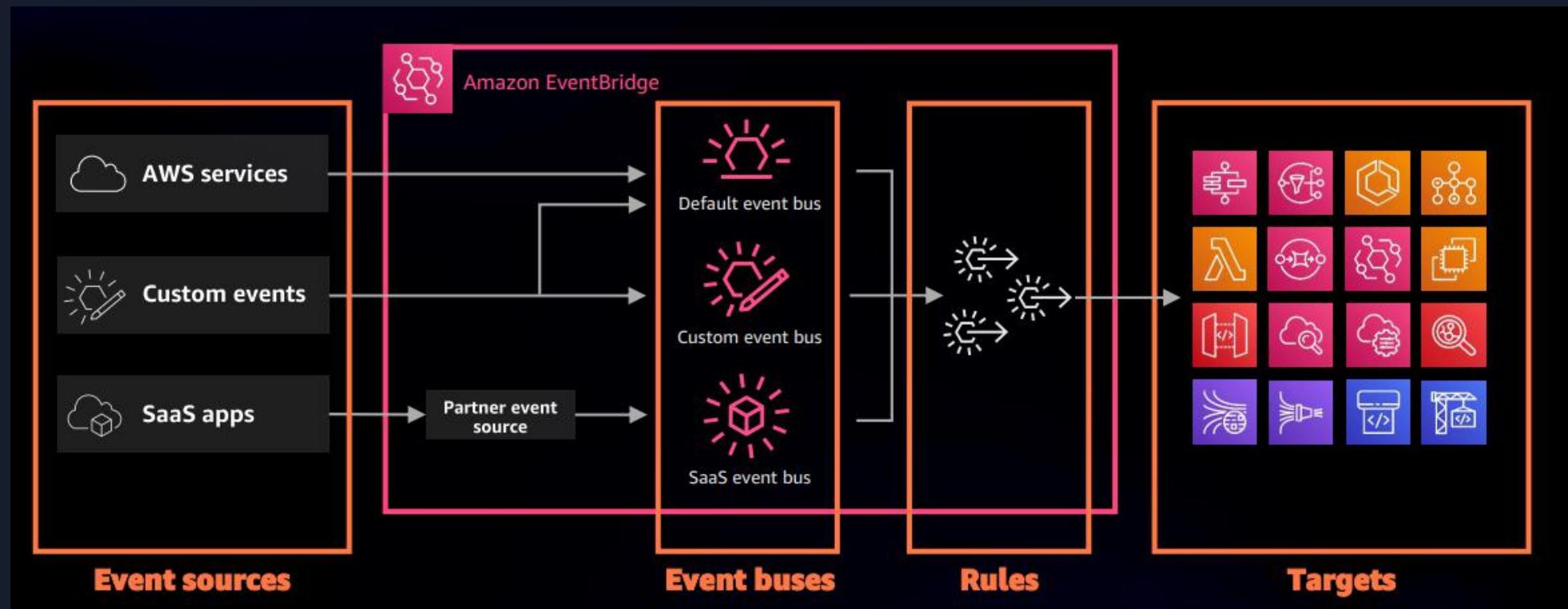
Serverless **event bus** service for AWS services

Build **event-driven** applications at scale using events generated from your apps

Use to connect your applications with data from a variety of sources, **integrated SaaS applications**

AWS services to targets such as AWS Lambda functions

Formerly **called Amazon CloudWatch Events**



What are events?

1. An event is defined in JSON
2. "Detail" is application specific
3. Envelope attributes are provided by
4. Amazon EventBridge
5. Producers create events
6. Consumers choose which events to
7. Listen to by using rules

```
{  
    "version": "0",  
    "id": "6ac4e27b-1234-1234-1234-5fb02c880319",  
    "detail-type": "OrderProcessor.OrderStarted",  
    "source": "awsserverlessda.serverlesspresso",  
    "account": "123456789012",  
    "time": "2021-11-28T13:12:30Z",  
    "region": "us-west-2",  
    "detail": {  
        "userId": "jbesw",  
        "orderId": "eYmAfqLD67v1bdUVile_D",  
        "drinkOrder": {  
            "icon": "barista-icons_cafe-latte",  
            "modifiers": [],  
            "drink": "Latte"  
        }  
    }  
}
```

Benefits of Amazon EventBridge

Build event-driven architectures

With EventBridge, your event targets don't need to be aware of event sources because you can filter and publish directly to EventBridge. Improves developer agility as well as application resiliency with loosely coupled event-driven architectures.

Connect SaaS apps

EventBridge ingests data from supported SaaS applications and routes it to AWS services and SaaS targets. SaaS apps to trigger workflows for customer support, business operations.

Write less custom code

You can ingest, filter, transform and deliver events without writing custom code.

The EventBridge schema registry stores a collection of easy-to-find event schemas.

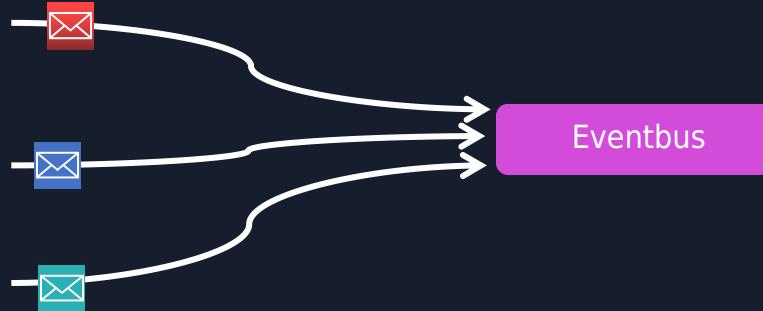
Reduce operational overhead

There are no servers to provision, patch, and manage. automatically scales based on the number of events ingested. Built-in distributed availability and fault-tolerance.

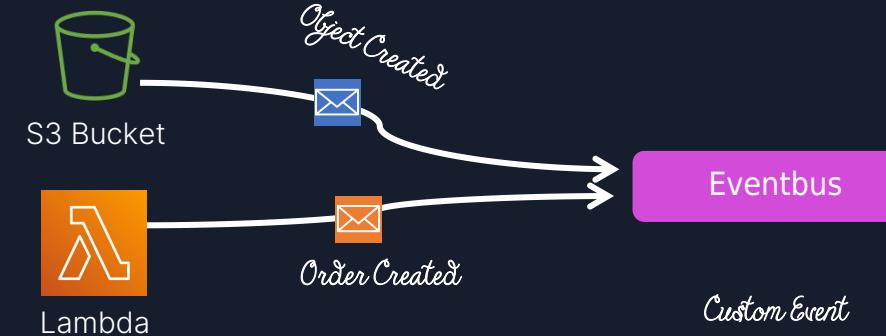
Native event archive and replay capability.



What are events?



Events are the heart of any event-driven application
Producers raise events consumers consume them



Some AWS Services send events to EventBridge for consumption
or you can raise your own custom events

EventBridge Concepts - Events - Event Buses - Rules - Targets

Amazon EventBridge Events

An event indicates a change in an environment such as an AWS environment or a SaaS partner service. Events are represented as JSON objects and they all have a similar structure, and the same top-level fields.

Amazon EventBridge Rules

A rule matches incoming events and sends them to targets for processing. A single rule can send an event to multiple targets, which then run in parallel.

An event pattern defines the event structure and the fields that a rule matches.

Amazon EventBridge Targets

A target is a resource or endpoint that EventBridge sends an event to when the event matches the event pattern defined for a rule. The rule processes the event data and sends the relevant information to the target.

Amazon EventBridge Event Buses

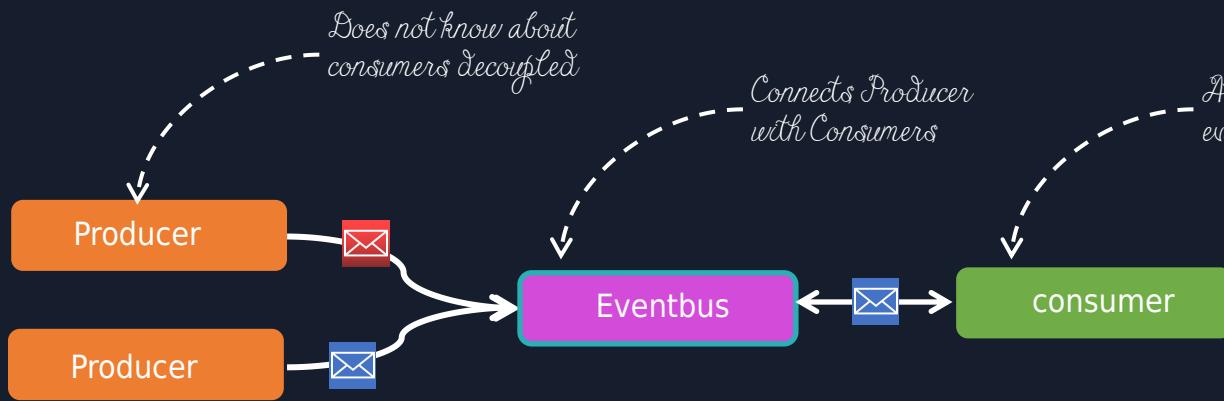
An event bus is a pipeline that receives events. Rules associated with the event bus evaluate events as they arrive.

A resource-based policy specifies which events to allow, and which entities have permission to create or modify rules or targets for an event.

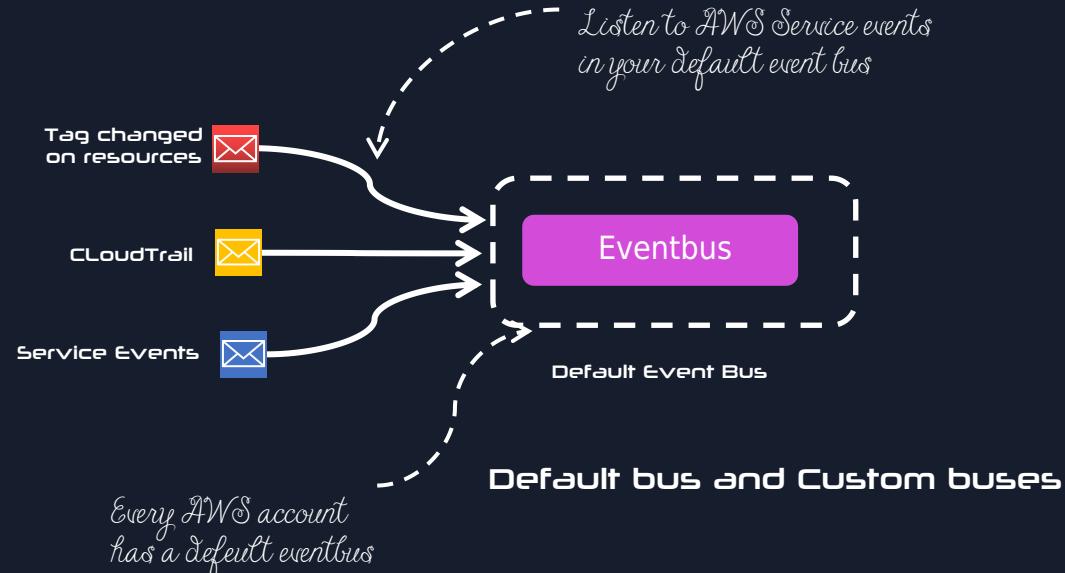


Amazon EventBridge

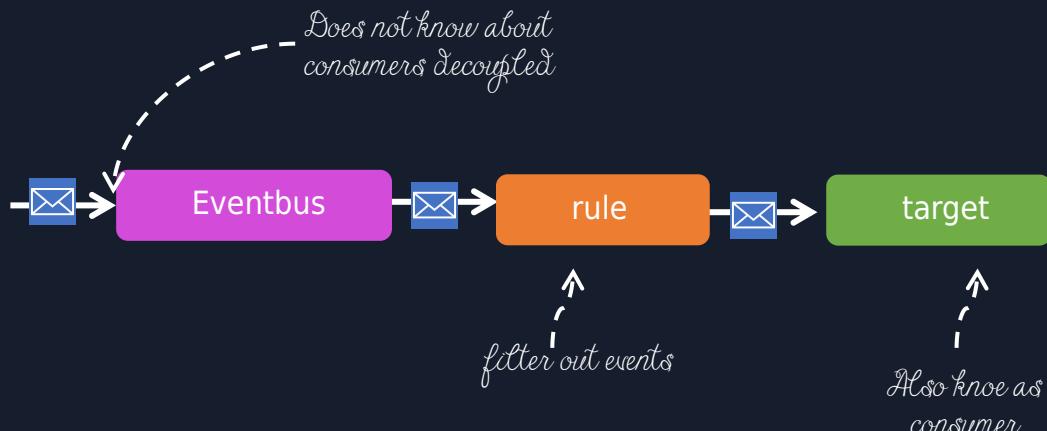
Understanding an EventBridge Bus



Events bus is a key Component

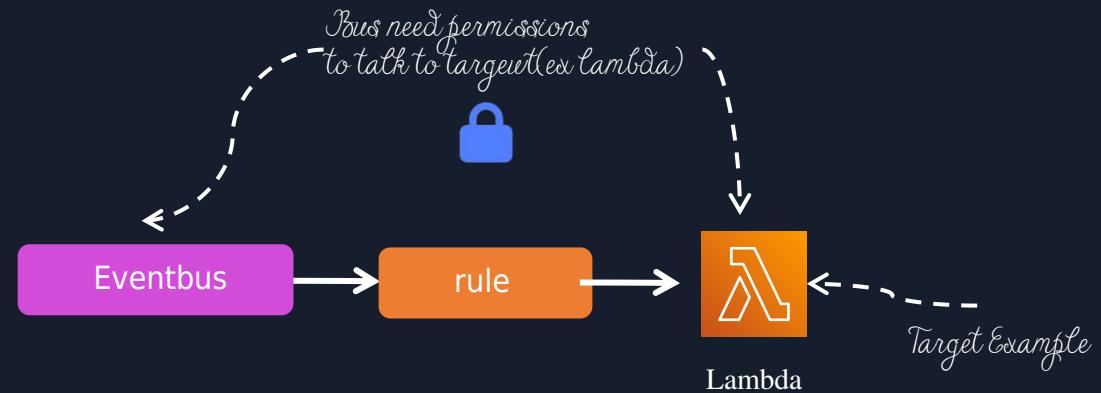


Understanding EventBridge target



What are target ?

Targets are consumer of your events
use rules to process events before reaching targets



Target Permissions

Your eventBridge bus need permissions to send events to targets

Amazon EventBridge CLI Commands and (Python) Boto3

joinCommunity2023 / learning / 19_eventbridge / README.md

weder96 learning serverless with cli commands

Preview Code Blame 201 lines (145 loc) · 4.4 KB Code 55% faster with GitHub Copilot

EventBridge

CLI

Amazon EventBridge

Amazon EventBridge - Developing with AWS SDK

EventBridge Client - AWS SDK for Python Boto3

<https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/events.html>

Commands : PutEventsCommand PutRuleCommand PutTargetsCommand

Event Bridge CLI Commands

Create an event bus

```
aws events create-event-bus --name "event-bridge-demo"
```

Create Role for EventBridge Event

Create Execution Role for AWS Lambda functions with AWS CLI

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



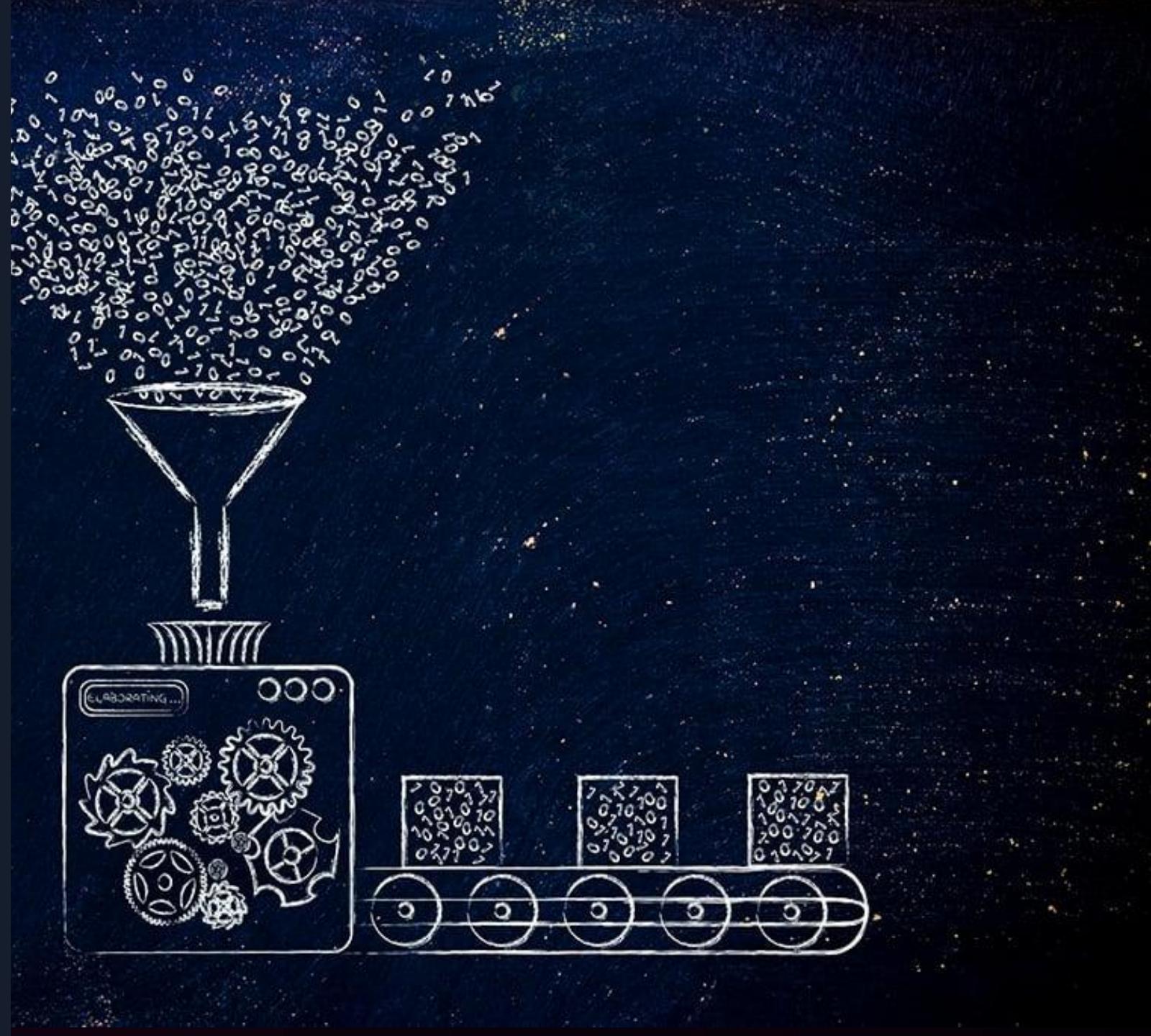
Amazon EventBridge

https://github.com/weder96/joinCommunity2023/blob/main/learning/19_eventbridge/README.md



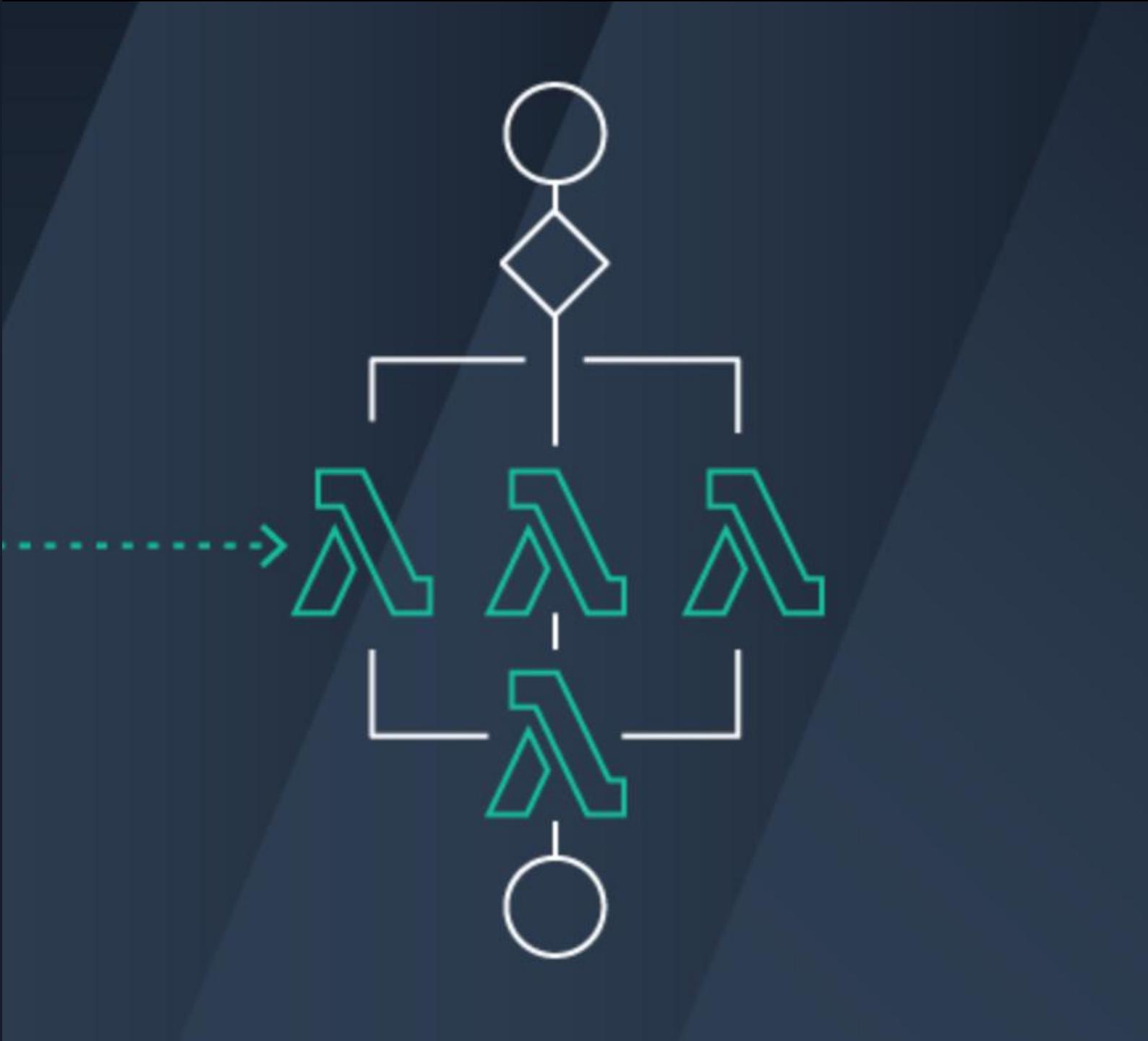
STEP FUNCTIONS

AWS Application Orchestration





AWS Step Functions



What is AWS Step Functions ?

AWS Step Functions is a low-code, **visual workflow** service that developers use to build **distributed applications**, automate IT and business processes.

Step Functions is a **serverless orchestration** service that combine AWS Lambda functions and other AWS services to build business critical applications.

Step Functions is based on **state machines and tasks**. A **state machine** is a **workflow**. A **task** is a **state** in a **workflow** that represents a single unit of work that another AWS service performs.

AWS Step Functions makes it **easier** to **orchestrate** multiple AWS services to accomplish tasks.

Step Functions allows you to **create steps** in a **process** where the output of one step becomes the input for another step, all using a visual workflow editor.



State types

Task	A single unit of work	 AWS Lambda Invoke  Amazon SNS Publish  Amazon ECS RunTask  AWS Step Functions StartExecution  AWS Glue StartJobRun
Choice	Adds branching logic	 Choice Adds if-then-else logic.
Parallel	Fork and join the data across tasks	 Parallel Adds parallel branches.
Wait	Delay for a specified time	 Wait Delays for a specified time.
Fail	Stops an execution and marks it as a failure	 Fail Stops and marks as failure.
Succeed	Stops an execution successfully	 Success Stops and marks as success.
Pass	Passes its input to its output	 Pass Transforms data or acts as placeholder.
Map	Loop and Foreach	 Map Adds a for-each loop.



Benefits of using AWS Step Functions

Build and deploy rapidly

Get started quickly with Workflow Studio, a simple drag-and-drop interface. With Step Functions, you can express complex business logic as low-code, event-driven workflows.

Write less integration code

Compose AWS resources from over 200 services including Lambda, ECS, Fargate, Batch, DynamoDB, SNS, SQS, SageMaker, EventBridge.

Build fault-tolerant and stateful workflows

Built-in try/catch, retry, and rollback capabilities deal with errors and exceptions automatically based on your defined business logic.

Designed for reliability and scale

Step Functions offers two workflow types - Standard or Express - that can be used depending on your specific use case.

Decouple application workflow logic from business logic that is another best practice of Serverless development.



AWS Step Functions

Drawbacks of using AWS Step Functions

■ Complex Configuration

Configuration with the Amazon States Language Amazon States Language is quite complex. Its syntax hard to read and modify.

■ Over Engineering when using unnecessary cases

Decoupling business logic from task sequencing can make your code harder to understand While decoupling services from the orchestration layer can make things more scalable and easier to operate.

■ Vendor Lock-in

Vendor lock-in The Amazon States Language is proprietary and can only be used on AWS. Therefore, if you decide to migrate to a different cloud provider, you'll need to re-implement the orchestration layer.



AWS Step Functions

Create a Serverless Workflow with AWS Step Functions and AWS Lambda



Step Functions > State machines > CallCenterStateMachine > Edit

Edit CallCenterStateMachine

Cancel **Start execution** Save

Definition
Define your workflow using [Amazon States Language](#). Test your data flow with the new [Data Flow Simulator](#).

Generate code snippet ▾ Format JSON

```
23  {
24      "Variable": "$.Status",
25      "NumericEquals": 1,
26      "Next": "Close Case"
27  },
28  {
29      "Variable": "$.Status",
30      "NumericEquals": 0,
31      "Next": "Escalate Case"
32  }
33  ],
34 },
35 "Close Case": {
36     "Type": "Task",
37     "Resource": "arn:aws:lambda:us-east-
1:         :function:CloseCaseFunction",
38     "End": true
39 },
40 "Escalate Case": {
41     "Type": "Task",
42     "Resource": "arn:aws:lambda:us-east-
1:         :function:EscalateCaseFunction",
43     "Next": "Fail"
44 },
45 "Fail": {
46     "Type": "Fail",
```

Workflow Studio **New**

```
graph TD
    Start((Start)) --> OpenCase[Open Case]
    OpenCase --> AssignCase[Assign Case]
    AssignCase --> WorkOnCase[Work on Case]
    WorkOnCase --> IsCaseResolved{Is Case Resolved}
    IsCaseResolved --> EscalateCase[Escalate Case]
    EscalateCase --> CloseCase[Close Case]
    EscalateCase --> Fail[Fail]
    CloseCase --> End((End))
    Fail --> End
```

Amazon Step Functions CLI Commands and (Python) Boto3

joinCommunity2023 / learning / 21_step_function / README.md

weder96 learning serverless with cli commands

Preview Code Blame 123 lines (85 loc) · 3.23 KB Code 55% faster with GitHub Copilot

Step Functions

CLI

create-state-machine

```
aws stepfunctions create-state-machine \
--name HasCreateCliStateMachine \
--definition "file://stateMachineDefinition.json" \
--role-arn arn:aws:iam::710304818543:role/service-role/StepFunctions-HelloWorld-role-6df6dc83
```

list state machines

```
aws stepfunctions list-state-machines
```

Execute state machines

```
aws stepfunctions start-execution --state-machine-arn arn:aws:states:us-east-1:710304818543:stateMachine:HasCreateCli
```



AWS Step Functions



AWS Lambda Overview



AWS Lambda Overview

Serverless on AWS, Build and run applications without thinking about servers

The **most popular serverless** compute platform that is using millions of customer

Running billions of invocations all over the world

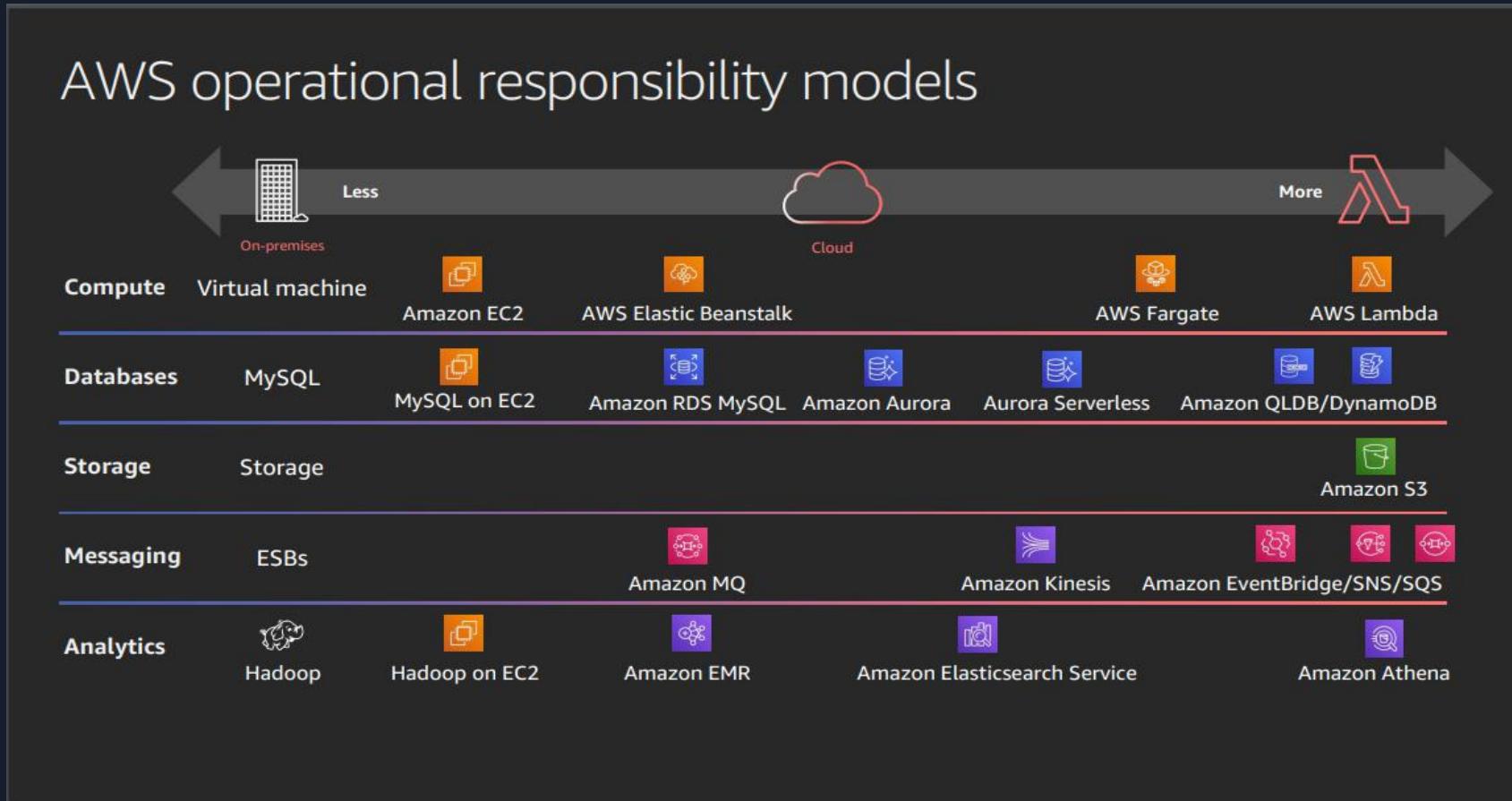
Compute service that runs code without thinking any servers or underlying services

Serverless function that you only **responsible for your actual code**.



https://aws.amazon.com/serverless/?nc1=h_ls
<https://aws.amazon.com/blogs/compute/serverless-icymi-q2-2023/>

Serverless Explained : AWS Operational Responsibility Model



AWS Lambda Invocation Types

Triggered lambda functions with different AWS Lambda Invocation Types

AWS Lambda has 3 Invocation Types;

Lambda Synchronous invocation

Lambda Asynchronous invocation

Lambda Event Source Mapping with polling invocation

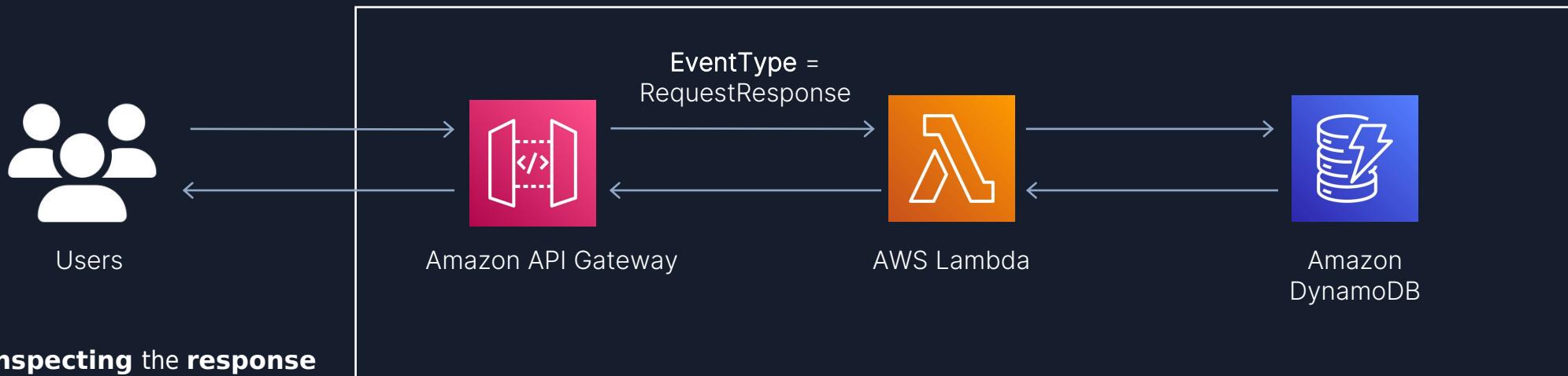


AWS Lambda Synchronous Invocation

Execute immediately when you perform the Lambda Invoke API call.
Wait for the function to process the function and return back to response.

API Gateway + Lambda + DynamoDB

Invocation-type flag should be “**RequestResponse**”



- Responsible for **inspecting** the **response** and **determining** if there was an **error** and **decide** to **retry** the invocation

- Example of synchronous invocation using the AWS CLI:

```
aws lambda invoke --function-name MyLambdaFunction --invocation-type RequestResponse --payload '{ "key": "value" }'
```

- Triggered **AWS services** of **synchronous invocation**: ELB (Application Load Balancer), Cognito, Lex, Alexa, API Gateway, CloudFront, Kinesis Data Firehose

AWS Lambda Asynchronous Invocation

Lambda **sends** the **event** to a **internal queue** and returns a success response without any additional information

Separate process **reads events** from the **queue** and **runs** our lambda function

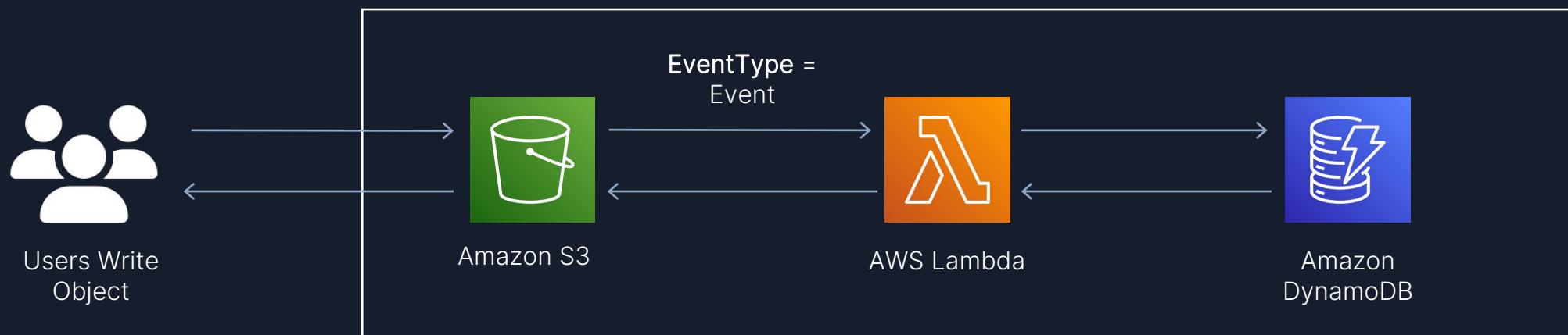
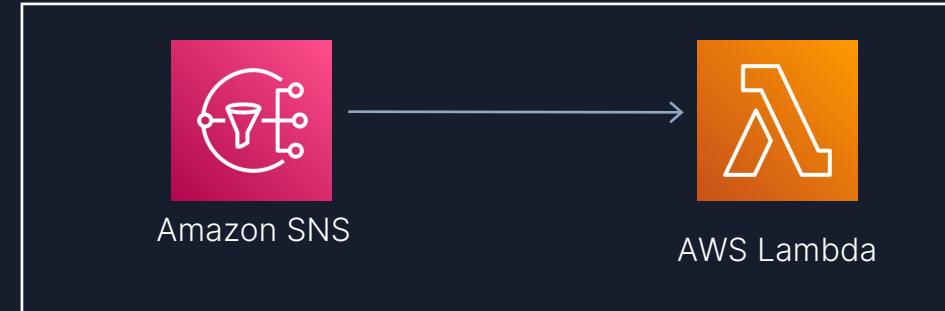
S3 / SNS + Lambda + DynamoDB

Invocation-type flag should be “**Event**”

AWS Lambda sets a **retry policy**

Retry Count = 2

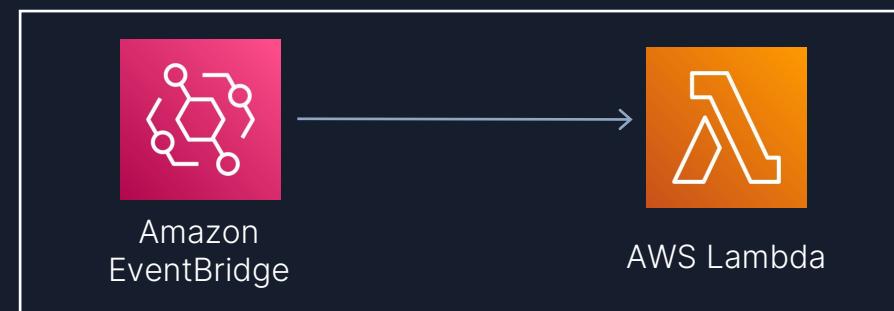
Attach a Dead-Letter Queue (DLQ)



Example of asynchronous invocation using the AWS CLI:

```
aws lambda invoke --function-name MyLambdaFunction --invocation-type Event --payload '{ "key": "value" }'
```

Triggered **AWS services** of **asynchronous invocation**; S3, EventBridge, SNS, SES, CloudFormation, CloudWatch Logs, CloudWatch Events, CodeCommit

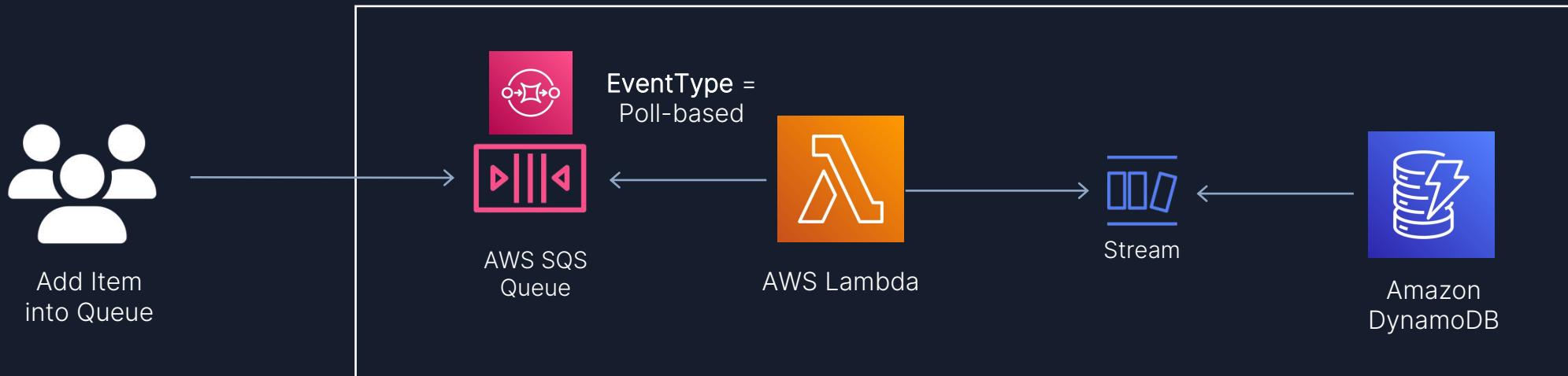


AWS Lambda Event Source Mapping with Polling Invocation

Pool-Based invocation model allows us to integrate with **AWS Stream and Queue based services**.

Lambda will poll from the **AWS SQS** or **Kinesis streams**, retrieve records, and invoke functions.

Data stream or queue are read in batches,
The function receives multiple items when execute function.
Batch sizes can configure according to service types



SQS + Lambda

Stream based processing with **DynamoDB Streams + Lambda**

Triggered AWS services of **Event Source** Mapping invocation; **Amazon Kinesis**,
DynamoDB, **Simple Queue Service (SQS)**

Lambda Function Code

AWS Lambda runs instances of **your function to process events**.

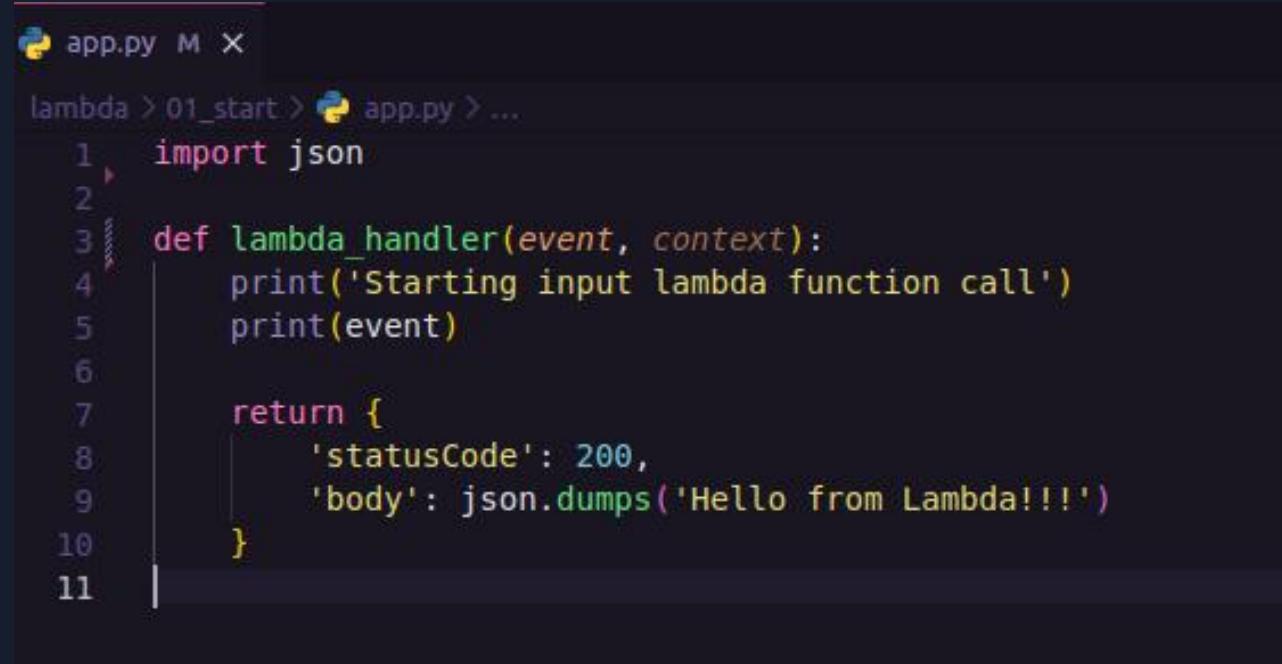
Invoke function directly **using the Lambda API**, or configure an AWS service or resource to invoke your function.

Lambda function **has code to process the events that you pass** into the function or that other AWS services send to the function with event json object.

The event object contains all the information about the **event that triggered this Lambda**.

The **context object contains info about** the runtime our Lambda function

Return the function **with the results**



```
app.py M X
lambda > 01_start > app.py > ...
1 import json
2
3 def lambda_handler(event, context):
4     print('Starting input lambda function call')
5     print(event)
6
7     return {
8         'statusCode': 200,
9         'body': json.dumps('Hello from Lambda!!!')
10    }
11
```



Key concepts of Lambda Function Code

Runtime: select the runtime as part of configuring the function, and Lambda loads that runtime when initializing the environment.

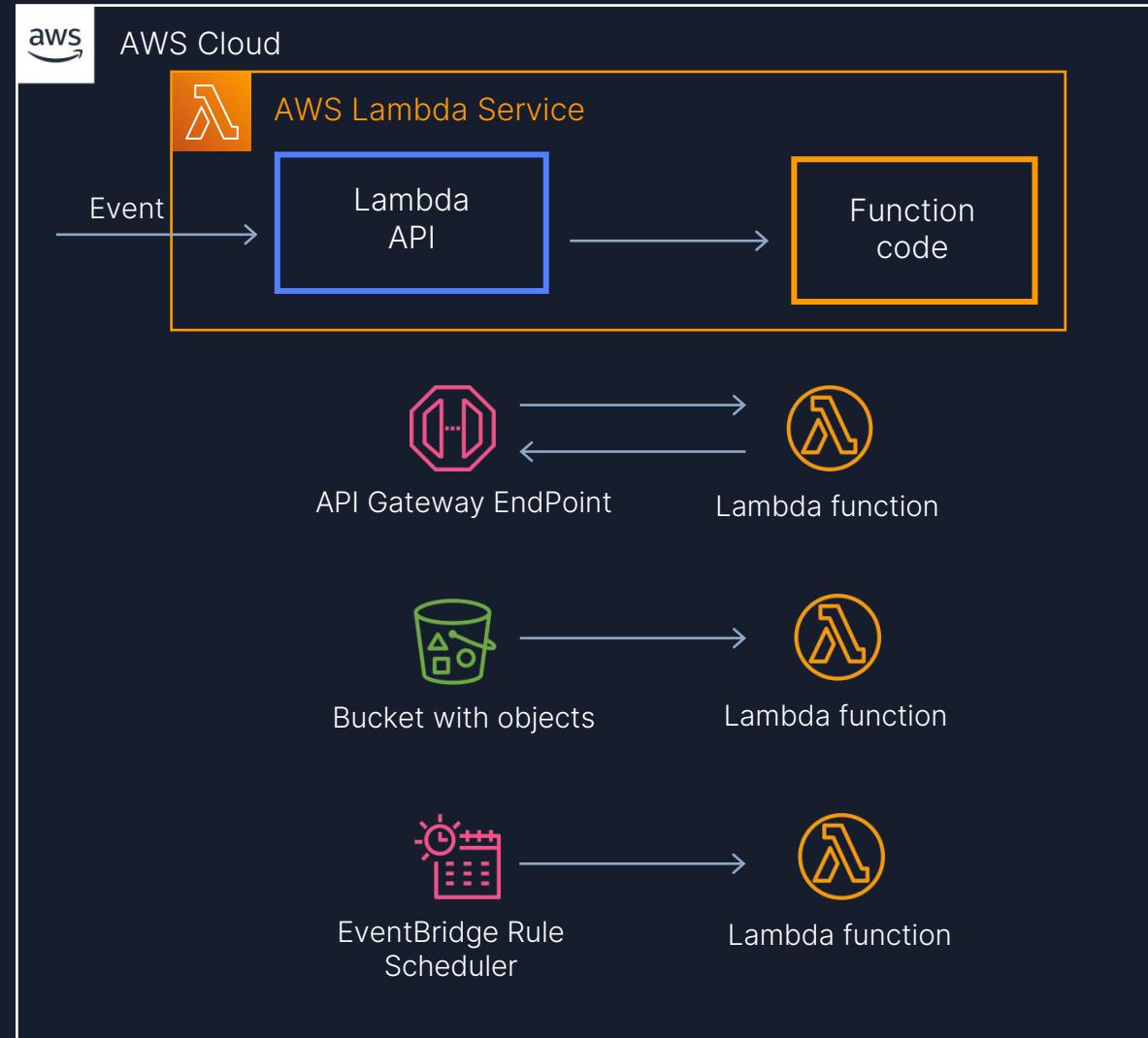
Handler: function runs starting at the handler method.

Function: is a resource that you can invoke to run your code in Lambda.

Trigger: is a resource or configuration that invokes a Lambda function.

Event: is a JSON-formatted document that contains data for a Lambda function to process.

Execution environment: provides a secure and isolated runtime environment for your Lambda function.

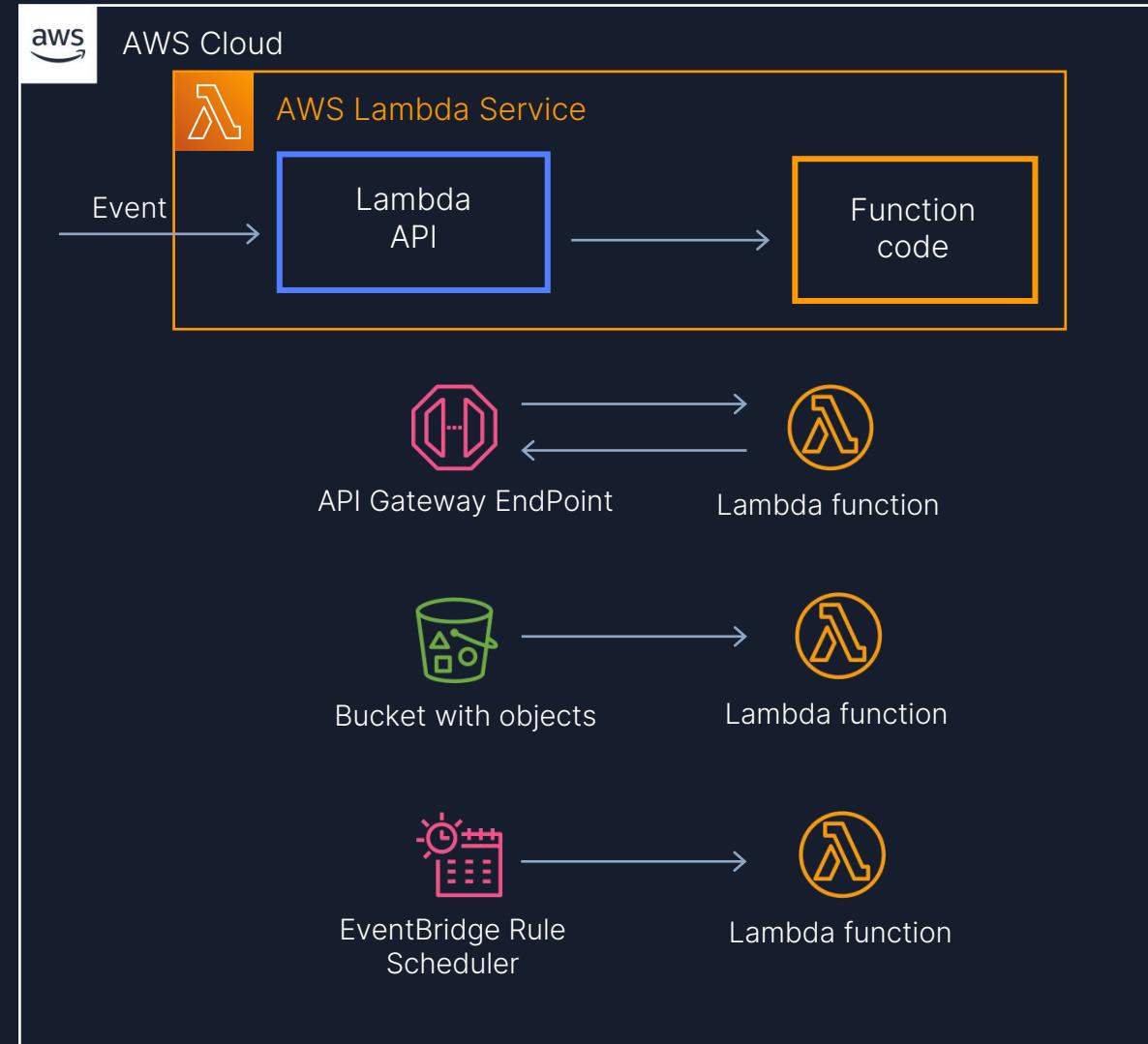


Key concepts of Lambda Function Code

Layer: can contain libraries, a custom runtime, data, or configuration files. Using layers reduces the size of uploaded deployment archives and makes it faster to deploy your code.

Concurrency: is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request.

Destination: is an AWS resource where Lambda can send events from an asynchronous invocation. configure a destination for events that fail processing like setting DLQ for Lambda fails.



Lambda Execution Environment Lifecycle

Init Phase

3 main tasks; Extension, Runtime and Function init.
Ends when the runtime and all extensions are ready.
Limited to 10 seconds. If not complete, Lambda retries the Init phase.

Invoke Phase

Lambda invokes the function handler.
Lambda prepares to handle another function invocation. Timeout setting limits the duration of the entire Invoke phase. Next API request invokes another function.

Shutdown Phase

Happens if the Lambda function doesn't receive any invocations. Lambda shuts down the runtime, removes the environment. Lambda sends a Shutdown event to each extension.

Lifecycle Lambda Execution



Reducing Cold Start

SNAPSTART

Init during deployment



(optional)

Resume

Invoke

Invoke

first request →

Resume
Snapshot

Post
Snapshot
Hook

(optional)

Code
execution

Code
execution

Concurrency on AWS Lambda: Reserved and Provisioned

Memory and Timeout Configuration Optimization

The memory setting determines how much CPU power will receive for our lambda function. And higher CPU power decrease the function execution time.

When function reaches to timeout value and not finished, lambda forcibly stop the execution.

Concurrency: The number of instances of your function that are active.

Reserved Concurrency

Provisioned Concurrency

When your function is invoked, **Lambda allocates an instance of it to process** the incoming event.

If the function is invoked again when the request is still being processed, another instance is allocated, which increases the function's concurrency.



AWS Lambda Execution Role

AWS Lambda Permissions

Lambda Execution Role
Resource-based policy

Lambda Execution Role

IAM role that Lambda has permissions to assume when invoking lambda function.

Create an execution role when creating a new lambda function, and after that we can also modify the policies **associated with the IAM role**.

if you have additional targets from your lambda function
performing crud operations on DynamoDB table
sending notification to SNS
retrieve messages from queue or streams

Lambda function's execution role required permissions to interact with those AWS services

Grant least privilege access to your Lambda execution role



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

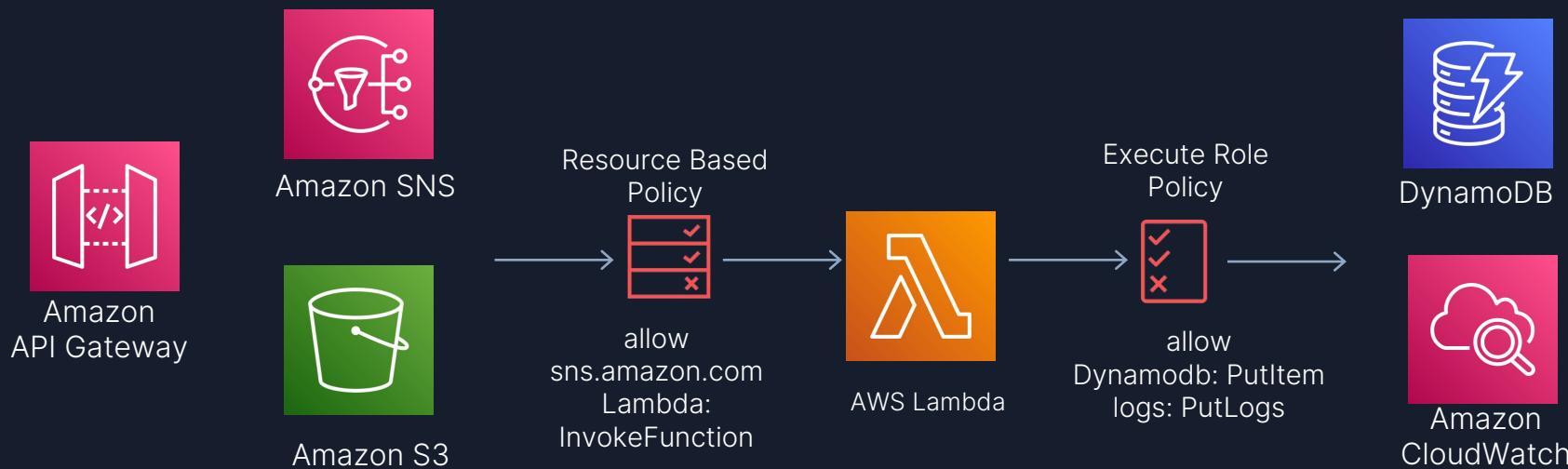
AWS Lambda Resource-based Policy

Lambda Resource-based policy

When any AWS service invokes Lambda function sync or async way. It lets you grant usage permission to other AWS accounts or organizations on a per-resource basis. Also use a resource-based policy to allow an AWS service to invoke your function on your behalf.

API Gateway that targets to Lambda function, we should add resource-based policy permission to invoke lambda function from API gateway.

Amazon S3 upload event triggers to lambda function asynchronously, so we should also add Resource-based policy into our Lambda function grants S3 invocation.





CLI



LAMBDAS

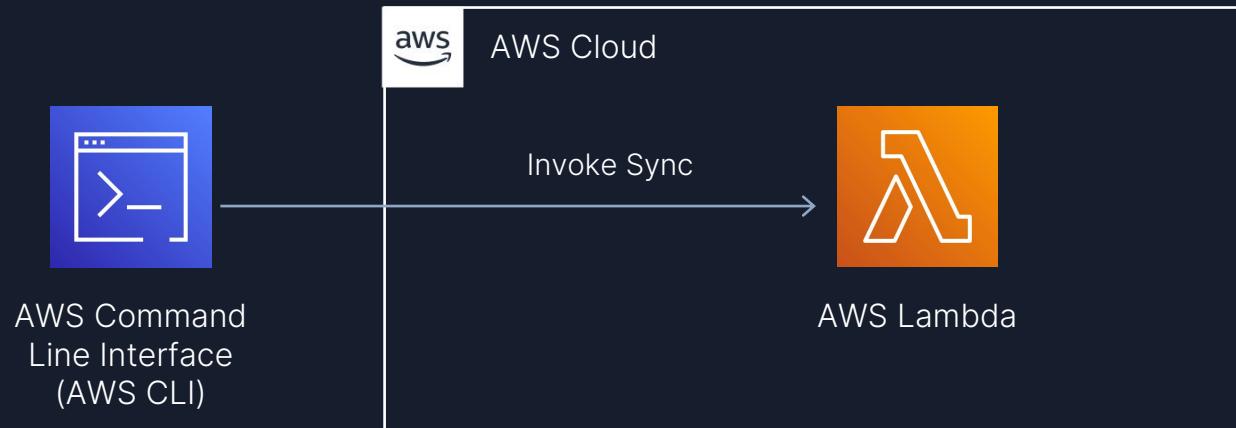
Invoke Lambda functions



Invoke Lambda functions with the AWS CLI

We can invoke Lambda functions directly using the:

- Lambda console
- Lambda API
- AWS SDK
- AWS Command Line Interface (AWS CLI)



Synchronous invocation, you wait for the function to process the event and return a response.

Asynchronous invocation, Lambda queues the event for processing and returns a response immediately.

Create AWS Lambda functions with AWS CLI

Steps of Create AWS Lambda functions with AWS CLI;

Create the execution role

Create function code

Create a deployment package with zip function code

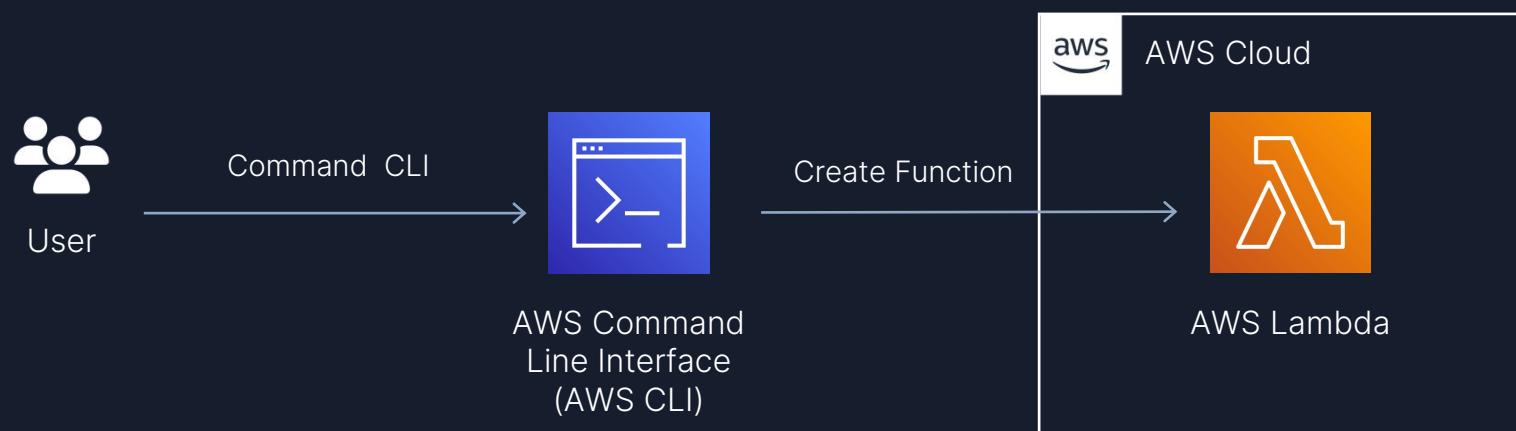
Create Lambda Function with AWS CLI

Invoke Lambda Function

Check logs

We will Check Lambda Function details with AWS CLI.

We will Clean up resources



Update AWS Lambda Function Code using AWS CLI

Steps of Update AWS Lambda functions with AWS CLI;

1. Develop lambda code to update one
2. zip function code
3. update lambda function with cli
4. invoke updated lambda function with cli

We will Clean up resources



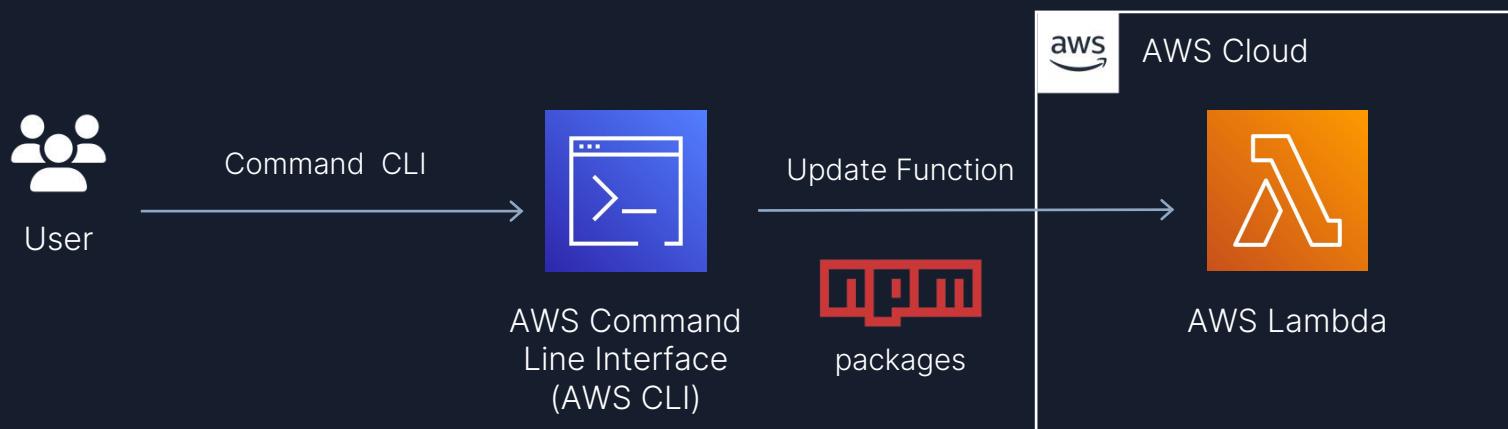
Update AWS Lambda Function with Dependencies using AWS CLI

Lambda function depends on libraries, we can use npm to include them in our deployment package.

Add any external package with using "npm install" command

Steps of Update AWS Lambda functions with AWS CLI;

1. Install required Dependencies into node_modules
2. zip function code
3. update lambda function with cli
4. invoke updated lambda function with cli



Using AWS Lambda Environment Variables

Use environment variables to adjust your function's behavior without updating code.

1. Environment variable is a pair of strings that is stored in a function's version-specific configuration.
2. Configure environment variables with using AWS Management Console.
3. Configure environment variables with the Lambda API with cli and sdk libraries.

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.](#)

dbHost	test_db.hJ3t4WTkmh9e.ap-southeast-1.rds.amazonaws.com	Remove
dbUser	james	Remove
dbPassword	bP5Qmnbs96ZTLBScGEGr	Remove
dbName	test_database	Remove
Key	Value	Remove

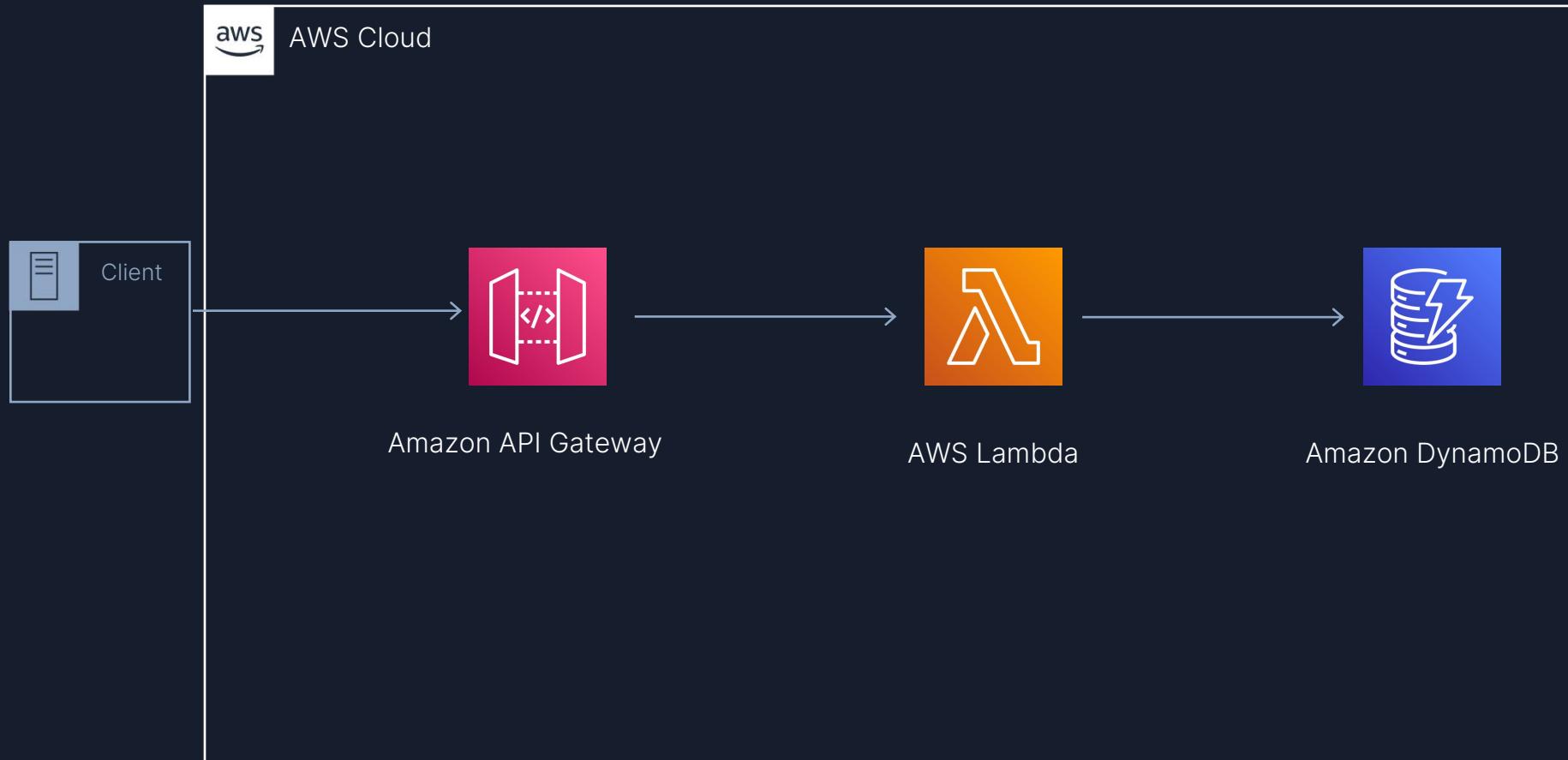
▶ [Encryption configuration](#)



Pattern: The comfortable “REST”



Pattern: The comfortable “REST”



OPERATIONS

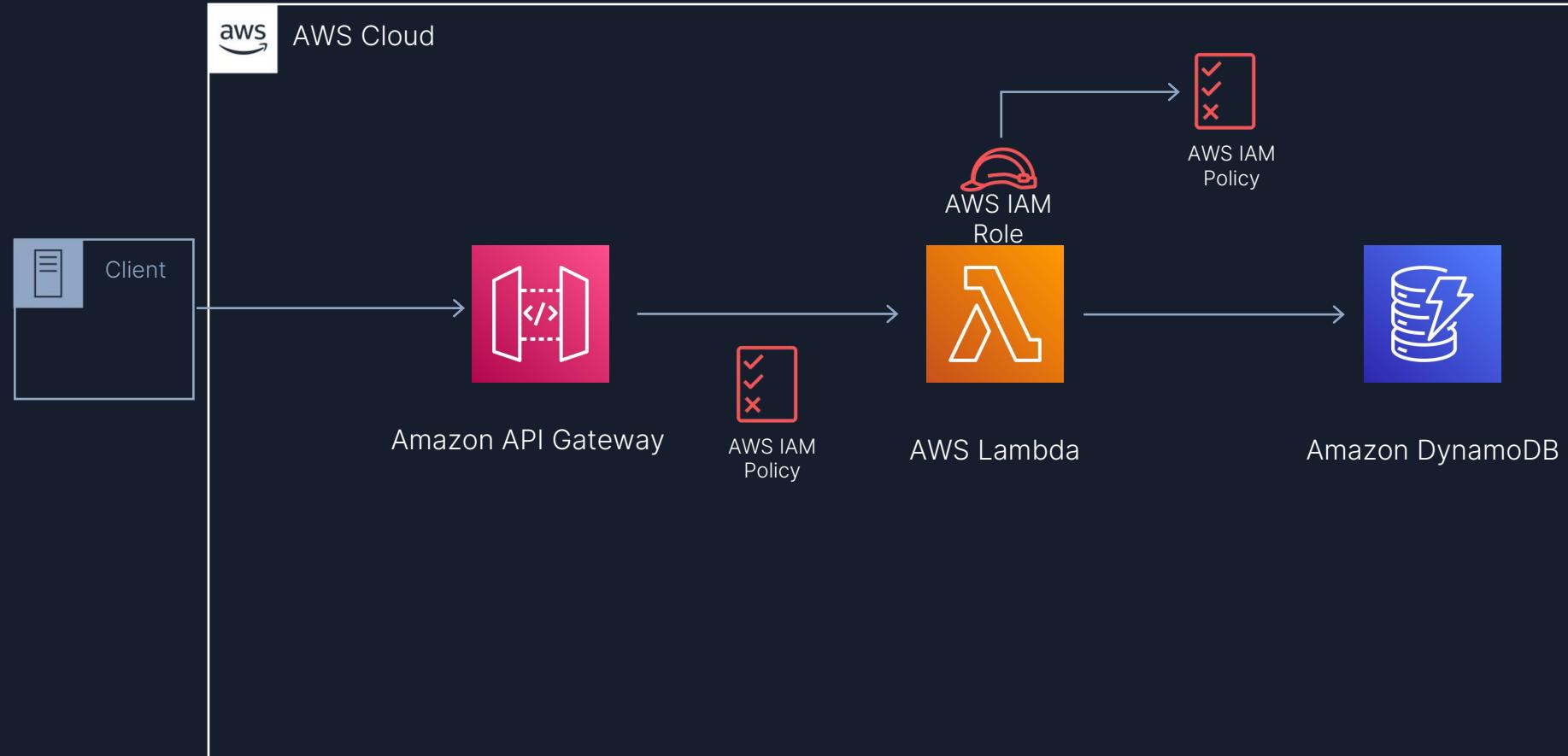
RELIABILITY

SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”



OPERATIONS

RELIABILITY

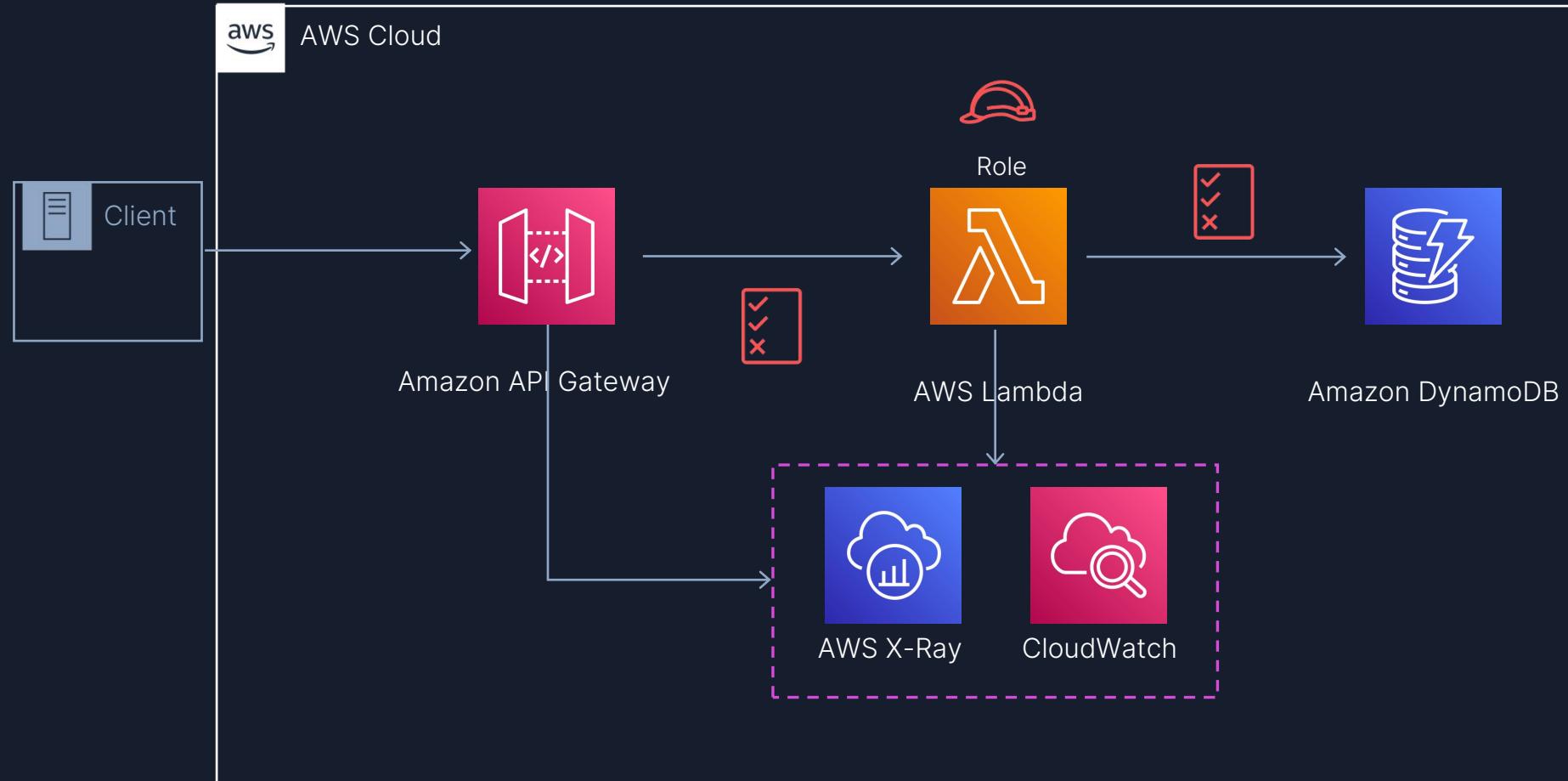
SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”

Best practices



Enable access logs, structure logs and instrument your code

Create metrics async with [CloudWatch Embedded Metric Format \(EMF\)](#)

OPERATIONS

RELIABILITY

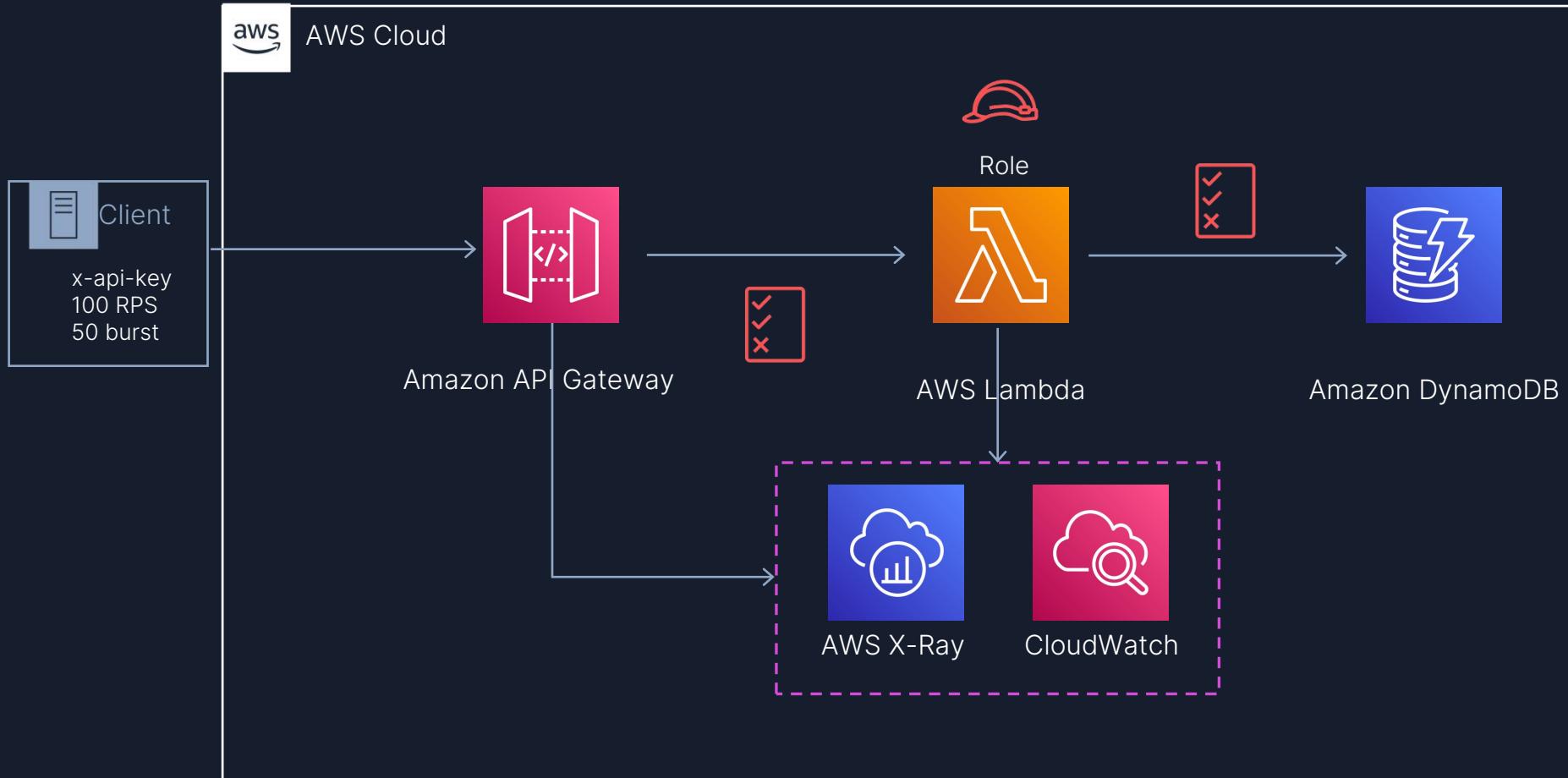
SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”

Best practices



Enable access logs, structure logs and instrument your code

Create metrics async with [CloudWatch Embedded Metric Format \(EMF\)](#)

Regulate inbound access rates

OPERATIONS

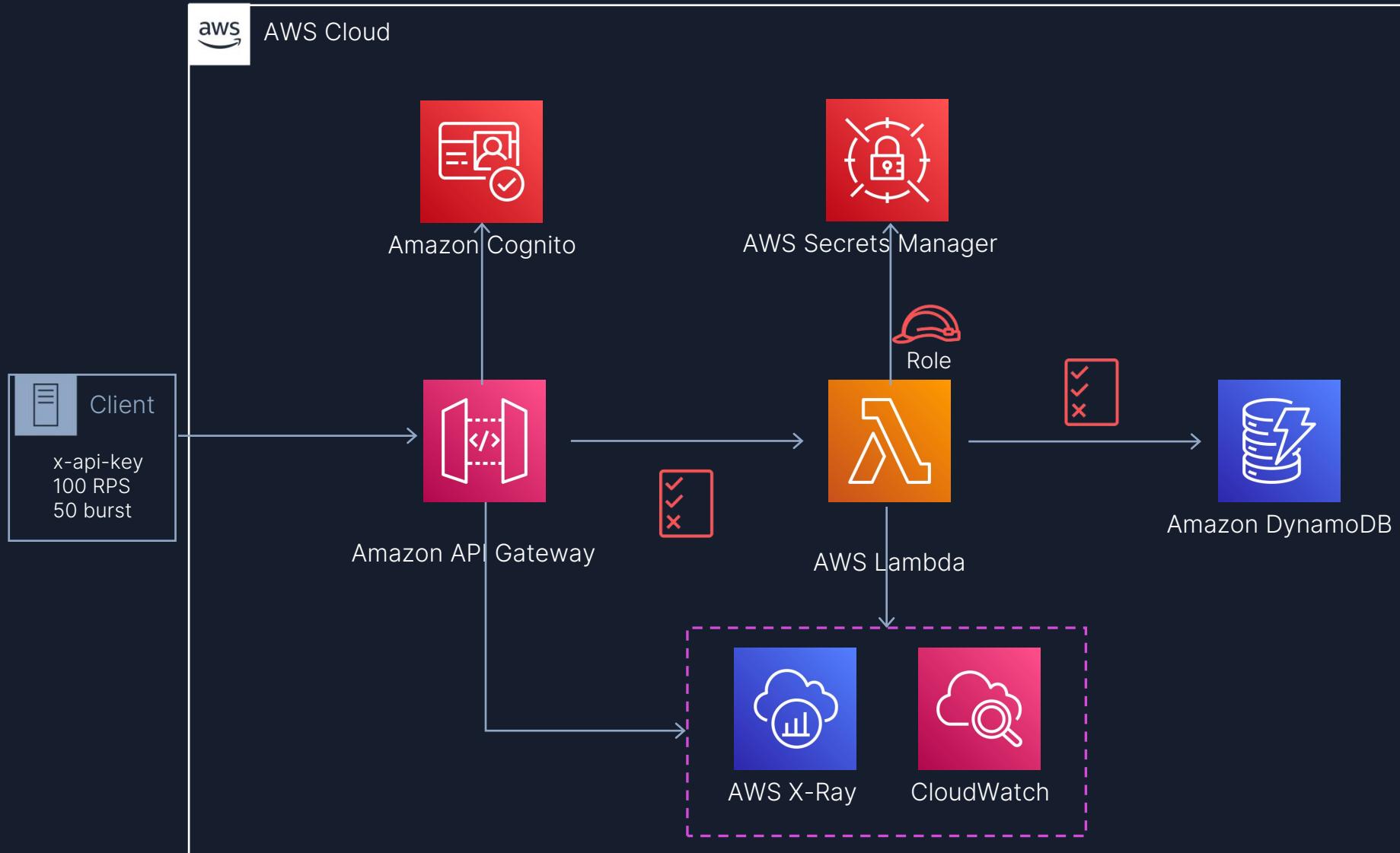
RELIABILITY

SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”



Best practices

- Enable access logs, structure logs and instrument your code
- Create metrics async with [CloudWatch Embedded Metric Format \(EMF\)](#)
- Regulate inbound access rates
- Authorize consumers. Manage secrets with [AWS Secrets Manager](#)

OPERATIONS

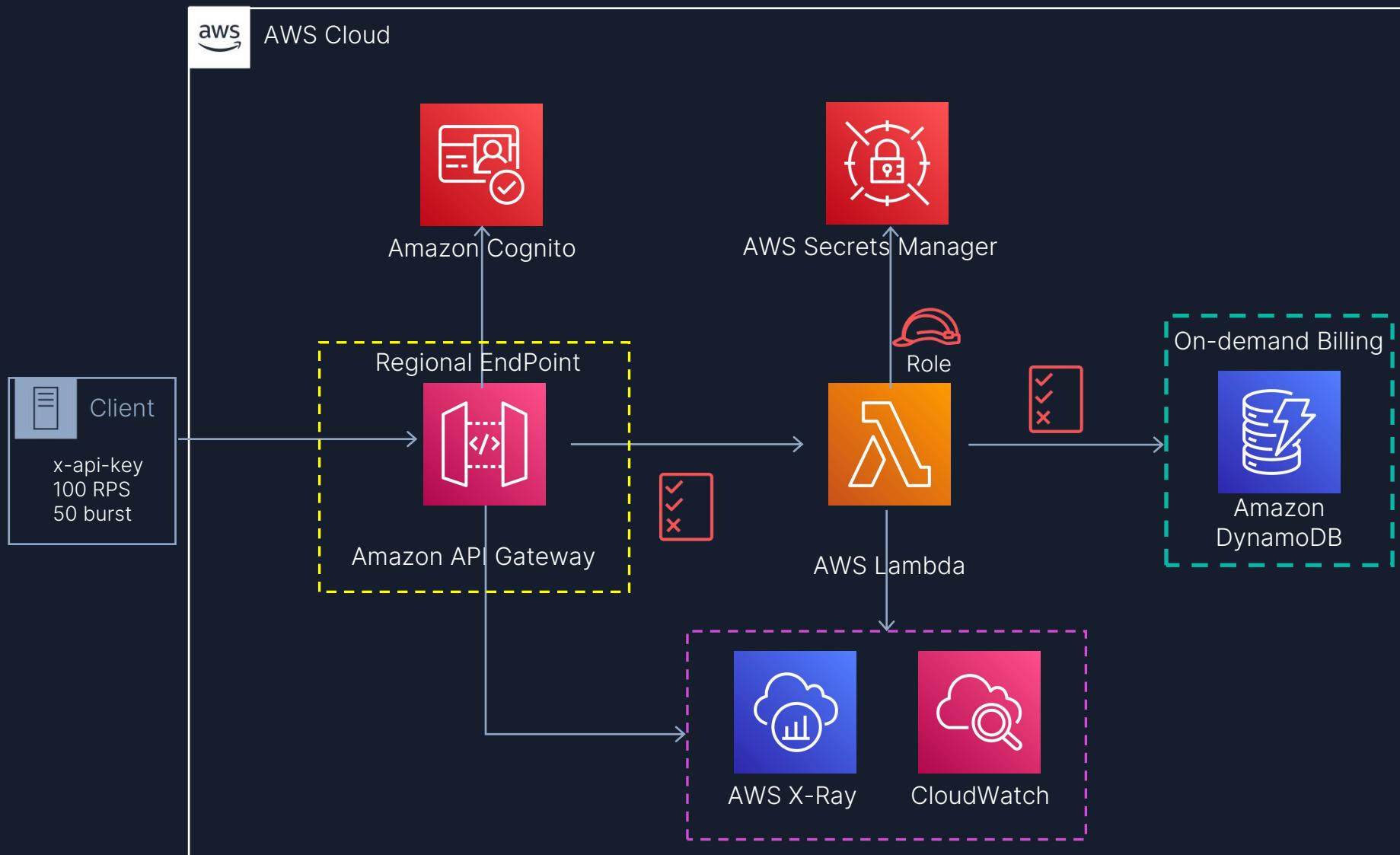
RELIABILITY

SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”



Best practices

- Enable access logs, structure logs and instrument your code
- Create metrics async with [CloudWatch Embedded Metric Format \(EMF\)](#)
- Regulate inbound access rates
- Authorize consumers. Manage secrets with [AWS Secrets Manager](#)
- On-demand tables support up to 40K read/write request units
- Regional endpoints support HTTP2

OPERATIONS

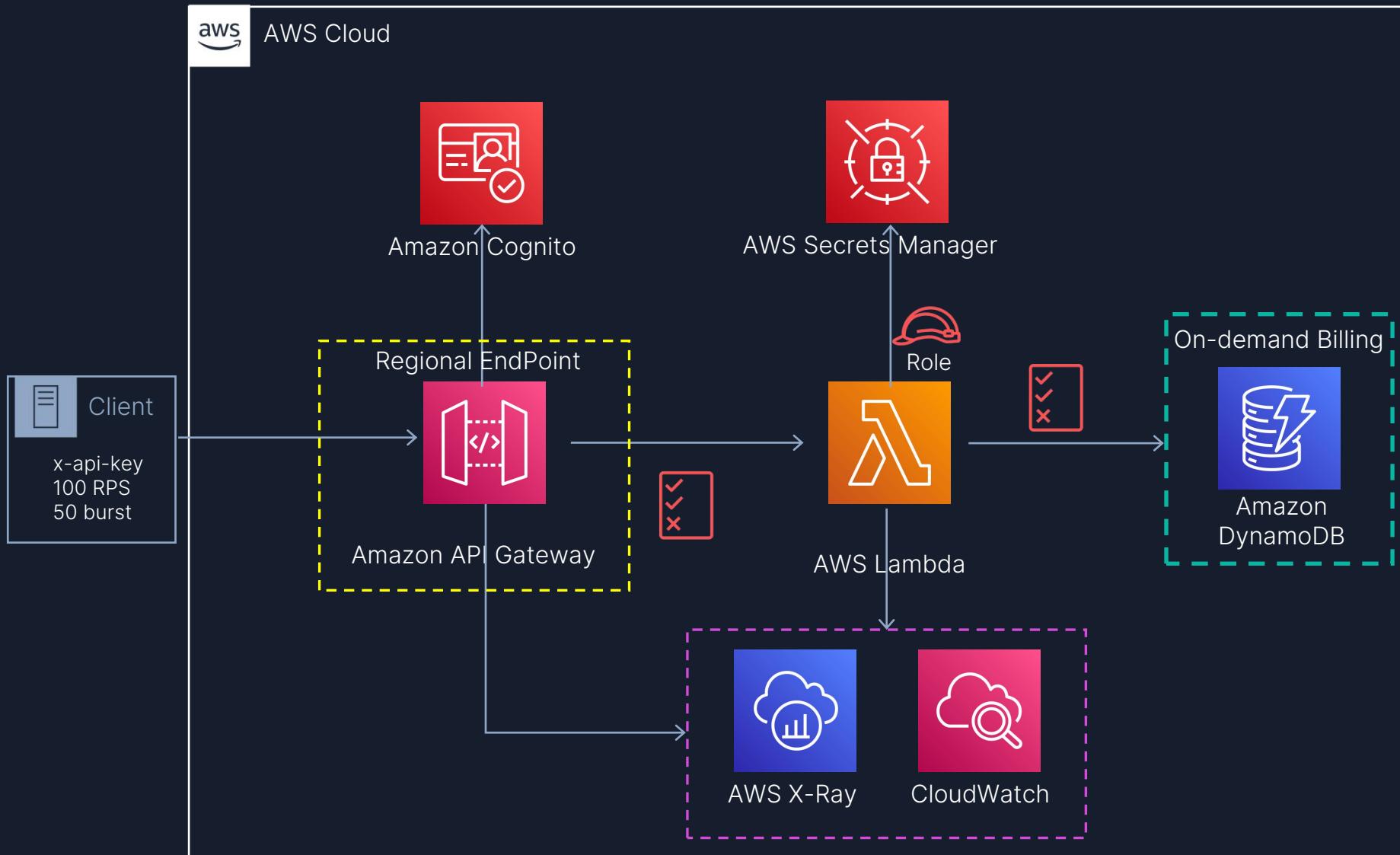
RELIABILITY

SECURITY

PERFORMANCE

COST

Pattern: The comfortable “REST”



Best practices

- Enable access logs, structure logs and instrument your code
- Create metrics async with [CloudWatch Embedded Metric Format \(EMF\)](#)
- Regulate inbound access rates
- Authorize consumers. Manage secrets with [AWS Secrets Manager](#)
- On-demand tables support up to 40K read/write request units
- Regional endpoints support HTTP2
- Use Lambda Power Tuning for perf/cost tuning

OPERATIONS

RELIABILITY

SECURITY

PERFORMANCE

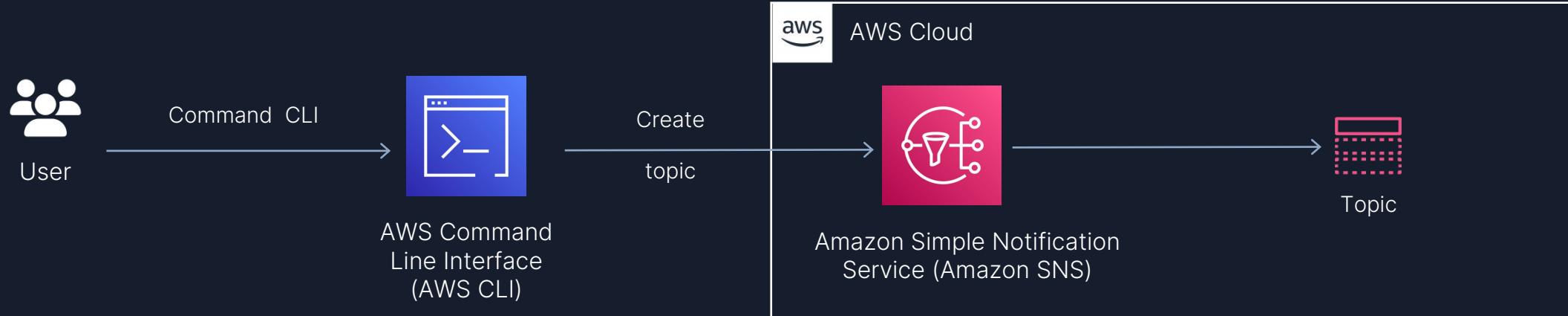
COST



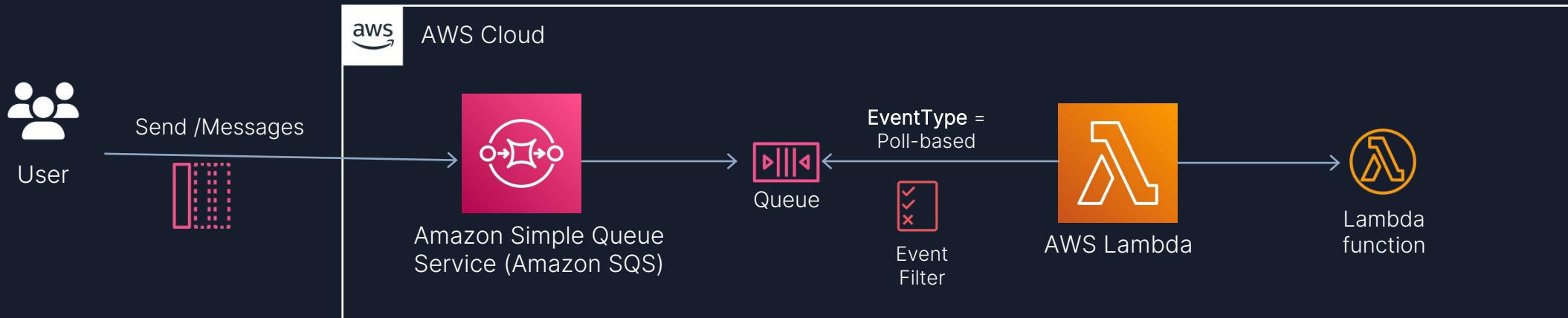
Architecture Serverless



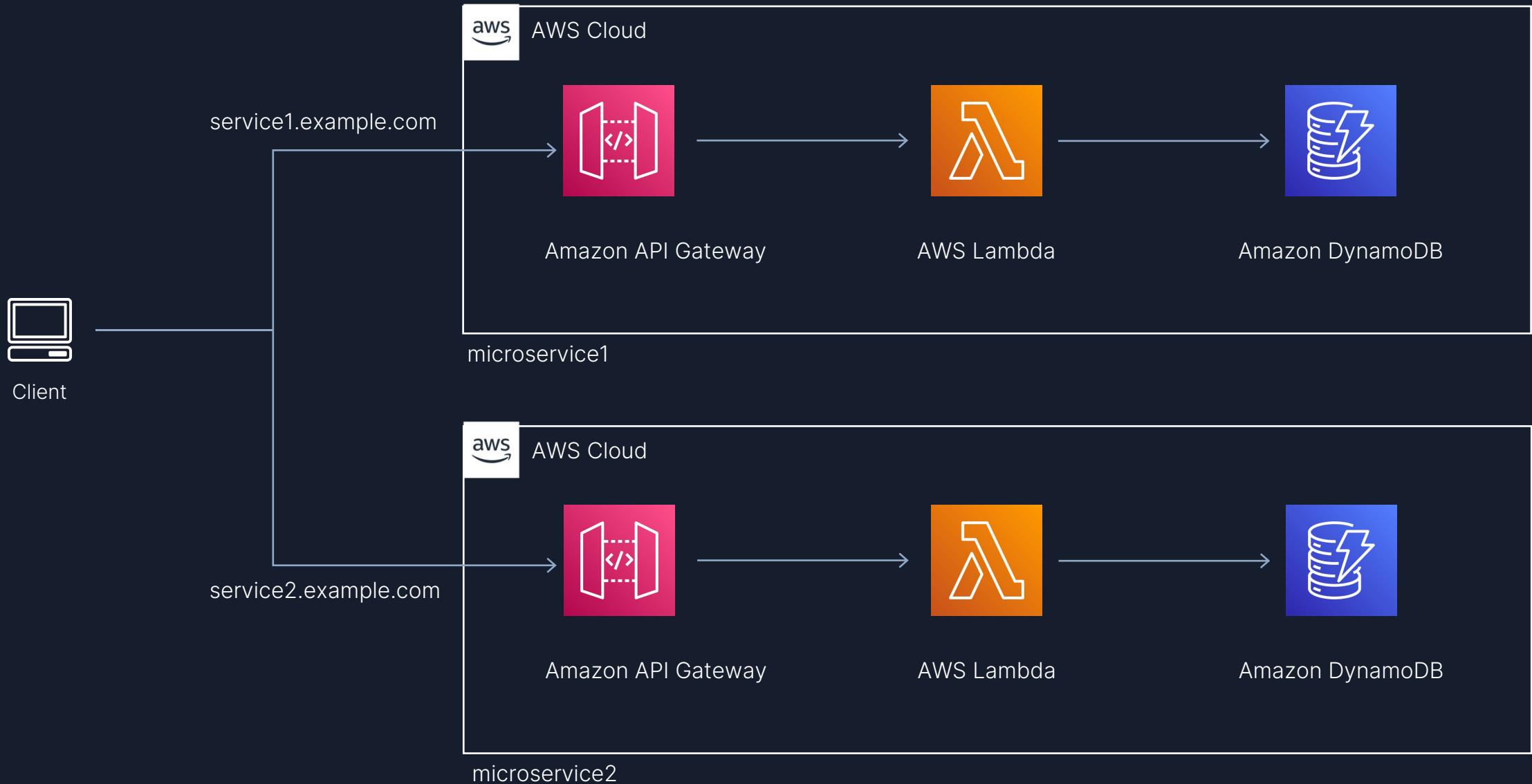
Amazon SNS with AWS Management Console



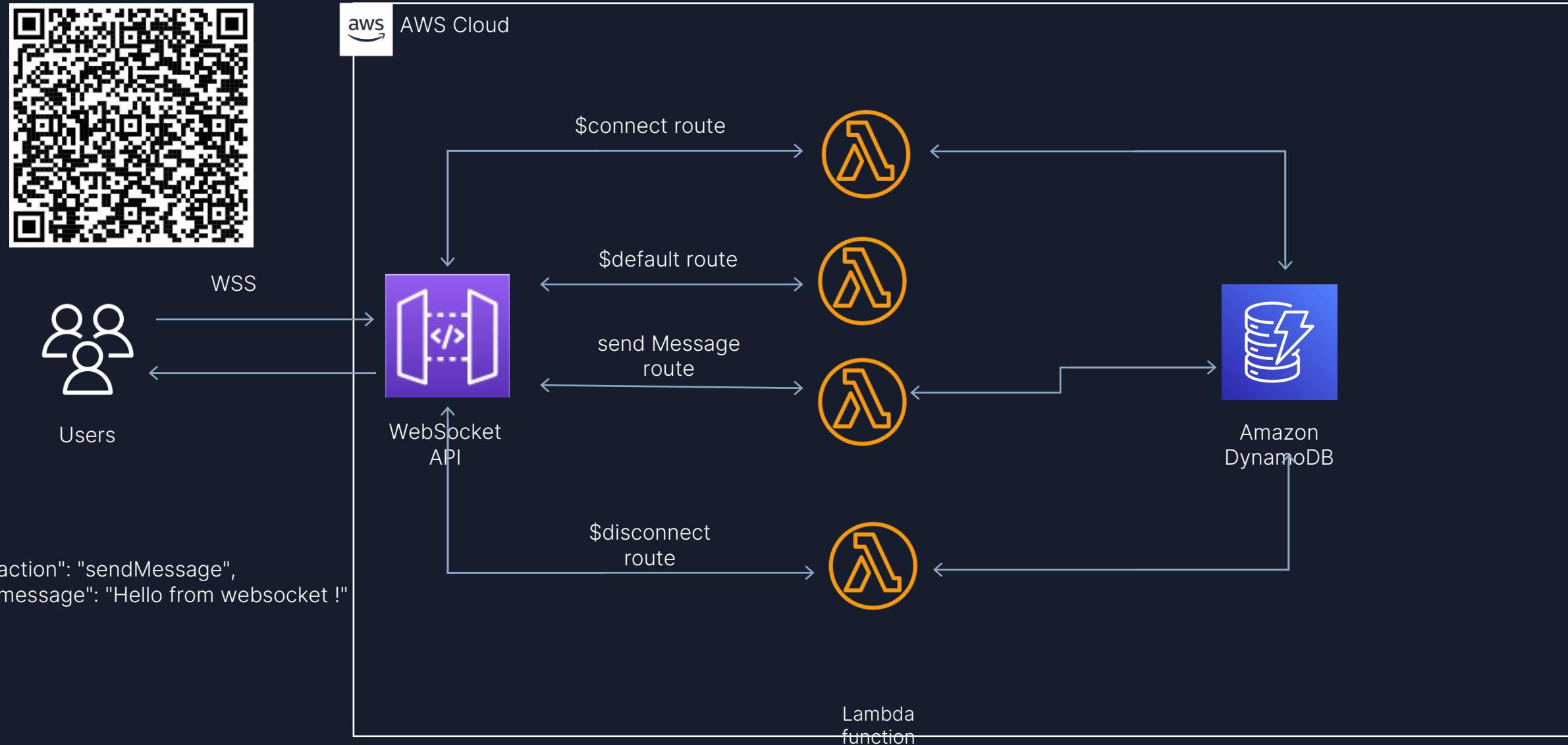
Amazon SQS Queue Polling From AWS Lambda



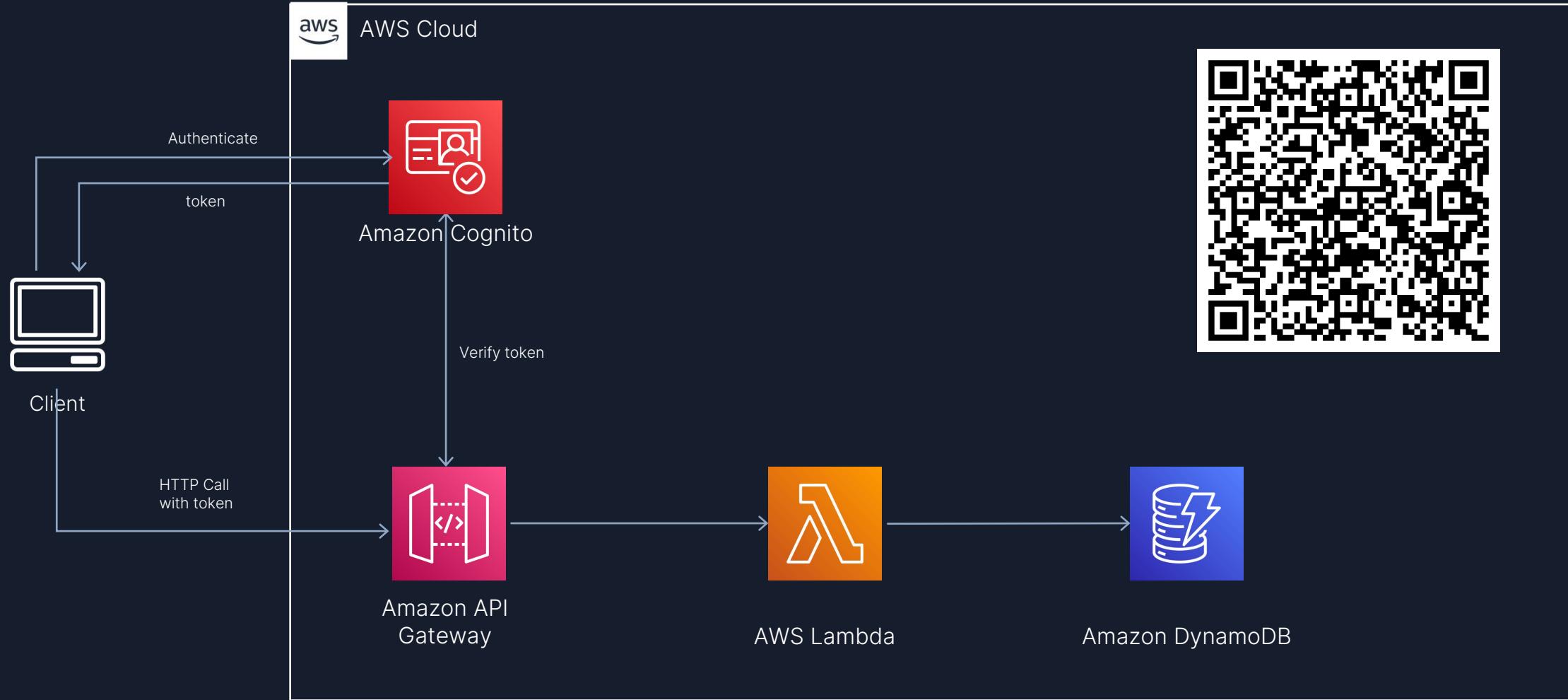
Microservices “REST”



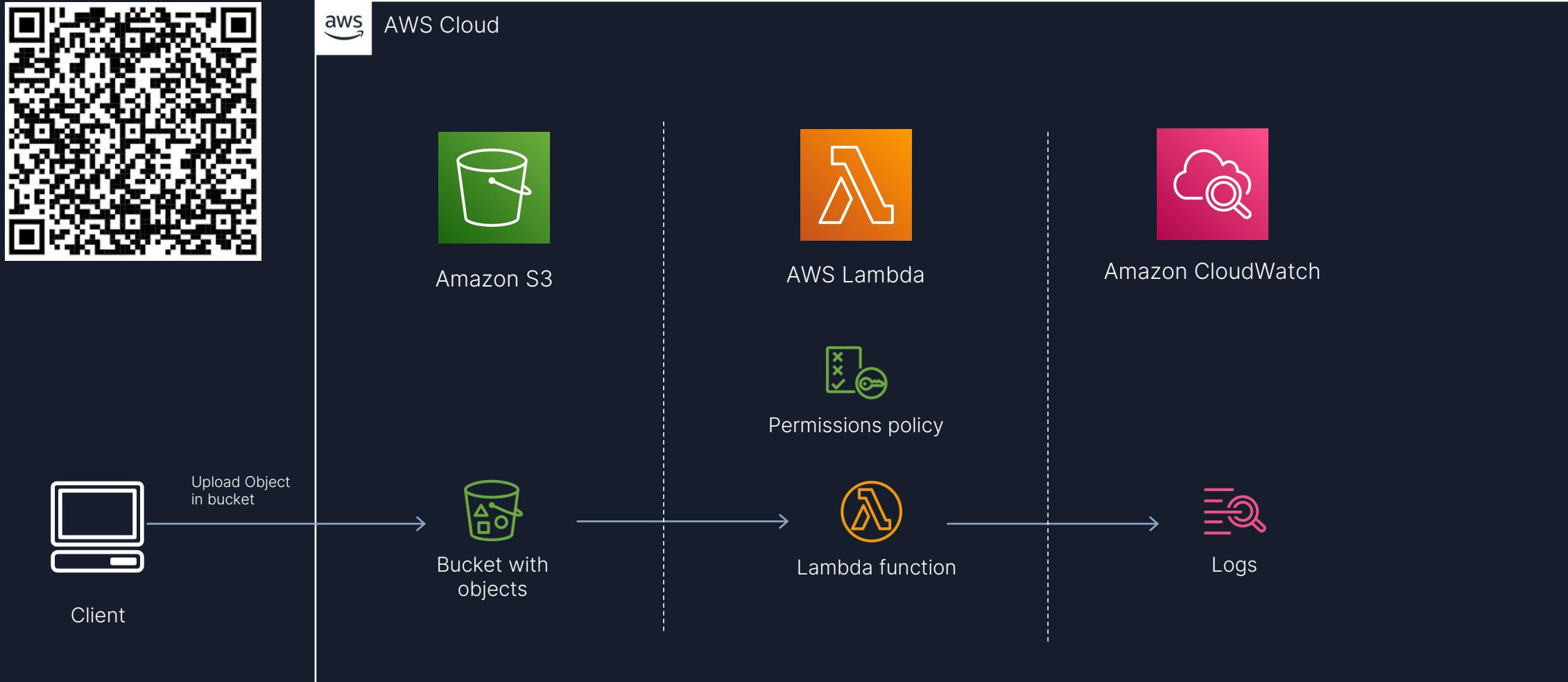
Build Serverless Chat App with a WebSocket API and Lambda



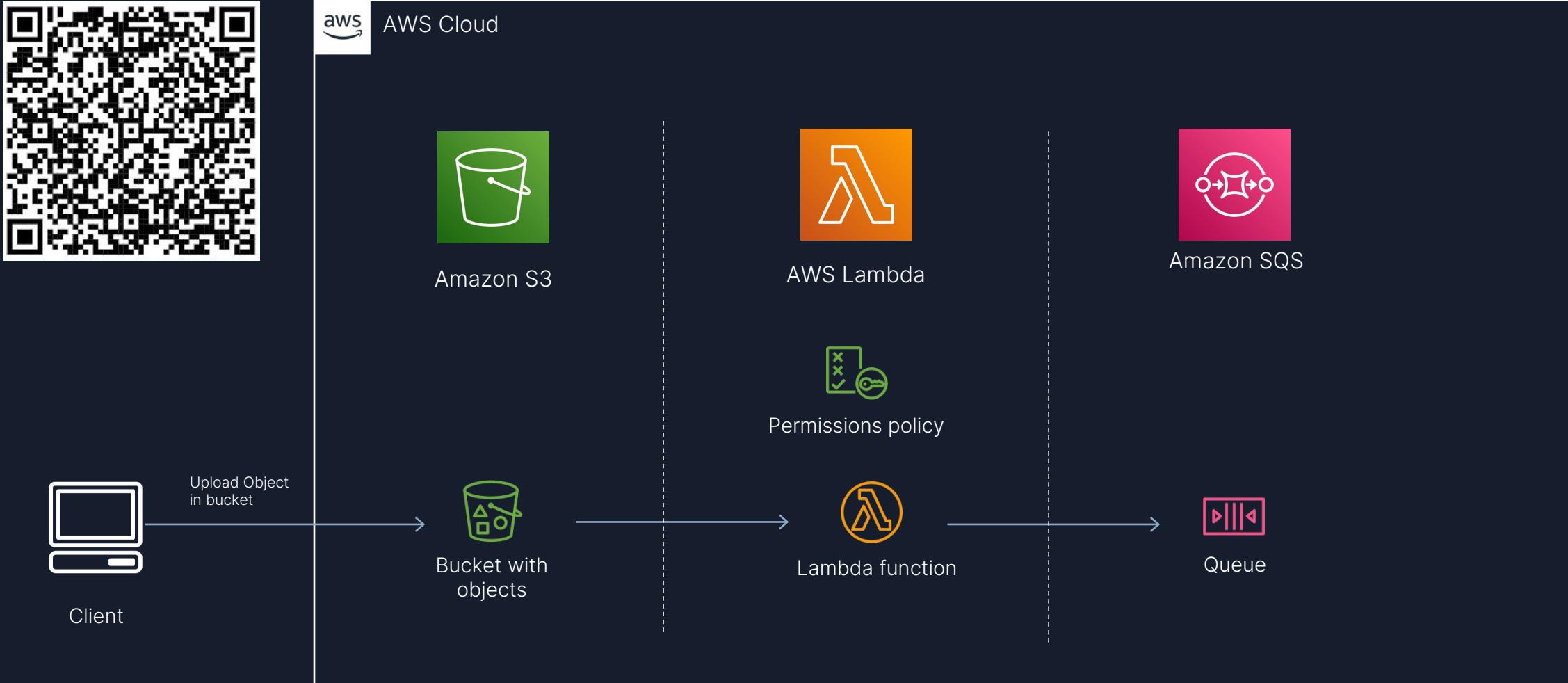
Secure your API Gateway with Amazon Cognito User Pools



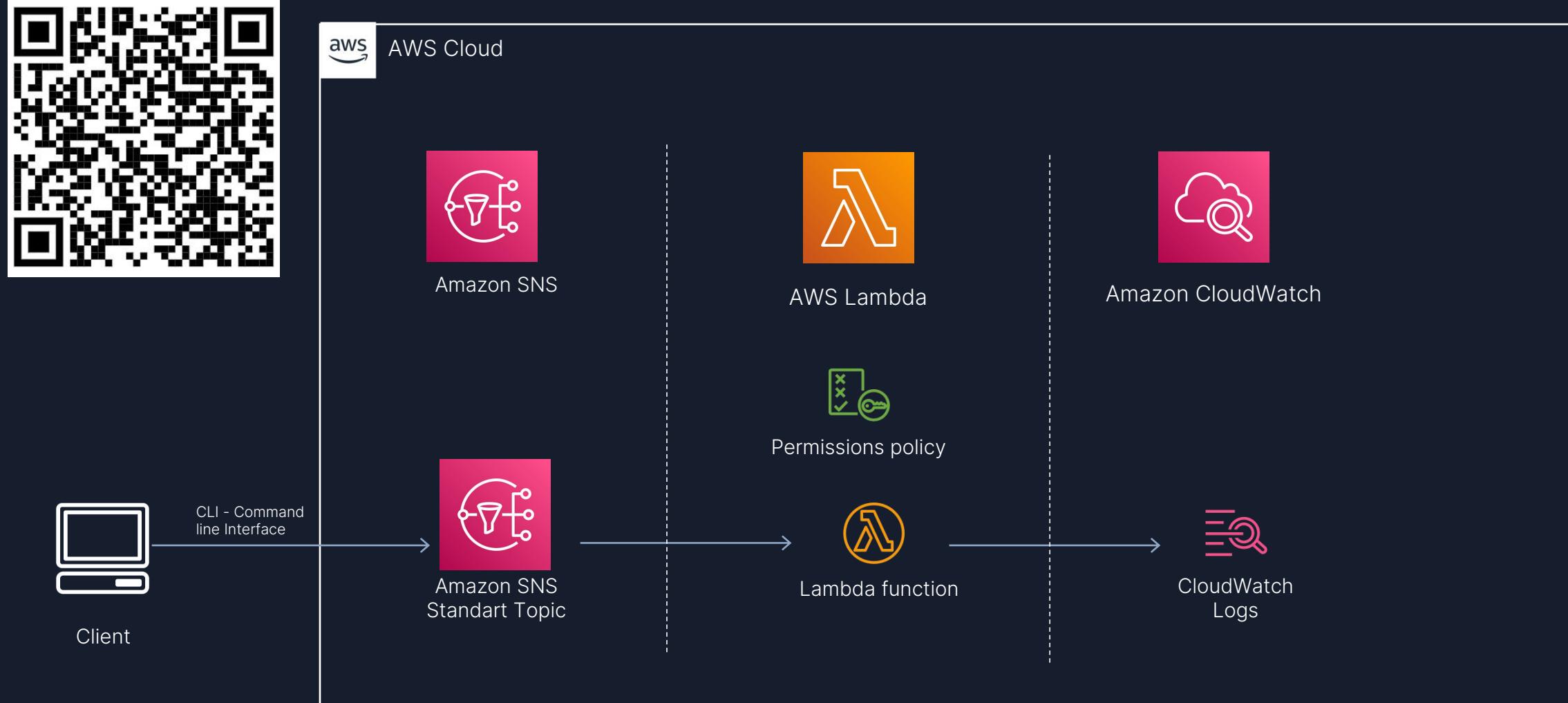
Using an Amazon S3 trigger to invoke a Lambda function



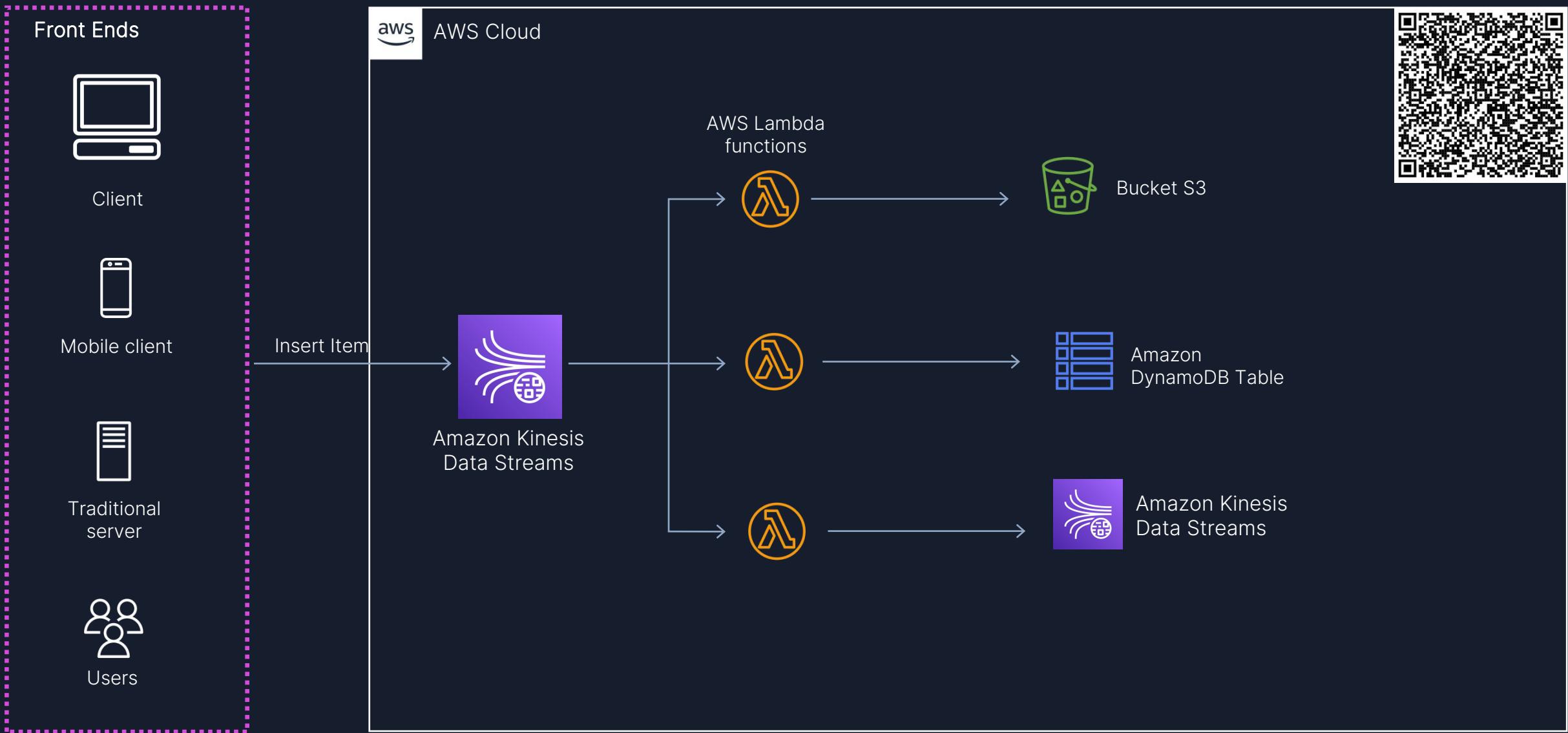
AWS Lambda Destination to SQS - DLQ Case



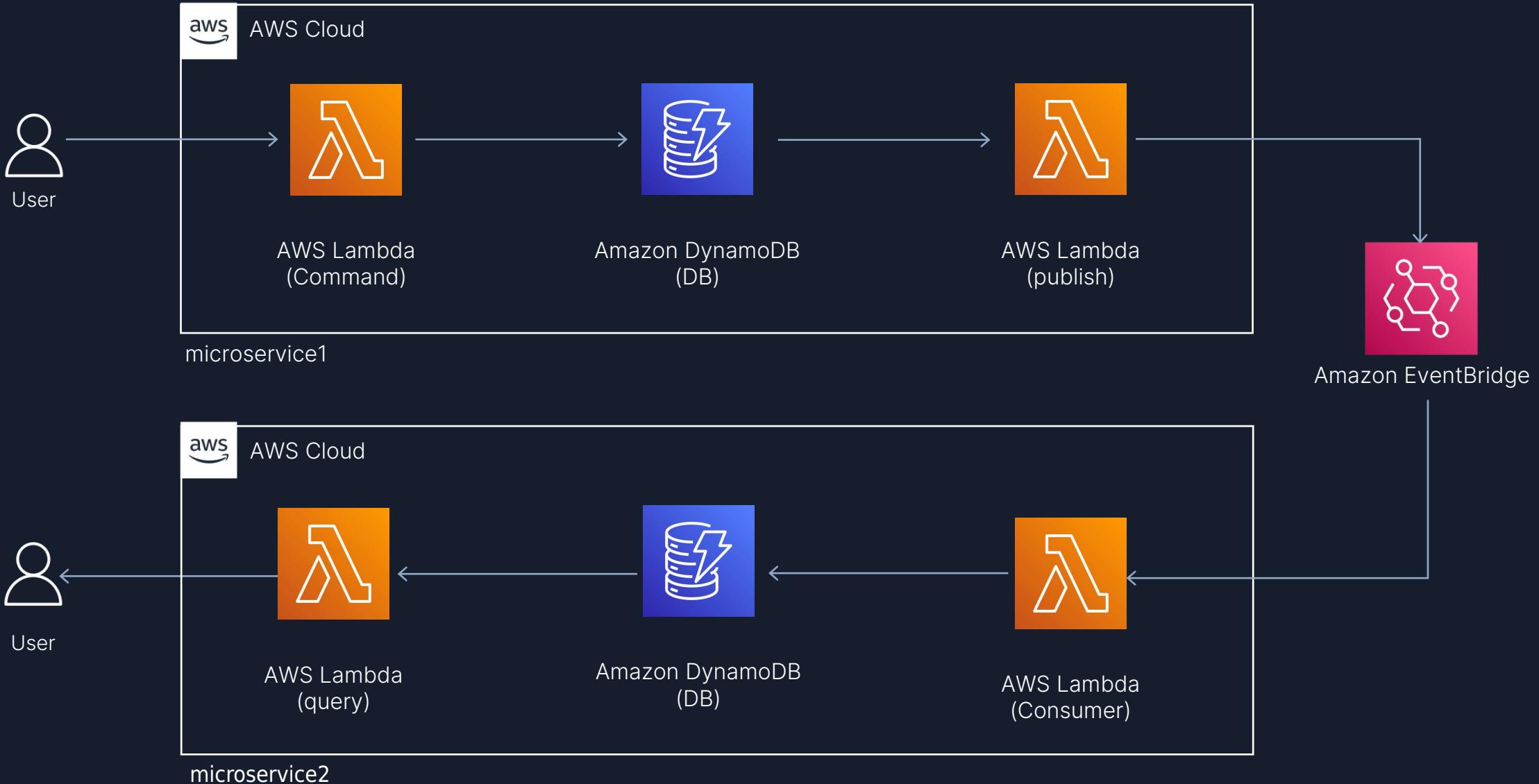
Using an Amazon SNS to send message to Lambda function



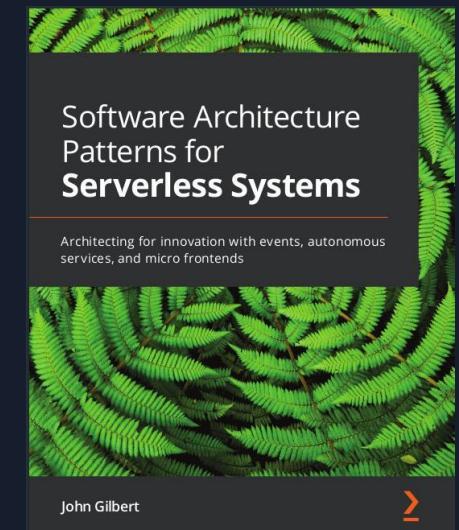
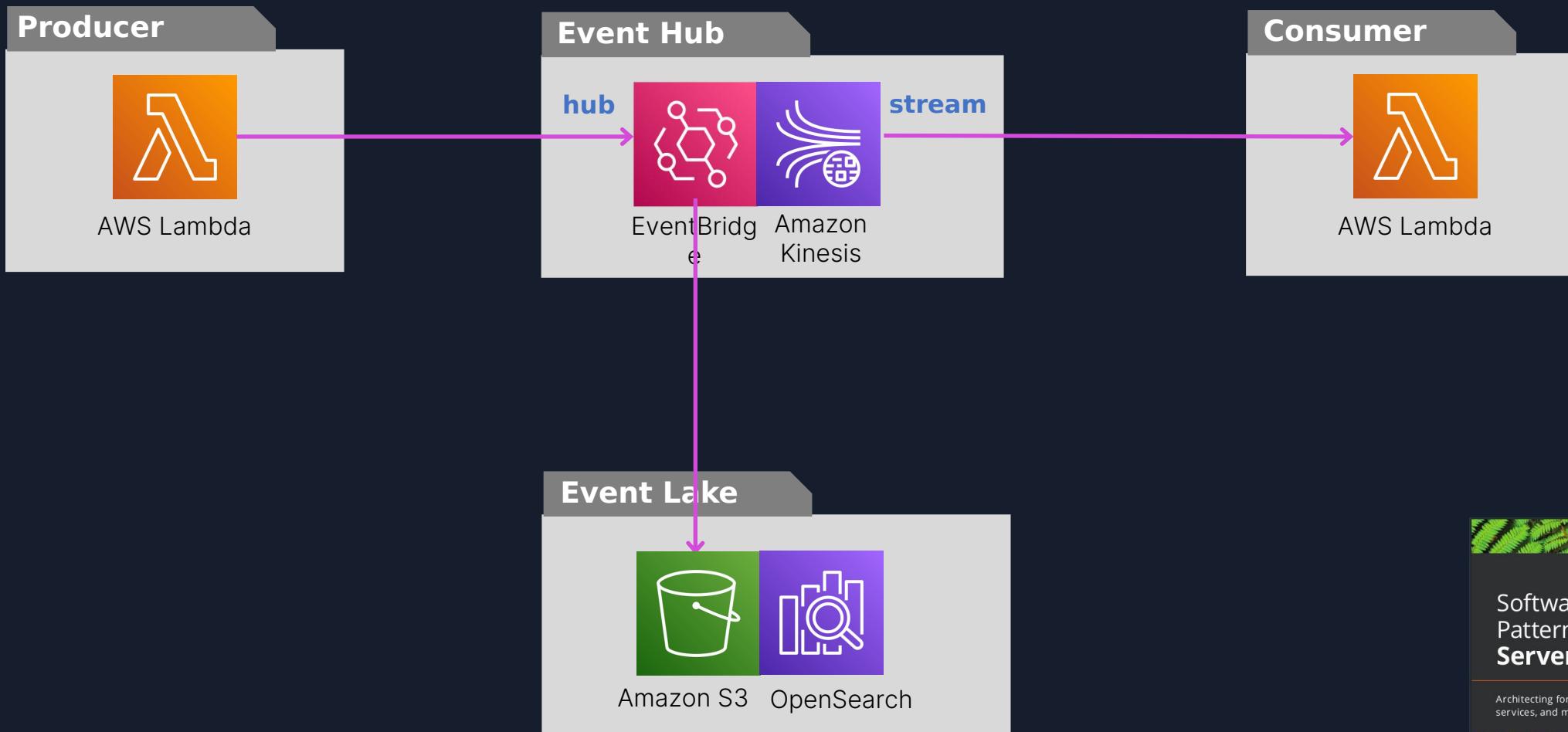
Processing performance with Amazon Kinesis Data Streams and AWS Lambda



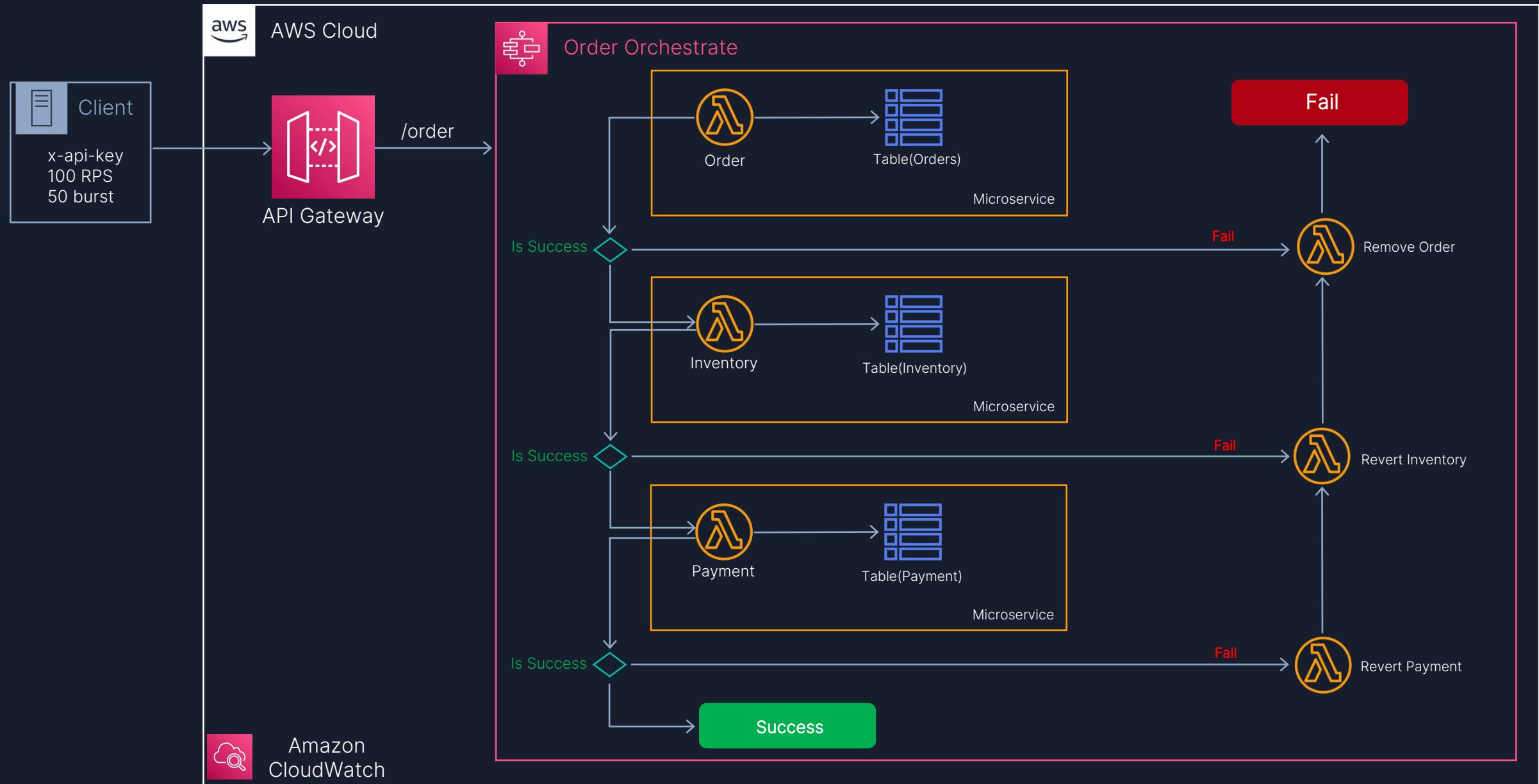
Command, Publish, Consume, Query (CPCQ)



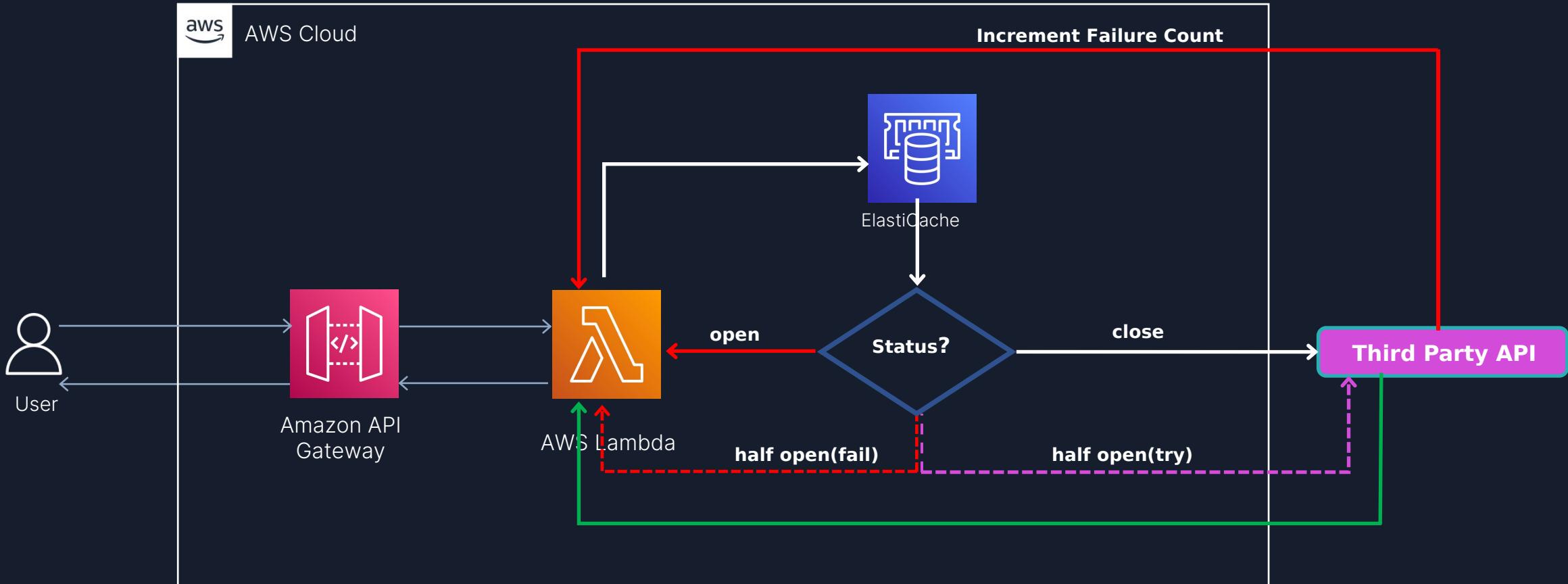
Event hub



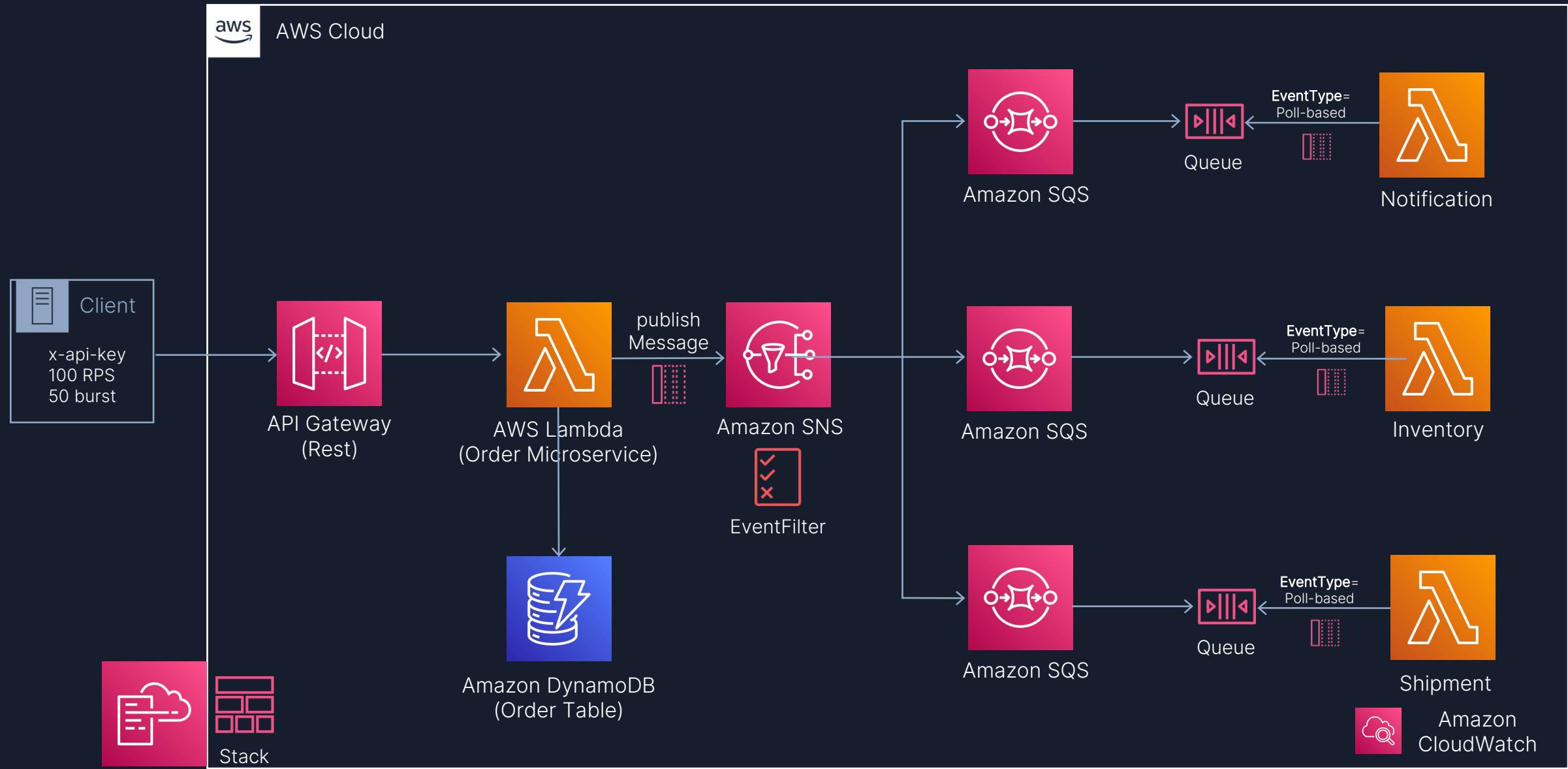
Saga Pattern For Orchestrate Distributed Transaction (AWS StepFunctions)



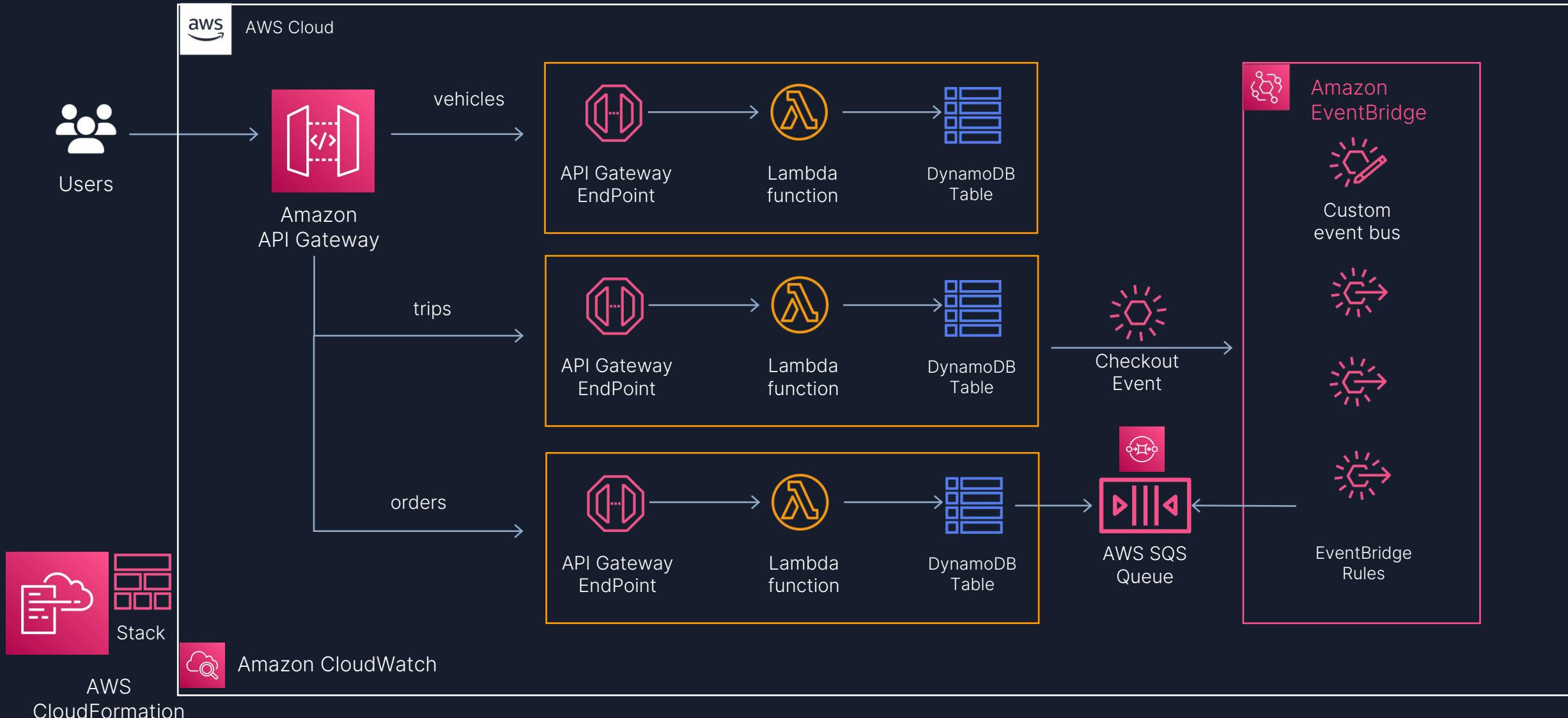
Circuit Breaker and Many More by Jeremy Daly



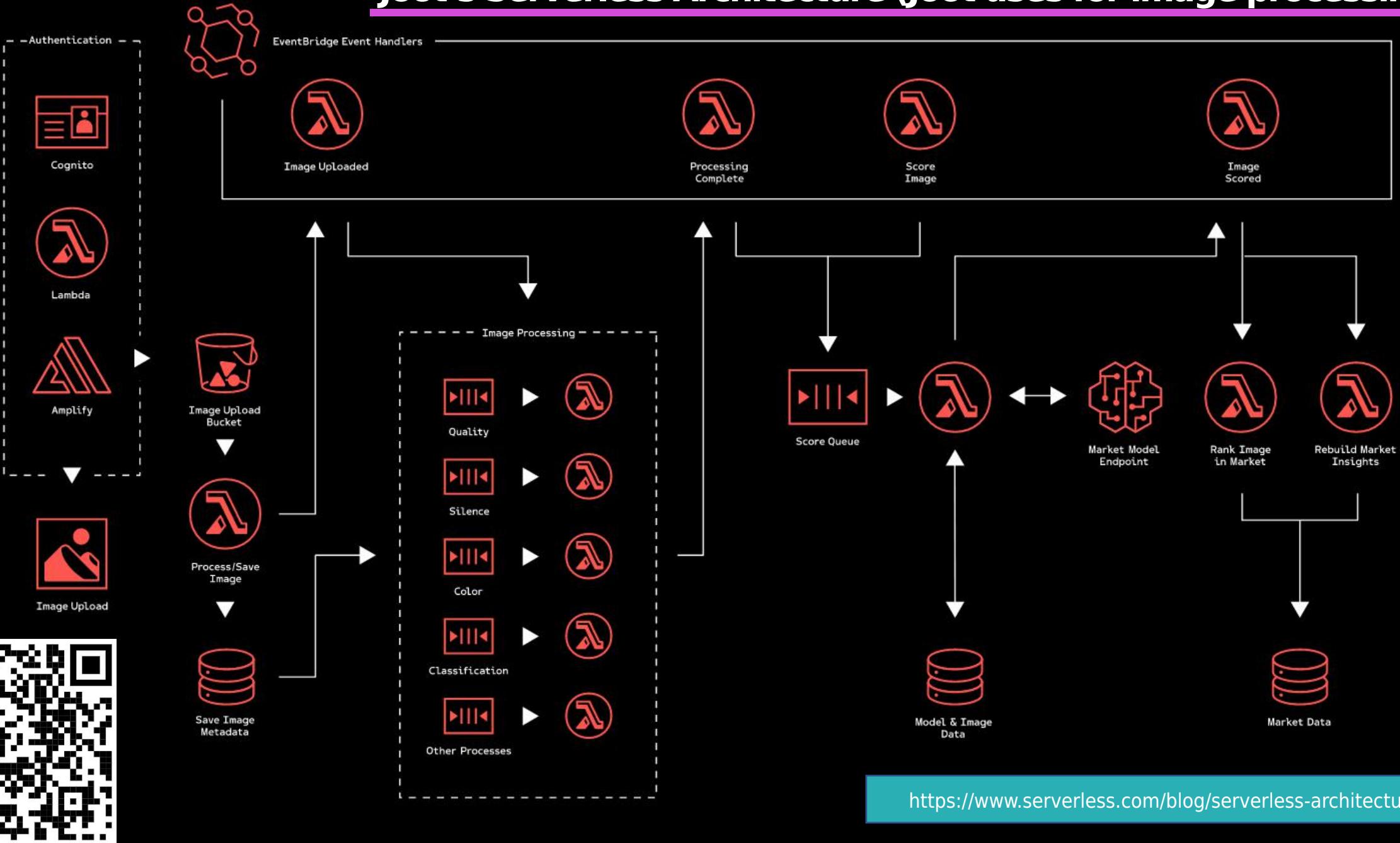
Fan-Out Serverless Architectures Using SNS, SQS and Lambda



Project Tracking Bus 2023 - Serverless



Joot's Serverless Architecture (Joot uses for image processing)



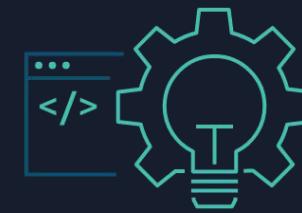


AWS CDK for Java and Python



What is AWS CDK?

The **AWS Cloud Development Kit (AWS CDK)** is an **open-source** software development framework for defining **cloud infrastructure as code(IAC)** with **modern programming languages** and deploying it through AWS CloudFormation.



<https://aws.amazon.com/cdk/faqs/>

https://aws.amazon.com/cdk/?nc1=h_ls

Demo CDK

Clone S3 AWS with CDK

Campus Party 2022

Home / Buckets

Buckets (4) Info Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Owner	Creation Date
data-lake-crime-results	US East (N. Virginia) us-east-1	Não informado	wmarians	27-04-2022 09:54:41
data-lake-crimes	US East (N. Virginia) us-east-1	Não informado	wmarians	27-04-2022 09:51:08
heroes-marvel	US East (N. Virginia) us-east-1	Não informado	wmarians	10-06-2022 15:36:53
heroes-marvel-iron-men	US East (N. Virginia) us-east-1	Não informado	wmarians	16-06-2022 15:17:48

<< < > >> Showing 1 to 4 of 4 buckets 10 ▾

Upload About: Seja Bem Vindo!!!

Message Plain Success Message Plain Info Message Plain Warn Message Plain Error

aws-image-upload-wsousa / src / main / resources / application.properties



weder96

download e upload de arquivos

Code

Blame

7 lines (6 loc) · 168 Bytes

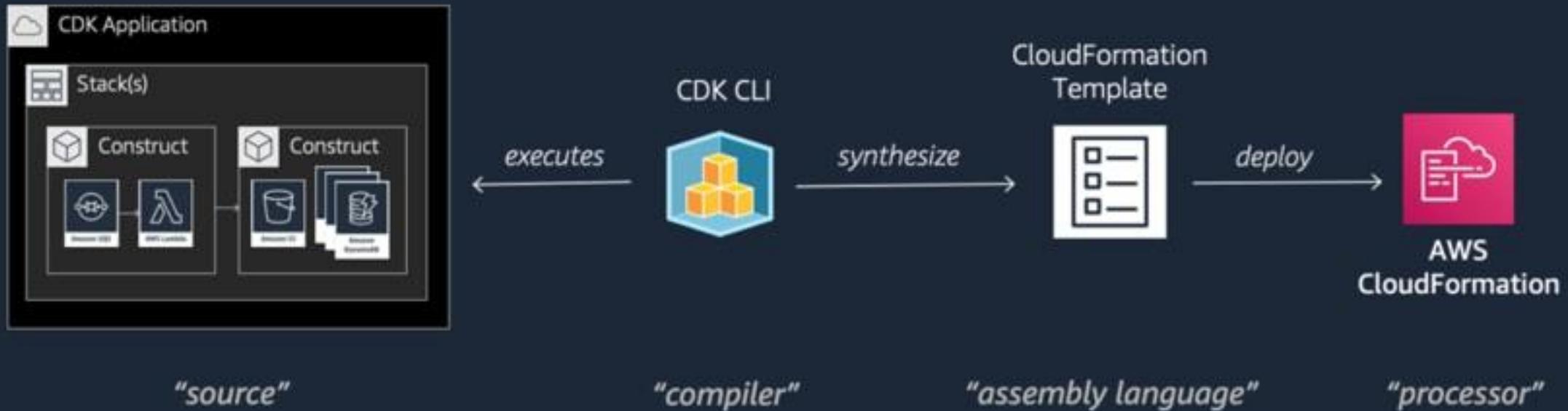
```
1 aws.access_key = ${value}
2 aws.secret_key = ${value}
3 aws.test = ${value}
4 spring.servlet.multipart.max-file-size=10MB
5
6 server.port=9090
7 springdoc.api-docs.path=/api-docs
```



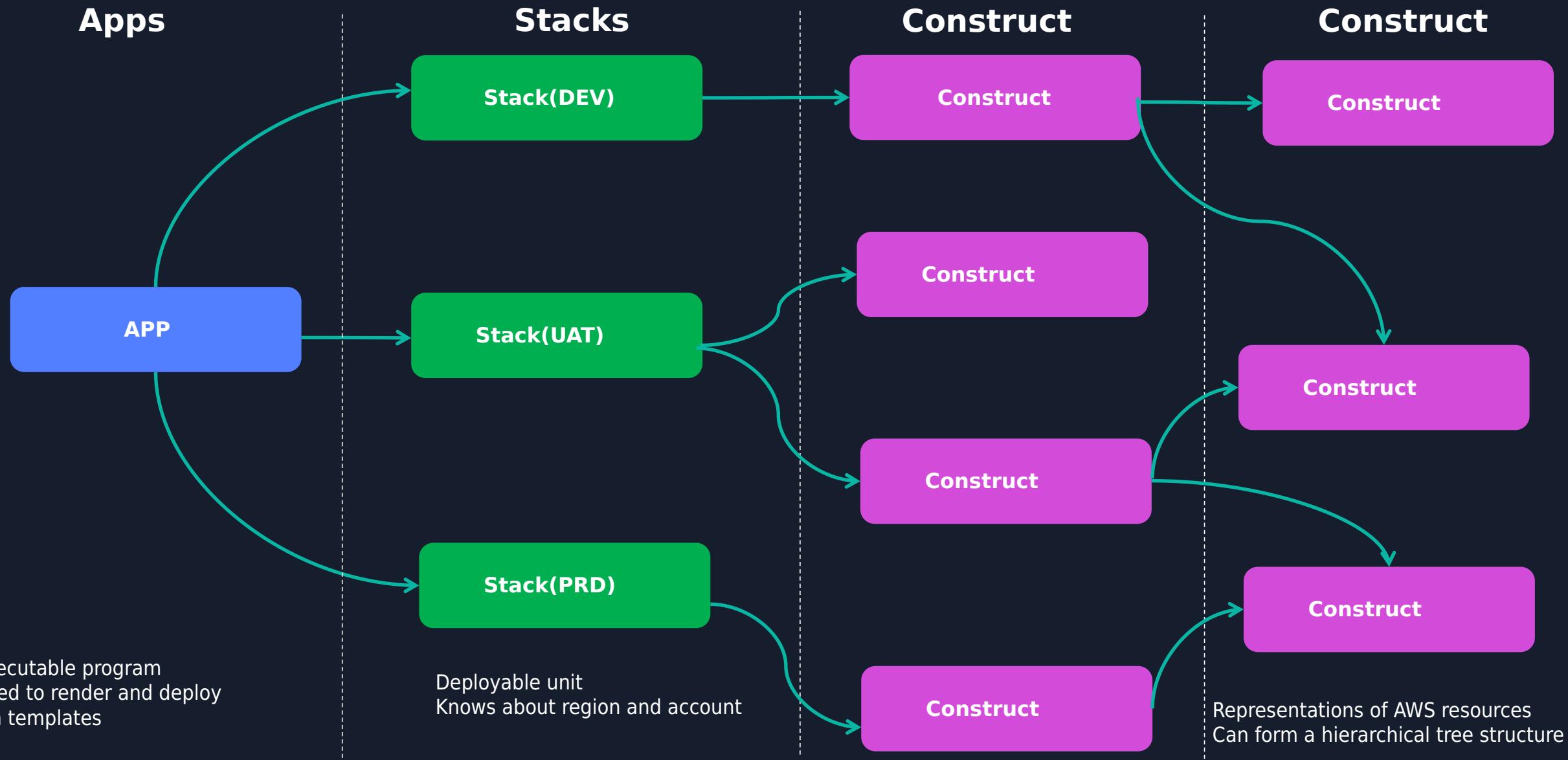
<https://github.com/weder96/aws-image-upload-wsousa>

<https://github.com/weder96/presentationCampusParty2022>

CDK Lifecycle

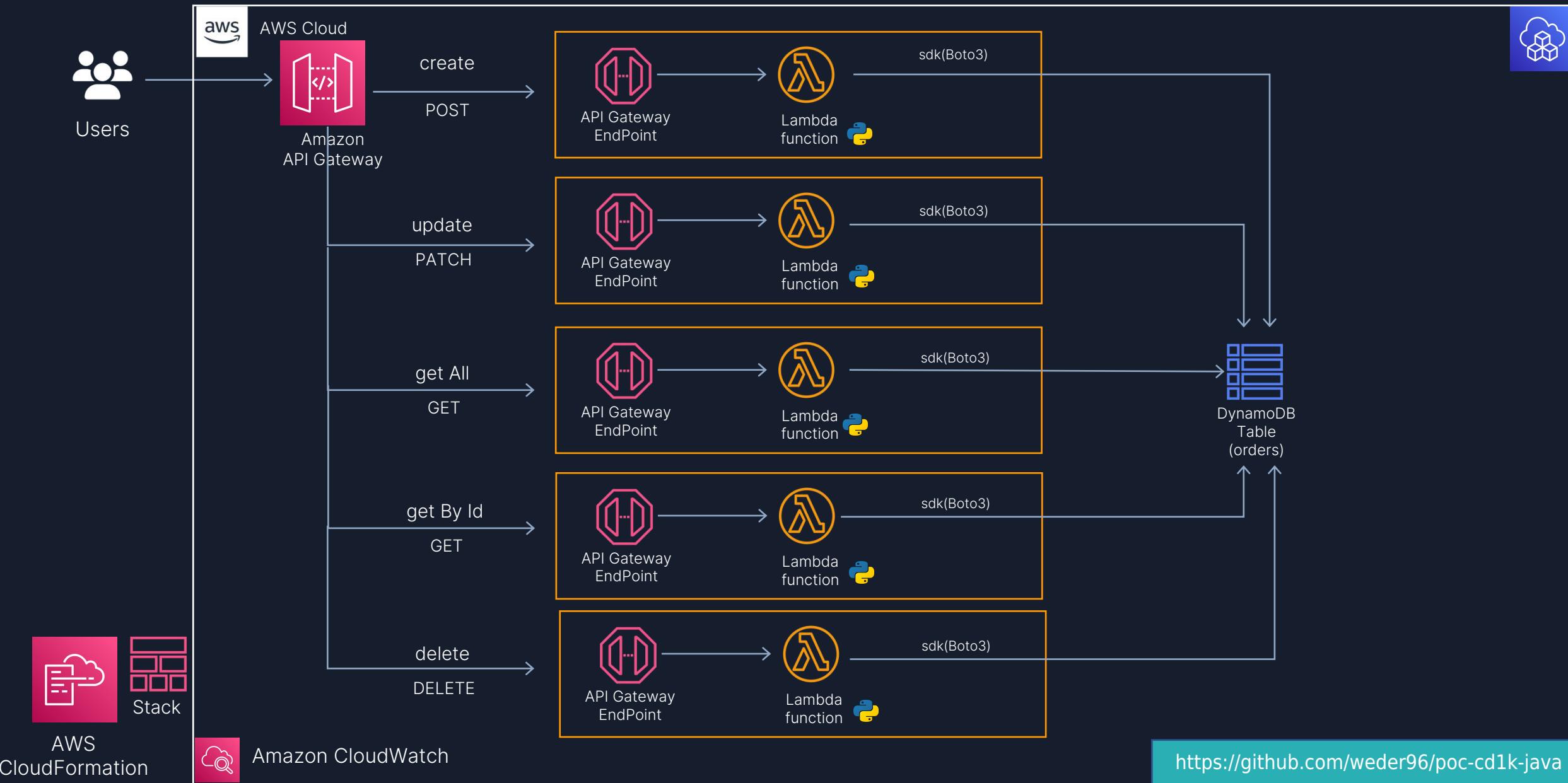


CDK Code Structure





Proof of Concepts AWS CDK - CRUD



Proof of Concepts AWS CDK



The purpose of this PoC is to understand how to use the CDK **to generate the infra(JAVA)** of a **CRUD(Create, Read, Update, Delete)** using , **API Gateway, lambdas(Python) and DynamoDB**, its advantages and limitations of traditional use with the creation of a Cloudformation(SAM) template.

Files

main my-cdk Lambda node_modules postman src test

my-cdk

Lambda

- app.py
- createItem.py
- deleteItem.py
- getAllItems.py
- getOneItem.py
- updateItem.py

src

main/java/com/myorg

- CreateDynamoDb.java
- CreateLambdaCron.java
- MyCdkApp.java
- MyStack.java

poc-cdk-java / my-cdk / src / main / java / com / myorg / CreateDynamoDb.java

Code

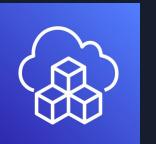
Blame

171 lines (142 loc) · 8.25 KB

Code 55% faster with GitHub Copilot

Raw

```
15 import java.nio.file.Files;
16 import java.nio.file.Paths;
17 import java.util.*;
18
19 public class CreateDynamoDb {
20     public CreateDynamoDb(MyStack myStack) {
21         cdkDynamodb(myStack);
22     }
23
24     public void cdkDynamodb(MyStack myStack){
25         TableProps tableProps = createTablePropsDynamoDB(myStack);
26         Table dynamodbTable = new Table(myStack, "orders", tableProps);
27         Map<String, String> lambdaEnvMap = createLambdaEnvMap(dynamodbTable);
28
29         Role role = createLambdaRole(myStack);
30
31         Function createItemFunction = new Function(myStack, "createItemFunction", getLambdaFunctionProps(role, "createItem","py"));
32         Function updateItemFunction = new Function(myStack, "updateItemFunction", getLambdaFunctionProps(role, "updateItem","py"));
33         Function deleteItemFunction = new Function(myStack, "deleteItemFunction", getLambdaFunctionProps(role, "deleteItem","py"));
34         Function getOneItemFunction = new Function(myStack, "getOneItemFunction", getLambdaFunctionProps(role, "getOneItem","py"));
35         Function getAllItemsFunction = new Function(myStack, "getAllItemsFunction", getLambdaFunctionProps(role, "getAllItems","py"));
36
37         dynamodbTable.grantReadWriteData(createItemFunction);
38         dynamodbTable.grantReadWriteData(updateItemFunction);
39         dynamodbTable.grantReadWriteData(deleteItemFunction);
40         dynamodbTable.grantReadWriteData(getOneItemFunction);
41         dynamodbTable.grantReadWriteData(getAllItemsFunction);
42
43         RestApi api = new RestApi(myStack, "itemsApi", RestApiProps.builder().restApiName("Items Service").build());
44     }
45 }
```



<https://github.com/weder96/poc-cdk-java>

Prerequisites for Installing AWS CDK



Before you can dive into the world of AWS CDK, you need to take care of a couple of prerequisites. First, ensure you have Node.js version 18.0.0 or later installed on your system.

```
> [weder@weder ~ ]$ node --version  
v18.17.1  
[weder@weder ~ ]$
```

Installing AWS CDK

Installing the AWS CDK is a breeze. Run the following command in your terminal or command prompt to install AWS CDK:

```
● ● ●  
1 npm install -g aws-cdk
```

Verifying AWS CDK Installation

```
> [weder@weder ~ ]$ cdk --version  
2.93.0 (build 724bd01)  
[weder@weder ~ ]$
```

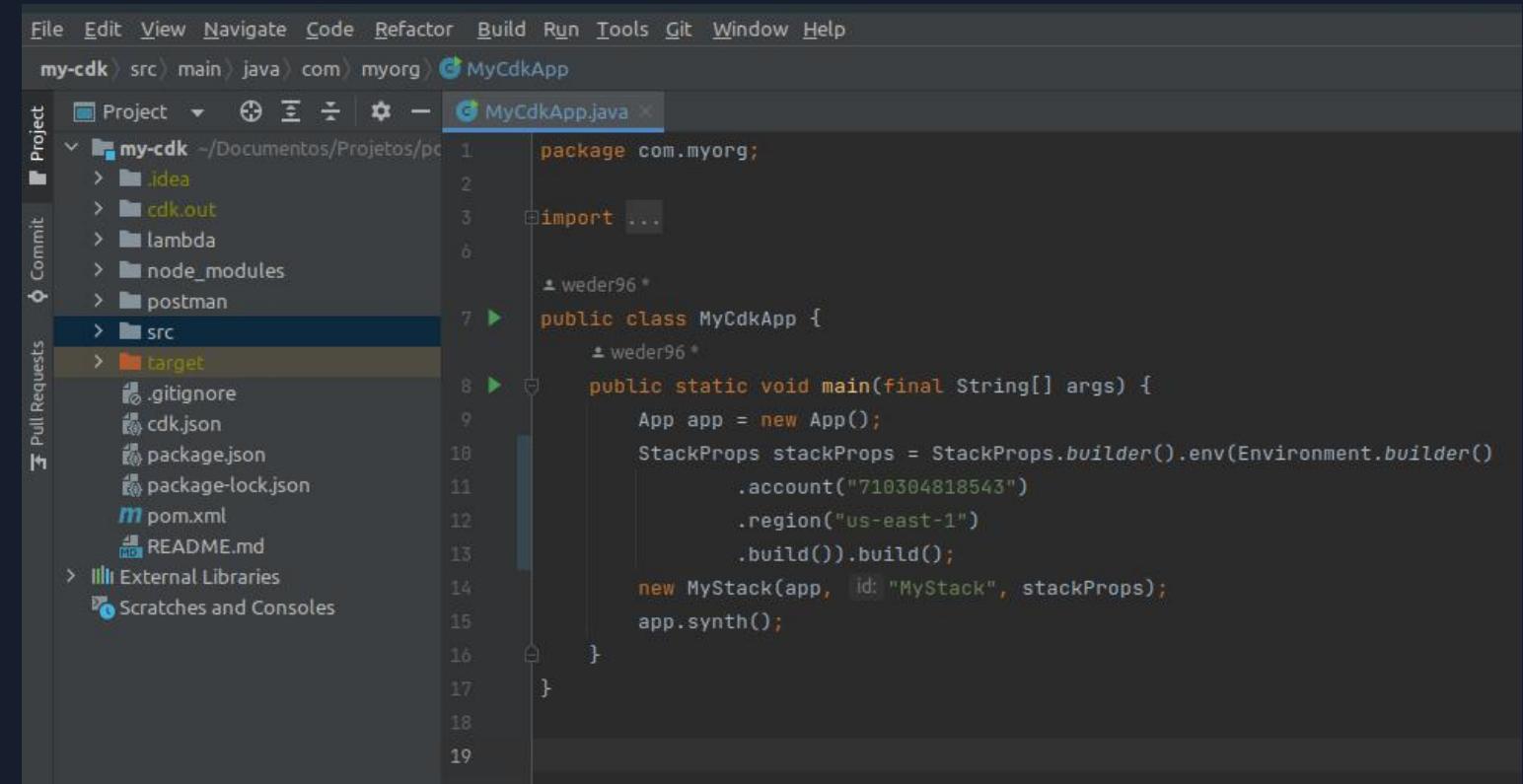
<https://hands-on.cloud/install-aws-cdk-guide/>

Initializing a CDK Project



To initialize a new CDK project, navigate to the desired directory in your terminal or command prompt and run the `cdk init` command with the appropriate template and programming language. For example, to create a Java-based CDK project, you would run the following command:

```
1 mkdir my-cdk
2 cd my-cdk
3 cdk init app --language java
```



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help
my-cdk src main java com myorg MyCdkApp
Project Commit Pull Requests External Libraries Scratches and Consoles
my-cdk ~/Documentos/Projetos/pc
> .idea
> cdk.out
> lambda
> node_modules
> postman
> src
> target
> .gitignore
> cdk.json
> package.json
> package-lock.json
> pom.xml
> README.md
> External Libraries
> Scratches and Consoles
MyCdkApp.java
package com.myorg;
import ...
public class MyCdkApp {
    public static void main(final String[] args) {
        App app = new App();
        StackProps stackProps = StackProps.builder().env(Environment.builder()
            .account("710304818543")
            .region("us-east-1")
            .build()).build();
        new MyStack(app, id: "MyStack", stackProps);
        app.synth();
    }
}
```

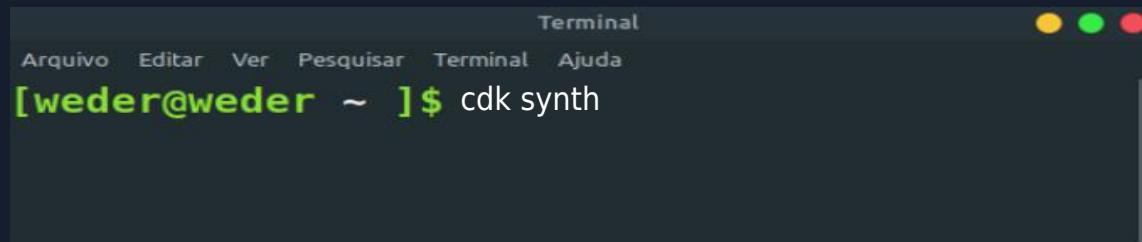
<https://hands-on.cloud/install-aws-cdk-guide/>

Building and Deploying Your AWS CDK App



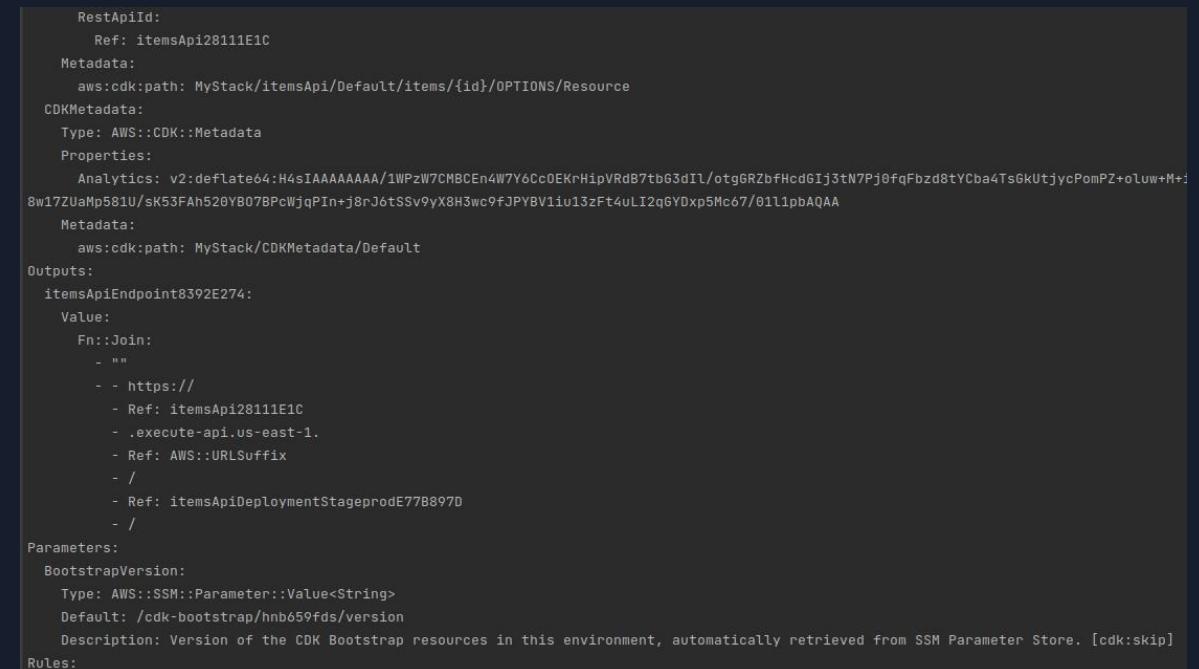
Once your CDK project is set up, the next step is to build and deploy your AWS CDK app. This process involves **synthesizing CloudFormation templates** in **YAML format**, saving them in the **cdk.out** folder as JSON, bootstrapping your stack, and deploying your app to your AWS account.

By following these steps, you can create and manage your **AWS infrastructure** using the power of AWS CDK and **your preferred programming language**.



A screenshot of a terminal window on a Mac OS X desktop. The window title is "Terminal". The menu bar shows "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The command prompt shows "[weder@weder ~]\$". Below the prompt, the command "cdk synth" is typed. The terminal window has a dark background with light-colored text.

To synthesize **CloudFormation templates for your AWS CDK app**, run the `cdk synth` command in your terminal or command prompt. This command generates **YAML-formatted CloudFormation templates** for each stack in your app, encompassing all the **resources you've defined in your CDK code**.



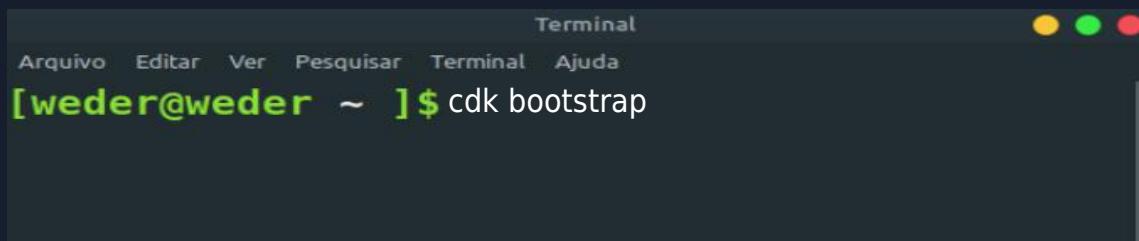
A screenshot of a YAML-formatted CloudFormation template. The template defines a REST API endpoint named "itemsApi" with a specific ID. It includes properties like "Metadata" and "Properties" (which contains an Analytics key), and outputs like "itemsApiEndpoint". The "Parameters" section includes a "BootstrapVersion" parameter with a default value of "cdk-bootstrap/hnb659fds/version". The "Description" for this parameter states: "Version of the CDK Bootstrap resources in this environment, automatically retrieved from SSM Parameter Store. [cdk:skip]". The template is very long, showing many nested structures and complex values.

Bootstrapping AWS CDK



Before deploying your AWS CDK app, **you'll need to bootstrap your stack**. Bootstrapping involves creating dedicated Amazon S3 buckets and other containers required for deploying your app using AWS CloudFormation.

To bootstrap your stack, run the `cdk bootstrap` command in your terminal or command prompt. This command deploys a CDKToolkit CloudFormation stack into the specified environment, provisioning the necessary resources for your AWS CDK app.



```
✖ Bootstrapping environment aws://710304818543/us-east-1...
Trusted accounts for deployment: (none)
Trusted accounts for lookup: (none)
Using default execution policy of 'arn:aws:iam::aws:policy/AdministratorAccess'. Pass '--cloudformation-execution-policies' to customize.

✖ hotswap deployment skipped - no changes were detected (use --force to override)

Environment aws://710304818543/us-east-1 bootstrapped (no changes).
```

Deploying the App

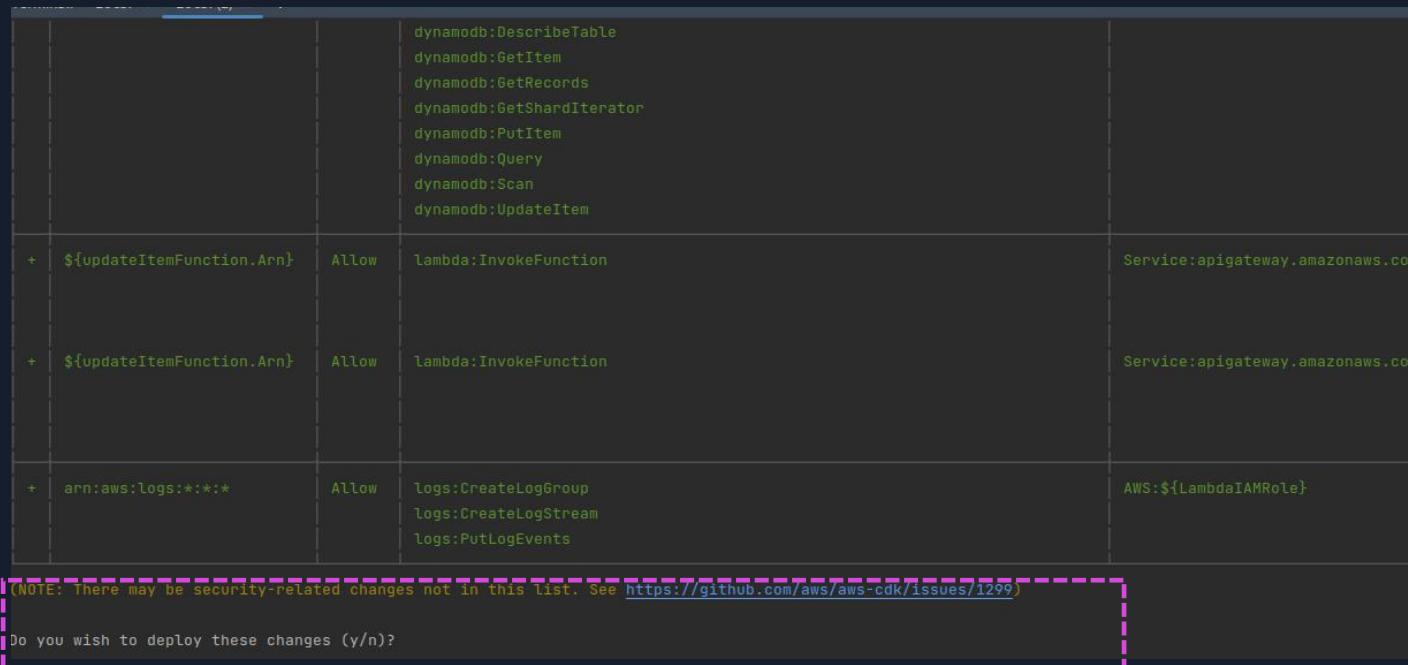


You're ready to deploy your AWS CDK app with your synthesized CloudFormation templates and bootstrapped stack. To do this, run the **cdk deploy** command in your terminal or command prompt.



```
Terminal
Arquivo Editar Ver Pesquisar Terminal Ajuda
[weder@weder ~ ]$ cdk deploy
```

This command deploys your CDK app to your AWS account, **creating and updating the necessary AWS resources** as defined in your CloudFormation templates. Once the deployment is complete, you can monitor and manage your deployed resources using the **AWS Management Console or AWS CLI**.



```
dynamodb:DescribeTable
dynamodb:GetItem
dynamodb:GetRecords
dynamodb:GetShardIterator
dynamodb:PutItem
dynamodb:Query
dynamodb:Scan
dynamodb:UpdateItem

+ ${updateItemFunction.Arn} Allow lambda:InvokeFunction Service:apigateway.amazonaws.com

+ ${updateItemFunction.Arn} Allow lambda:InvokeFunction Service:apigateway.amazonaws.com

+ arn:aws:logs:*** Allow logs>CreateLogGroup
logs>CreateLogStream
logs>PutLogEvents AWS:${LambdaIAMRole}

[NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299]

Do you wish to deploy these changes (y/n)?
```

Deploying the App



aws resources being created by the **deploy** command, using the template created in the **synth** command

```
MyStack: deploying... [1/1]
MyStack: creating CloudFormation changeset...
[██████████ .....] (3/33)

18:21:16 | CREATE_IN_PROGRESS | AWS::CloudFormation::Stack | MyStack
18:21:21 | CREATE_IN_PROGRESS | AWS::S3::Bucket           | MyBucket
18:21:21 | CREATE_IN_PROGRESS | AWS::IAM::Role            | LambdaIAMRole
18:21:21 | CREATE_IN_PROGRESS | AWS::DynamoDB::Table      | orders
18:21:24 | CREATE_IN_PROGRESS | AWS::ApiGateway::Resource | itemsApi/Default/items/{id}
18:21:25 | CREATE_IN_PROGRESS | AWS::ApiGateway::Method   | itemsApi/Default/items/OPTIONS
```

After creating it, it shows the time it was created and whether it managed to **create all the necessary resources**

```
Do you wish to deploy these changes (y/n)? y
MyStack: deploying... [1/1]
MyStack: creating CloudFormation changeset...

MyStack
  Deployment time: 88.96s

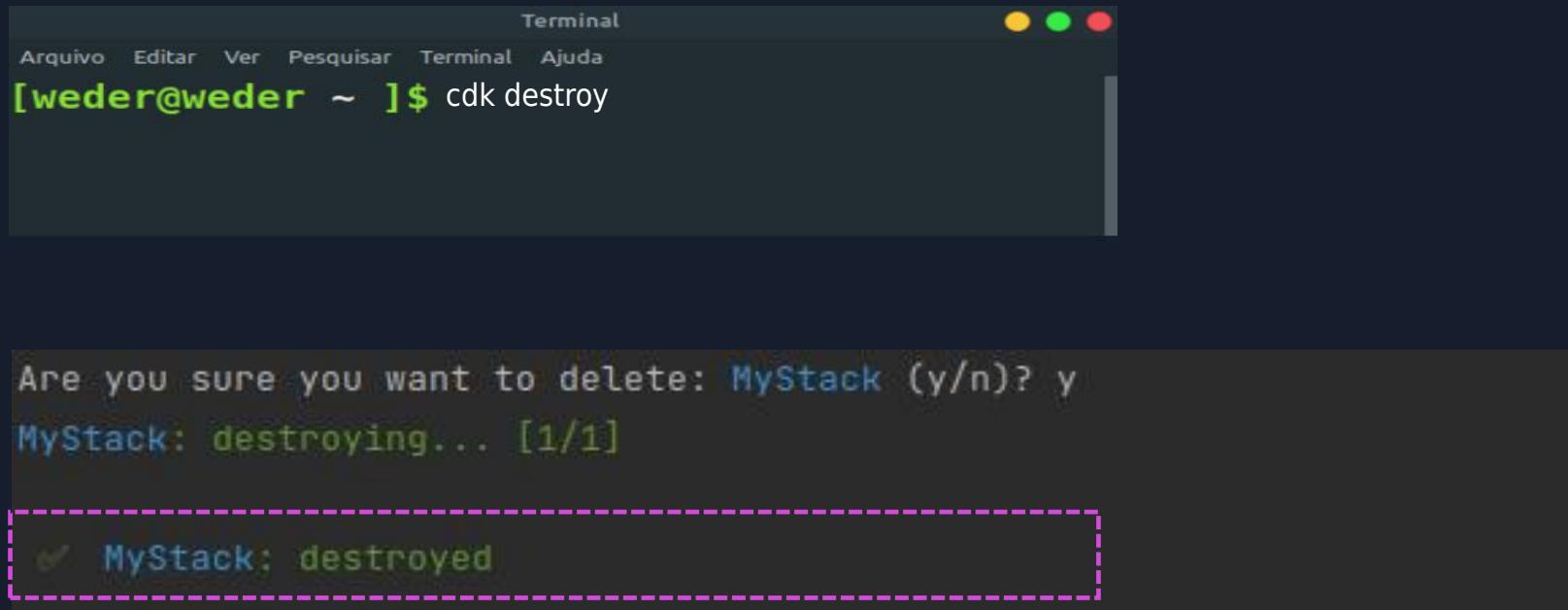
Outputs:
MyStack.itemsApiEndpoint8392E274 = https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/
Stack ARN:
arn:aws:cloudformation:us-east-1:710304818543:stack/MyStack/9ce51c40-4e8d-11ee-b5aa-0a53f035b84d

Total time: 93.57s
```

Destroy the App



When you no longer need certain resources in your AWS CDK app, **you can destroy them using the cdk destroy command**. This command deletes the specified stack and all its associated resources unless they have been marked with a Deletion Policy to be retained.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a menu bar with options: Arquivo, Editar, Ver, Pesquisar, Terminal, and Ajuda. Below the menu, the terminal prompt is [weder@weder ~]\$, followed by the command \$ cdk destroy. The terminal then displays a confirmation message: Are you sure you want to delete: MyStack (y/n)? followed by the response y. It then shows the progress of the destruction: MyStack: destroying... [1/1]. Finally, it shows a success message with a green checkmark: ✓ MyStack: destroyed. A dashed red box highlights the success message.

```
Are you sure you want to delete: MyStack (y/n)? y
MyStack: destroying... [1/1]
✓ MyStack: destroyed
```

Diff the App



When you have **made changes to your infrastructure code** by **adding** or **removing** certain **resources** in your AWS CDK application, you can use the **cdk diff** command. This **command** will show what has been **modified** and which resources will be created and will be **listed** for **monitoring**.

The screenshot shows a terminal window with two tabs: "Terminal" and "Local (2)". The "Local (2)" tab is active and displays the output of the "cdk diff" command. The output lists numerous AWS resources, primarily Lambda functions and API Gateway stages, with their names and deployment stages. A pink dashed box highlights the "Outputs" section at the bottom of the terminal window, which contains a single entry for an "Output" resource named "itemsApi/Endpoint". The status bar at the bottom of the terminal window indicates "Number of stacks with differences: 1".

```
[+] AWS::ApiGateway::Stage itemsApi/DeploymentStage.prod itemsApiDeploymentStageprodE77B897D
[+] AWS::ApiGateway::Resource itemsApi/Default/items itemsApiitems9015DBED
[+] AWS::Lambda::Permission itemsApi/Default/items/GET/ApiPermission.MyStackitemsApiA38DACFB.GET..items itemsApiitemsGETApiPe
[+] AWS::Lambda::Permission itemsApi/Default/items/GET/ApiPermission.Test.MyStackitemsApiA38DACFB.GET..items itemsApiitemsGET
[+] AWS::ApiGateway::Method itemsApi/Default/items/GET itemsApiitemsGET59B0F78A
[+] AWS::Lambda::Permission itemsApi/Default/items/POST/ApiPermission.MyStackitemsApiA38DACFB.POST..items itemsApiitemsPOSTAp
[+] AWS::Lambda::Permission itemsApi/Default/items/POST/ApiPermission.Test.MyStackitemsApiA38DACFB.POST..items itemsApiitemsP
[+] AWS::ApiGateway::Method itemsApi/Default/items/POST itemsApiitemsPOSTDD3E83D0
[+] AWS::ApiGateway::Method itemsApi/Default/items/OPTIONS itemsApiitemsOPTIONSB46B4D53
[+] AWS::ApiGateway::Resource itemsApi/Default/items/{id} itemsApiitemsidA29927C2
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/GET/ApiPermission.MyStackitemsApiA38DACFB.GET..items.{id} itemsApiite
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/GET/ApiPermission.Test.MyStackitemsApiA38DACFB.GET..items.{id} itemsA
[+] AWS::ApiGateway::Method itemsApi/Default/items/{id}/GET itemsApiitemsidGET38A333A8
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/PATCH/ApiPermission.MyStackitemsApiA38DACFB.PATCH..items.{id} itemsAp
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/PATCH/ApiPermission.Test.MyStackitemsApiA38DACFB.PATCH..items.{id} it
[+] AWS::ApiGateway::Method itemsApi/Default/items/{id}/PATCH itemsApiitemsidPATCH0548CB6A
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/DELETE/ApiPermission.MyStackitemsApiA38DACFB.DELETE..items.{id} items
[+] AWS::Lambda::Permission itemsApi/Default/items/{id}/DELETE/ApiPermission.Test.MyStackitemsApiA38DACFB.DELETE..items.{id}
[+] AWS::ApiGateway::Method itemsApi/Default/items/{id}/DELETE itemsApiitemsidDELETE21550005
[+] AWS::ApiGateway::Method itemsApi/Default/items/{id}/OPTIONS itemsApiitemsidOPTIONS62809100

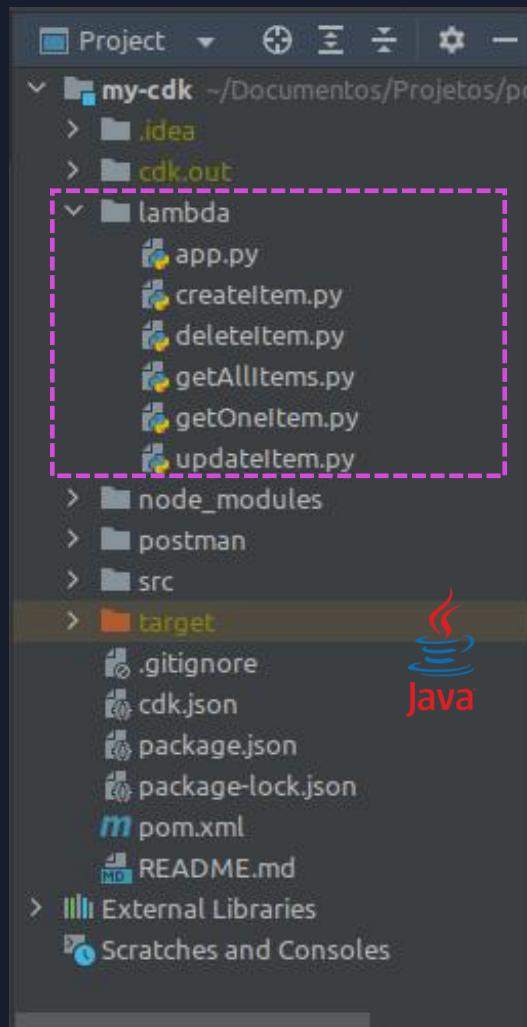
Outputs
[+] Output itemsApi/Endpoint itemsApiEndpoint8392E274: {"Value": {"Fn::Join": "", ["https://", {"Ref": "itemsApi28111E1C"}]}, ".exec
```

* Number of stacks with differences: 1

Code Lambda with Python



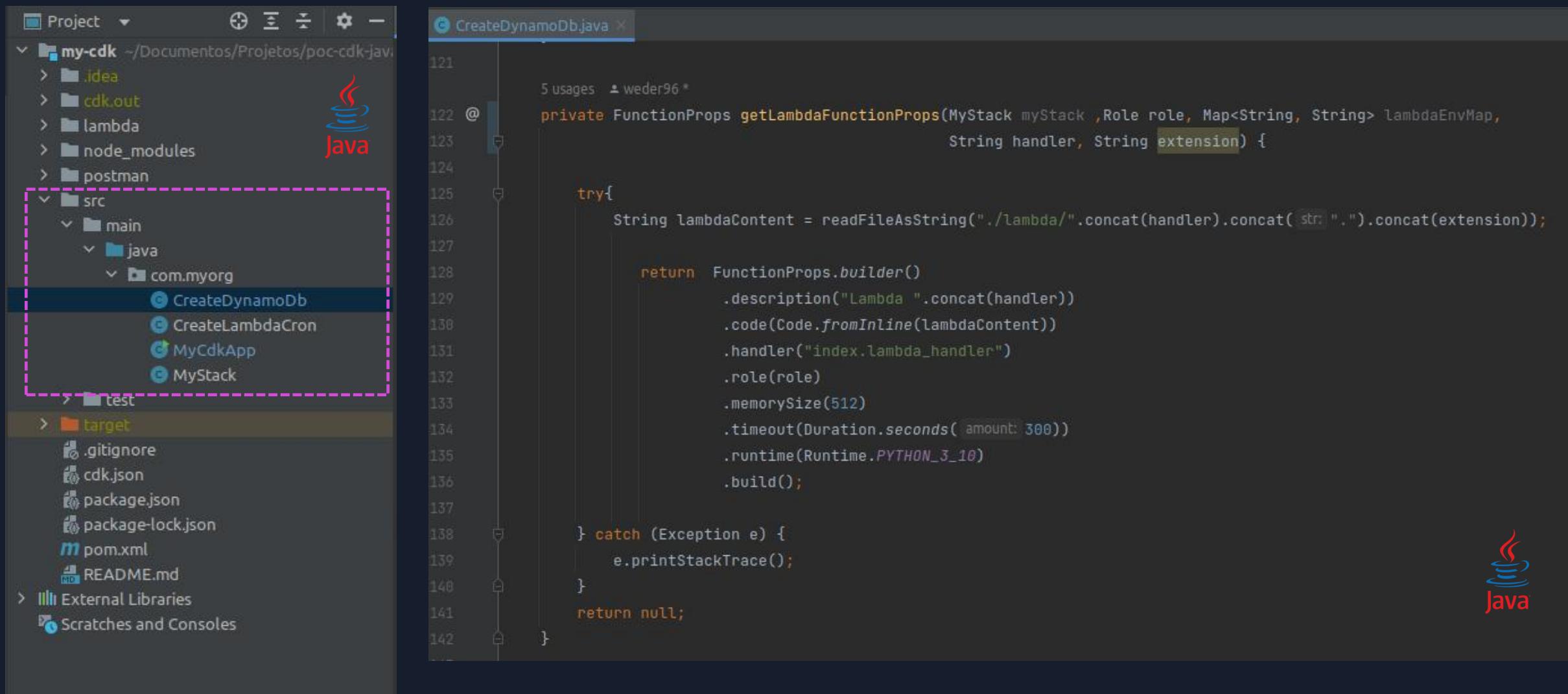
The **AWS SDK for Python (Boto3)** provides a **Python API for AWS infrastructure services**. Using the **SDK for Python**, you can build applications on top of **Amazon S3, Amazon EC2, Amazon DynamoDB**, and more.



```
1 import json
2 import boto3
3 import uuid
4
5 client = boto3.resource('dynamodb')
6
7 def lambda_handler(event, context):
8     print('Starting input lambda function call')
9     table = client.Table('orders')
10    myuuid = uuid.uuid4()
11
12    print('Your UUID is: ' + str(myuuid))
13
14    order=json.loads(event['body'])
15
16    data = table.put_item(
17        Item={
18            'orderId': str(myuuid),
19            'orderDate': order['orderDate'],
20            'status': order['status'],
21            'desc': order['desc'],
22            'updateOrderDate': order['updateOrderDate'],
23            'Name': order['Name'],
24            'Email': order['Email']
25        }
26    )
27
28    response = {
29        'statusCode': 200,
30        'body': json.dumps('successfully created item!'),
31        'headers': { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*'}
32    }
33
34
35    return response
```

Code IAC(infrastructure as Code)

In this case we **are generating the infrastructure** using the AWS SDK, however the code controller is **written in Java**, using the **CDK** it generates the Cloudformation template, which is then deployed creating the services automatically in **AWS in your account** configured in your machine via **credentials**.



The screenshot shows an IDE interface with a Java project structure on the left and a code editor on the right.

Project Structure:

- Project: my-cdk (~/Documentos/Projetos/poc-cdk-java)
- Files: .idea, cdk.out, lambda, node_modules, postman, src/main/java/com.myorg (highlighted with a dashed purple border), test, target, .gitignore, cdk.json, package.json, package-lock.json, pom.xml, README.md, External Libraries, Scratches and Consoles.

Code Editor (CreateDynamoDb.java):

```
121
122 @Private
123 private FunctionProps getLambdaFunctionProps(MyStack myStack, Role role, Map<String, String> lambdaEnvMap,
124                                                 String handler, String extension) {
125
126     try{
127         String lambdaContent = readFileAsString("./lambda/.concat(handler).concat(str).concat(extension));
128
129         return FunctionProps.builder()
130             .description("Lambda ".concat(handler))
131             .code(Code.fromInline(lambdaContent))
132             .handler("index.lambda_handler")
133             .role(role)
134             .memorySize(512)
135             .timeout(Duration.seconds(amount: 300))
136             .runtime(Runtime.PYTHON_3_10)
137             .build();
138     } catch (Exception e) {
139         e.printStackTrace();
140     }
141
142 }
```

A Java icon is visible next to the project structure and code editor.

Code IAC (infrastructure as Code)

Create Lambda Role

```
1 usage  ▲ weder96
public Role createLambdaRole(MyStack myStack){
    PolicyStatement statement2 = PolicyStatement.Builder.create()
        .effect(Effect.ALLOW)
        .actions(Arrays.asList(new String[] {"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"}))
        .resources(Arrays.asList(new String[] {"arn:aws:logs:*:*:*"})).build();

    PolicyDocument policyDocument = PolicyDocument.Builder.create()
        .statements(Arrays.asList(new PolicyStatement[]{statement2})).build();

    return Role.Builder.create(myStack, id: "LambdaIAMRole")
        .roleName("LambdaIAMRole")
        .inlinePolicies(Collections.singletonMap("key", policyDocument))
        .path("/")
        .assumedBy(new ServicePrincipal("lambda.amazonaws.com")).build();
}
```



Code IAC (infrastructure as Code)

Create DynamoDB and Grant Permissions Lambda Functions

```
private TableProps createTablePropsDynamoDB(MyStack myStack) {  
    TableProps tableProps;  
    Attribute partitionKey = Attribute.builder().name("orderId").type(AttributeType.STRING).build();  
    Attribute sortKey = Attribute.builder().name("orderDate").type(AttributeType.STRING).build();  
    tableProps = TableProps.builder().tableName("orders")  
        .partitionKey(partitionKey)  
        .sortKey(sortKey).removalPolicy(RemovalPolicy.DESTROY).build();  
  
    return tableProps;  
}
```

```
private static Map<String, String> createLambdaEnvMap(Table dynamodbTable) {  
    Map<String, String> lambdaEnvMap = new HashMap<>();  
    lambdaEnvMap.put("TABLE_NAME", dynamodbTable.getTableName());  
    lambdaEnvMap.put("PRIMARY_KEY", "orderId");  
    return lambdaEnvMap;  
}
```

```
public void cdkDynamodb(MyStack myStack){  
    TableProps tableProps = createTablePropsDynamoDB(myStack);  
    Table dynamodbTable = new Table(myStack, id: "orders", tableProps);  
    Map<String, String> lambdaEnvMap = createLambdaEnvMap(dynamodbTable);  
  
    Role role = createLambdaRole(myStack);
```

```
    Function createItemFunction = new Function(myStack, id: "createItemFunction", getLambdaFunctionProps(myStack, role, lambdaEnvMap, handler: "createItem", extension: "py"));  
    Function updateItemFunction = new Function(myStack, id: "updateItemFunction", getLambdaFunctionProps(myStack, role, lambdaEnvMap, handler: "updateItem", extension: "py"));  
    Function deleteItemFunction = new Function(myStack, id: "deleteItemFunction", getLambdaFunctionProps(myStack, role, lambdaEnvMap, handler: "deleteItem", extension: "py"));  
    Function getOneItemFunction = new Function(myStack, id: "getOneItemFunction", getLambdaFunctionProps(myStack, role, lambdaEnvMap, handler: "getOneItem", extension: "py"));  
    Function getAllItemsFunction = new Function(myStack, id: "getAllItemsFunction", getLambdaFunctionProps(myStack, role, lambdaEnvMap, handler: "getAllItems", extension: "py"));
```

```
dynamodbTable.grantReadWriteData(createItemFunction);  
dynamodbTable.grantReadWriteData(updateItemFunction);  
dynamodbTable.grantReadWriteData(deleteItemFunction);  
dynamodbTable.grantReadWriteData(getOneItemFunction);  
dynamodbTable.grantReadWriteData(getAllItemsFunction);
```

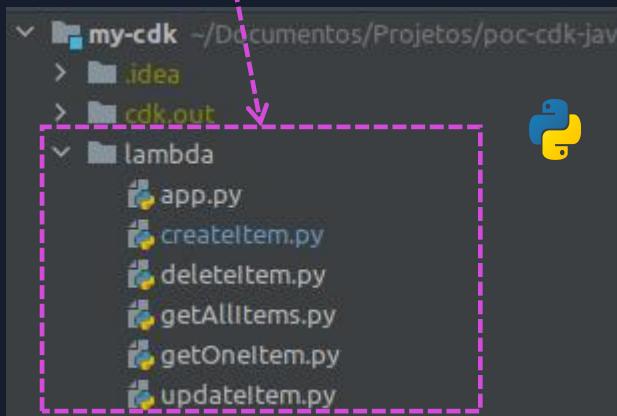


Code IAC (infrastructure as Code)



Reading Lambda Functions

```
public String readFileSync(String fileName) throws Exception {  
    String data = "";  
    try {  
        data = new String(Files.readAllBytes(Paths.get(fileName)), charsetName: "UTF-8");  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return data;  
}
```



```
Function createItemFunction = new Function(myStack, id: "createItemFunction",  
Function updateItemFunction = new Function(myStack, id: "updateItemFunction",  
Function deleteItemFunction = new Function(myStack, id: "deleteItemFunction",  
Function getOneItemFunction = new Function(myStack, id: "getOneItemFunction",  
Function getAllItemsFunction = new Function(myStack, id: "getAllItemsFunction",  
                                         getLambdaFunctionProps(role, nameFunction: "createItem", extension: "py"));  
                                         getLambdaFunctionProps(role, nameFunction: "updateItem", extension: "py"));  
                                         getLambdaFunctionProps(role, nameFunction: "deleteItem", extension: "py"));  
                                         getLambdaFunctionProps(role, nameFunction: "getOneItem", extension: "py"));  
                                         getLambdaFunctionProps(role, nameFunction: "getAllItems", extension: "py"));
```

```
private FunctionProps getLambdaFunctionProps(Role role, String nameFunction, String extension) {  
    try{  
        String lambdaContent = readFileSync("./lambda/.concat(nameFunction).concat(str: ".).concat(extension));  
        return FunctionProps.builder()  
            .description("Lambda ".concat(nameFunction))  
            .code(Code.fromInline(lambdaContent))  
            .handler("index.lambda_handler")  
            .role(role)  
            .memorySize(512)  
            .timeout(Duration.seconds(amount: 300))  
            .runtime(Runtime.PYTHON_3_10)  
            .build();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return null;  
}
```

Code IAC (infrastructure as Code) - Create API Gateway



```
RestApi api = new RestApi(myStack, id: "itemsApi", RestApiProps.builder().restApiName("Items Service").build());

IResource items = api.getRoot().addResource(s: "items");

Integration getAllIntegration = new LambdaIntegration(getAllItemsFunction);
items.addMethod(s: "GET", getAllIntegration);

Integration createOneIntegration = new LambdaIntegration(createItemFunction);
items.addMethod(s: "POST", createOneIntegration);
addCorsOptions(items);

IResource singleItem = items.addResource(s: "{id}");
Integration getOneIntegration = new LambdaIntegration(getOneItemFunction);
singleItem.addMethod(s: "GET", getOneIntegration);

Integration updateOneIntegration = new LambdaIntegration(updateItemFunction);
singleItem.addMethod(s: "PATCH", updateOneIntegration);

Integration deleteOneIntegration = new LambdaIntegration(deleteItemFunction);
singleItem.addMethod(s: "DELETE", deleteOneIntegration);
addCorsOptions(singleItem);
```

```
private void addCorsOptions(IResource item) {
    List<MethodResponse> methodResponses = new ArrayList<>();

    Map<String, Boolean> responseParameters = new HashMap<>();
    responseParameters.put("method.response.header.Access-Control-Allow-Headers", Boolean.TRUE);
    responseParameters.put("method.response.header.Access-Control-Allow-Methods", Boolean.TRUE);
    responseParameters.put("method.response.header.Access-Control-Allow-Credentials", Boolean.TRUE);
    responseParameters.put("method.response.header.Access-Control-Allow-Origin", Boolean.TRUE);
    methodResponses.add(MethodResponse.builder()
        .responseParameters(responseParameters)
        .statusCode("200")
        .build());
    MethodOptions methodOptions = MethodOptions.builder()
        .methodResponses(methodResponses)
        .build();

    Map<String, String> requestTemplate = new HashMap<>();
    requestTemplate.put("application/json", "{\"statusCode\": 200}");
    List<IntegrationResponse> integrationResponses = new ArrayList<>();
```

Postman Call API Gateway

The screenshot shows the Postman application interface. On the left, a sidebar lists API endpoints under the category 'CDK_API_GATEWAY_ORDERS'. The 'CREATE ITEM' endpoint is selected and highlighted with a pink dashed box. Below the sidebar, the main workspace displays a 'POST / CREATE ITEM' request. The 'Body' tab is active, showing a JSON payload. The URL for the request is 'https://kp90ktq1qb.execute-api.us-east-1.amazonaws.com/prod/items'. The JSON body is as follows:

```
1 [ {  
2     "orderDate": "2023-09-07",  
3     "status": "IN_PROGRESS",  
4     "desc": "Xiomai readmi 12C",  
5     "updateOrderDate": "2023-09-07",  
6     "Name": "Weder Sousa2",  
7     "Email": "weder.sousa96@handson.cloud"  
8 } ]
```

Postman Call API Gateway

The screenshot shows the Postman interface with a dark theme. On the left, a sidebar lists a collection named "CDK_API_GATEWAY_ORDERS" containing several API endpoints: "CREATE ITEM" (POST), "GET ALL ITEMS" (GET), "GET ITEM BY ID" (GET), "DELETE ITEM BY ID" (DEL), and "UPDATE ITEM" (PATCH). The "UPDATE ITEM" endpoint is highlighted with a pink dashed box. The main workspace shows a detailed view of the "UPDATE ITEM" request. The method is set to "PATCH" and the URL is "https://kp90ktq1qb.execute-api.us-east-1.amazonaws.com/prod/items/c0fe2aa0-a1d0-4d86-8091-ff76dda73086". The "Body" tab is selected, showing a JSON payload:

```
1 {  
2   "status": "CANCELED",  
3   "updateOrderDate": "2023-09-08"  
4 }
```

Postman Call API Gateway

The screenshot shows the Postman application interface. On the left, a sidebar lists several API endpoints under the category "CDK_API_GATEWAY_ORDERS". The endpoints are:

- POST CREATE ITEM
- GET GET ALL ITEMS (highlighted with a purple dashed box)
- GET GET ITEM BY ID
- DEL DELETE ITEM BY ID
- PATCH UPDATE ITEM

The main panel displays the details for the "GET GET ALL ITEMS" request. The request method is "GET" and the URL is "https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items".

The "Params" tab is selected, showing a table with one row:

KEY	VALUE	DESCRIPTION

The "Body" tab is selected, showing a JSON response. The response is a single-element array containing one object with the following properties:

```
1  []
2  [
3  {
4    "orderDate": "2023-09-07",
5    "orderId": "0d16a630-c89c-4b1f-b03d-aa179c737d76",
6    "status": "IN_PROGRESS",
7    "updateOrderDate": "2023-09-07",
8    "Email": "weder.sousa96@handson.cloud",
9    "desc": "Xiomi readmi 12C",
10   "Name": "Weder Sousa2"
11 }
```

Postman Call API Gateway

The screenshot shows the Postman application interface. On the left, a sidebar lists several API endpoints under the category "CDK_API_GATEWAY_ORDERS". The endpoints include:

- POST CREATE ITEM
- GET GET ALL ITEMS
- GET GET ITEM BY ID** (highlighted with a dashed red box)
- DEL DELETE ITEM BY ID
- PATCH UPDATE ITEM

The main workspace displays a "GET ITEM BY ID" request. The method is set to "GET", and the URL is "https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items/0d16a630-c89c-4b1f-b03d-aa179c737d76". The "Headers" tab shows 6 headers. The "Body" tab is selected and displays a JSON response with line numbers:

```
1 {  
2   "Items": [  
3     {  
4       "orderDate": {  
5         "S": "2023-09-07"  
6       },  
7       "orderId": {  
8         "S": "0d16a630-c89c-4b1f-b03d-aa179c737d76"  
9       },  
10      "status": {  
11        "S": "IN_PROGRESS"  
12      },  
13      "updateOrderDate": {  
14        "S": "2023-09-07"  
15      },  
16      "Email": {  
17        "S": "weder.sousa96@handson.cloud"  
18      },  
19      "desc": {  
20        "S": "Xiomi readmi 12C"  
21      },  
22      "Name": {  
23        "S": "Weder Sousa2"  
24      }  
25    }  
26  ]  
27}
```

Postman Call API Gateway

The screenshot shows the Postman interface with a request to a CDK API Gateway endpoint. The request method is DELETE, targeting the URL `https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items/0d16a630-c89c-4b1f-b03d-aa179c737d76`. The response body is a JSON object containing metadata about the successful item deletion.

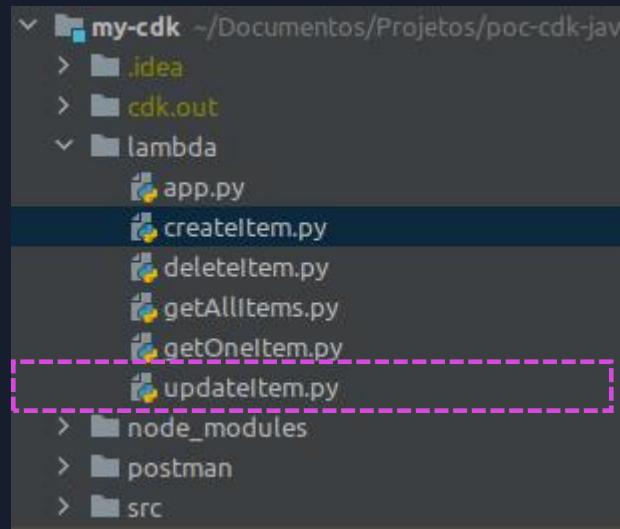
```
1 Item delete with Successfully : []
2 "ResponseMetadata": {
3     "RequestId": "2M1885SIFULQH3HU404EJ82R2BVV4KQNS05AEMVJF6609ASUAAJG",
4     "HTTPStatusCode": 200,
5     "HTTPHeaders": {
6         "server": "Server",
7         "date": "Mon, 11 Sep 2023 21:17:00 GMT",
8         "content-type": "application/x-amz-json-1.0",
9         "content-length": "2",
10        "connection": "keep-alive",
11        "x-amzn-requestid": "2M1885SIFULQH3HU404EJ82R2BVV4KQNS05AEMVJF6609ASUAAJG",
12        "x-amz-crc32": "2745614147"
13    },
14    "RetryAttempts": 0
15 }
16 ]
```

Lambda Function Python(Boto3) - Create Item

The image shows a terminal window and a code editor side-by-side. The terminal on the left displays a file tree for a project named 'my-cdk' located at '/Documentos/Projetos/poc-cdk-jav'. The 'lambda' directory contains several files: 'app.py', 'createItem.py' (which is selected and highlighted with a dashed blue border), 'deleteItem.py', 'getAllItems.py', 'getOneItem.py', and 'updateItem.py'. Below these are 'node_modules', 'postman', and 'src'. The code editor on the right contains the content of 'createItem.py'. The code uses Boto3 to interact with a 'orders' table in DynamoDB. It starts by importing json, boto3, and uuid. It then defines a lambda_handler function that prints the input event, creates a table resource, generates a UUID, prints it, and loads the event body as JSON. It then uses the table's put_item method to insert a new item with fields like orderId, orderDate, status, desc, updateOrderDate, Name, and Email. Finally, it constructs a response object with a statusCode of 200, a body indicating success, and appropriate headers.

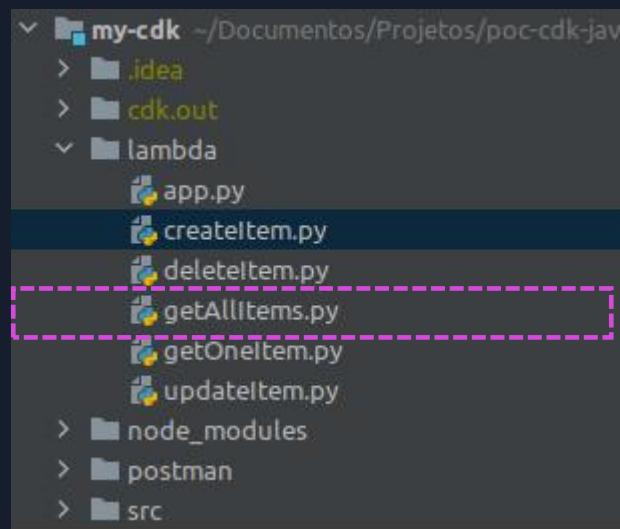
```
1 import json
2 import boto3
3 import uuid
4
5 client = boto3.resource('dynamodb')
6
7 def lambda_handler(event, context):
8     print('Starting input lambda function call')
9     table = client.Table('orders')
10    myuuid = str(uuid.uuid4())
11
12    print('Your UUID is: ' + str(myuuid))
13
14    order=json.loads(event['body'])
15
16    data = table.put_item(
17        Item={
18            'orderId': str(myuuid),
19            'orderDate': order['orderDate'],
20            'status': order['status'],
21            'desc': order['desc'],
22            'updateOrderDate': order['updateOrderDate'],
23            'Name': order['Name'],
24            'Email': order['Email']
25        }
26    )
27
28    response = {
29        'statusCode': 200,
30        'body': json.dumps('successfully created item!'),
31        'headers': { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*'}
32    }
33
34    return response
```

Lambda Function Python(Boto3) - Update Item



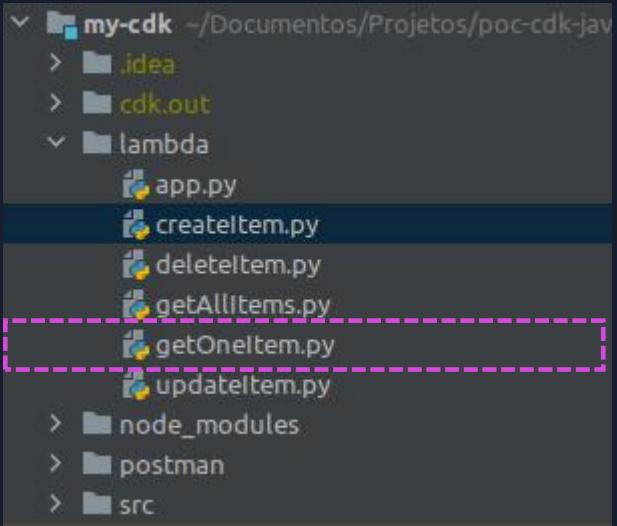
```
1 import json
2 import boto3
3 from boto3.dynamodb.conditions import Key
4
5 def lambda_handler(event, context):
6     print('Starting input lambda function call')
7
8     client = boto3.client('dynamodb')
9
10    print(event['pathParameters']['id'])
11    idParam = event['pathParameters']['id']
12
13    order=json.loads(event['body'])
14
15    # put (idempotent)
16    data = client.update_item(
17        TableName="orders",
18        Key={
19            "orderId": {"S": str(idParam)},
20        },
21        ExpressionAttributeValues={
22            ":val1": {"S": order['status']},
23            ":val2": {"S": order['updateOrderDate']},
24        },
25        ExpressionAttributeNames={
26            "#updateOrderDate": "updateOrderDate",
27            "#status": "status"
28        },
29        UpdateExpression="set #status = :val1, #updateOrderDate = :val2 ",
30        ReturnValues="ALL_NEW"
31    )
32
33    print(data)
34
```

Lambda Function Python(Boto3) - Get All Item



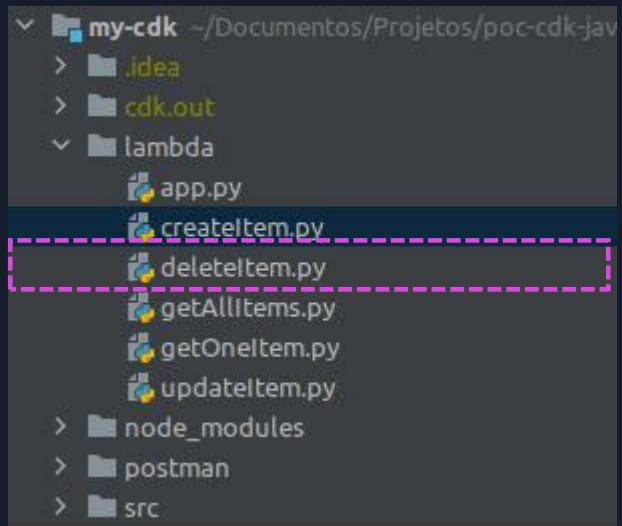
```
1 import json
2 import boto3
3 from boto3.dynamodb.conditions import Key, Attr
4
5 client = boto3.resource('dynamodb')
6
7 def lambda_handler(event, context):
8     print('Starting input lambda function call')
9     print(event)
10
11     table = client.Table('orders')
12     #responseScan = table.scan(FilterExpression=Attr('Name').eq('Weder Sousa'))
13     responseScan = table.scan()
14
15     print("The query returned the following items:")
16     for item in responseScan['Items']:
17         print(item)
18
19     response = {
20         'statusCode': 200,
21         'body': json.dumps(responseScan['Items']),
22         'headers': {
23             'Content-Type': 'application/json',
24             'Access-Control-Allow-Origin': '*'
25         },
26     }
27
28     return response
```

Lambda Function Python(Boto3) - Get One Item By Id



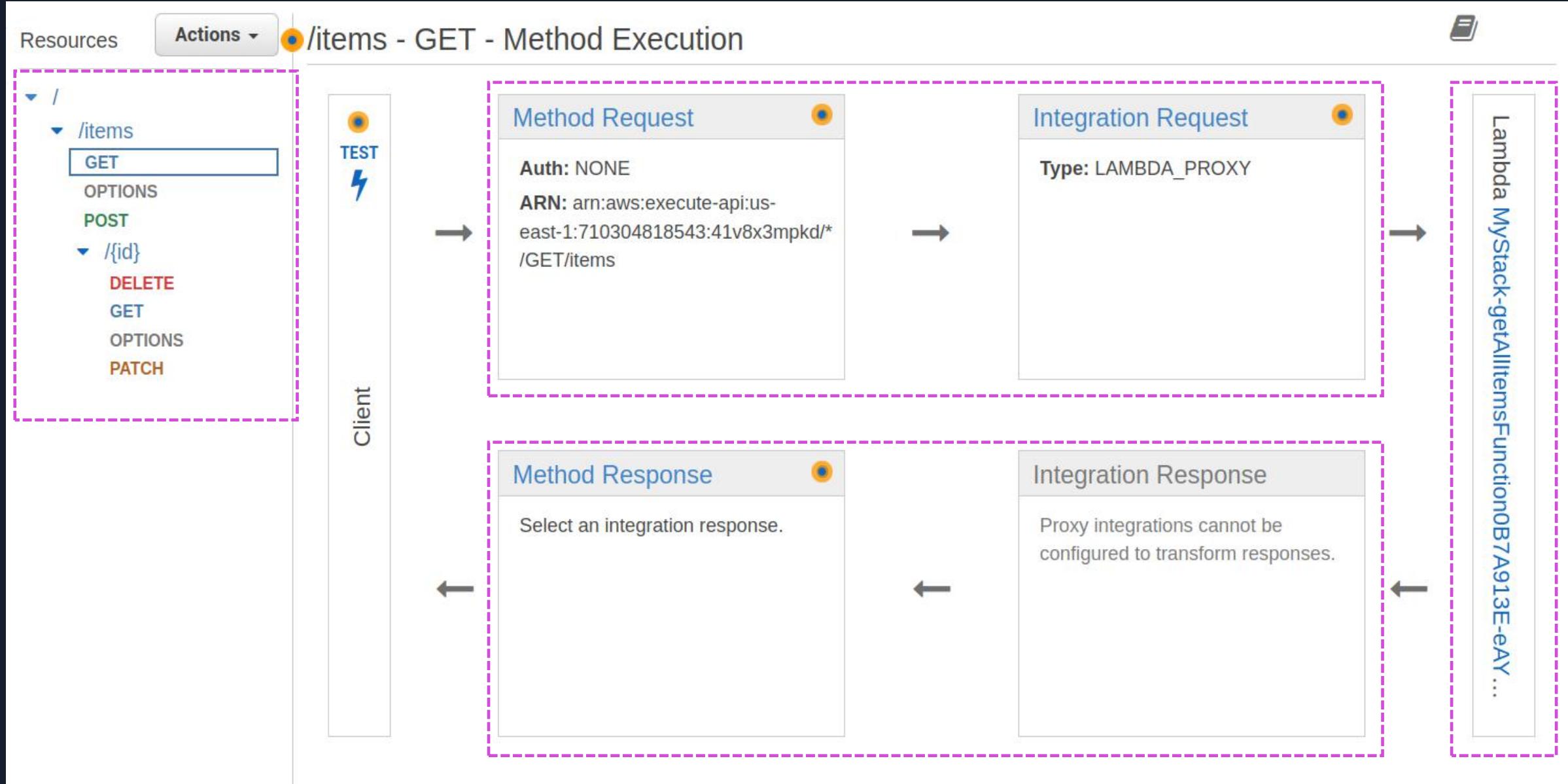
```
1 import json
2
3 import boto3
4 from boto3 import dynamodb
5 from boto3.dynamodb.conditions import Key, Attr
6
7
8 def lambda_handler(event, context):
9     print('Starting input lambda function call')
10    print(event)
11    print(event['pathParameters']['id'])
12
13    idParam = event['pathParameters']['id']
14    client = boto3.client('dynamodb')
15    # item = client.get_item(TableName='orders', Key={'orderId': { 'S' : str(idParam)}})
16
17    item = client.query(
18        ExpressionAttributeValues={
19            ':vId': {
20                'S': str(idParam)
21            },
22        },
23        KeyConditionExpression='orderId = :vId',
24        TableName='orders'
25    )
26
27
28    print("The query returned the following item:")
29    print(item)
30
```

Lambda Function Python(Boto3) - Delete Item By Id

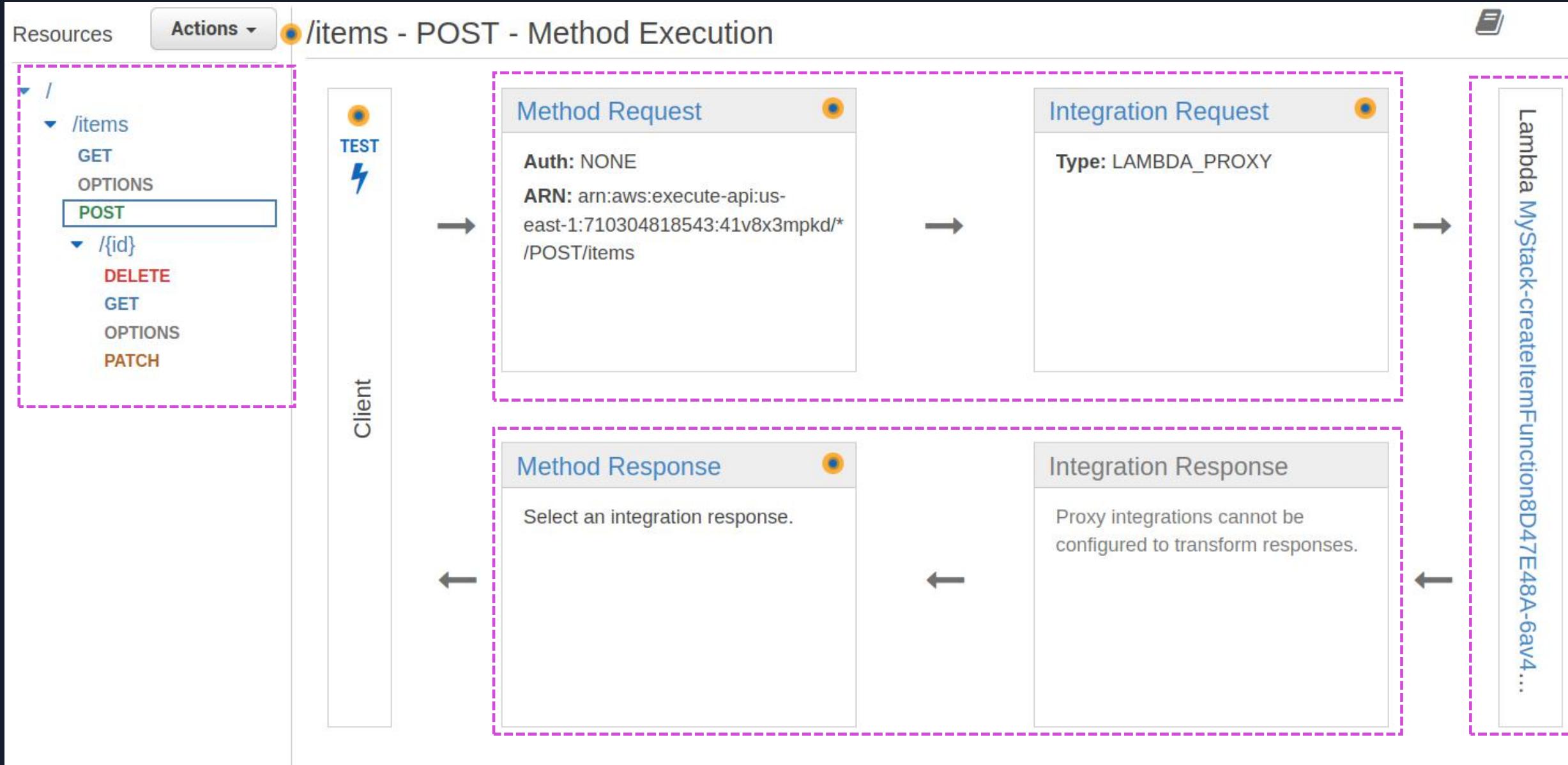


```
1 import json
2 import boto3
3 from boto3 import dynamodb
4 from boto3.dynamodb.conditions import Key, Attr
5
6
7 def lambda_handler(event, context):
8     print(event['pathParameters']['id'])
9
10    idParam = event['pathParameters']['id']
11    client = boto3.client('dynamodb')
12
13    item = client.query(
14        ExpressionAttributeValues={':vId': { 'S': str(idParam)}},
15        KeyConditionExpression='orderId = :vId',
16        TableName='orders'
17    )
18
19    if len(item["Items"]) > 0:
20        print(item["Items"][0]["orderDate"]["S"])
21        deleteItem = client.delete_item(
22            TableName='orders' ,
23            Key={
24                'orderId': { 'S' : str(idParam)},
25                'orderDate': { 'S' : str(item["Items"][0]["orderDate"]["S"])}
26            }
27        )
28
29        print("The query returned the following item:")
30        print(deleteItem)
```

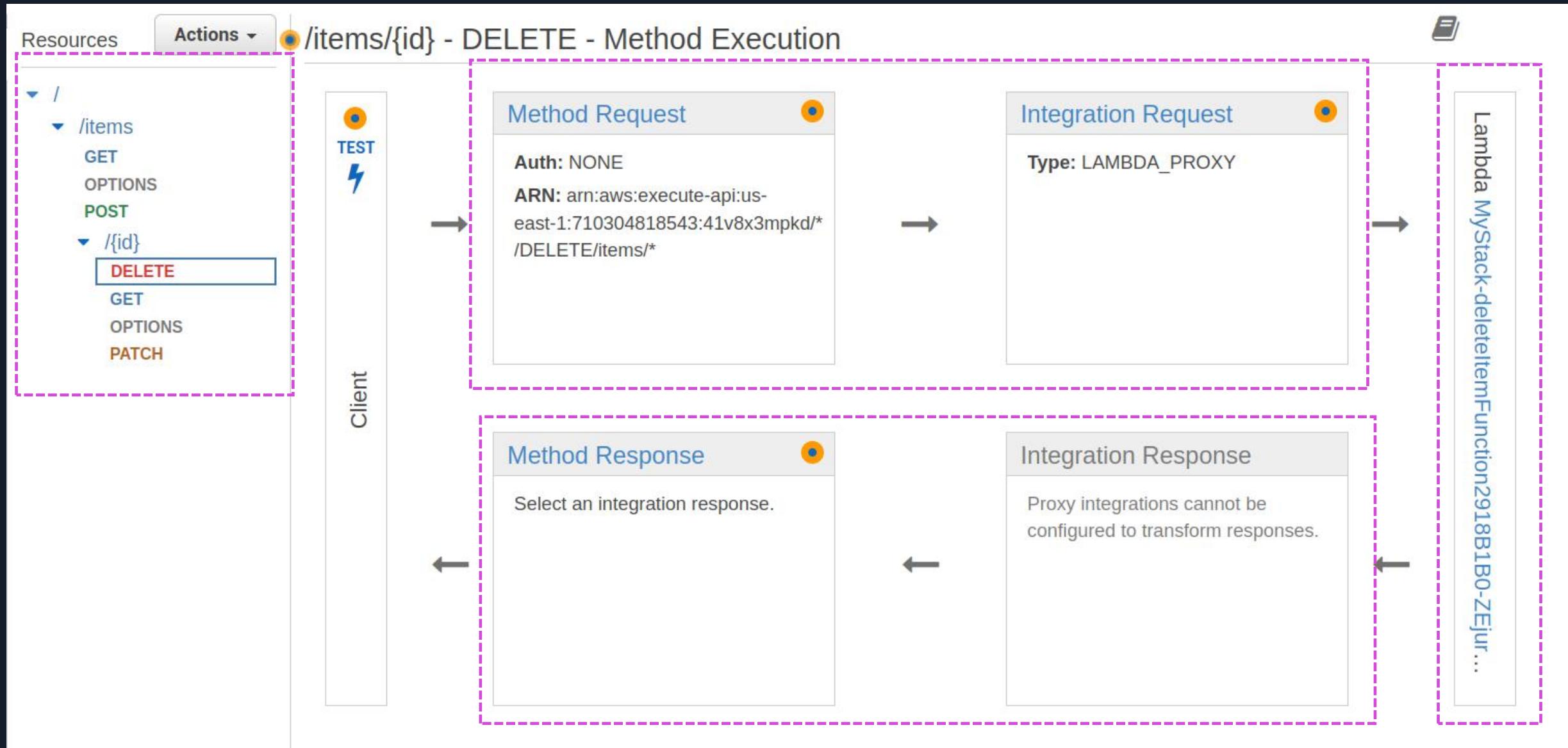
AWS Resources (API Gateway) create(GET)



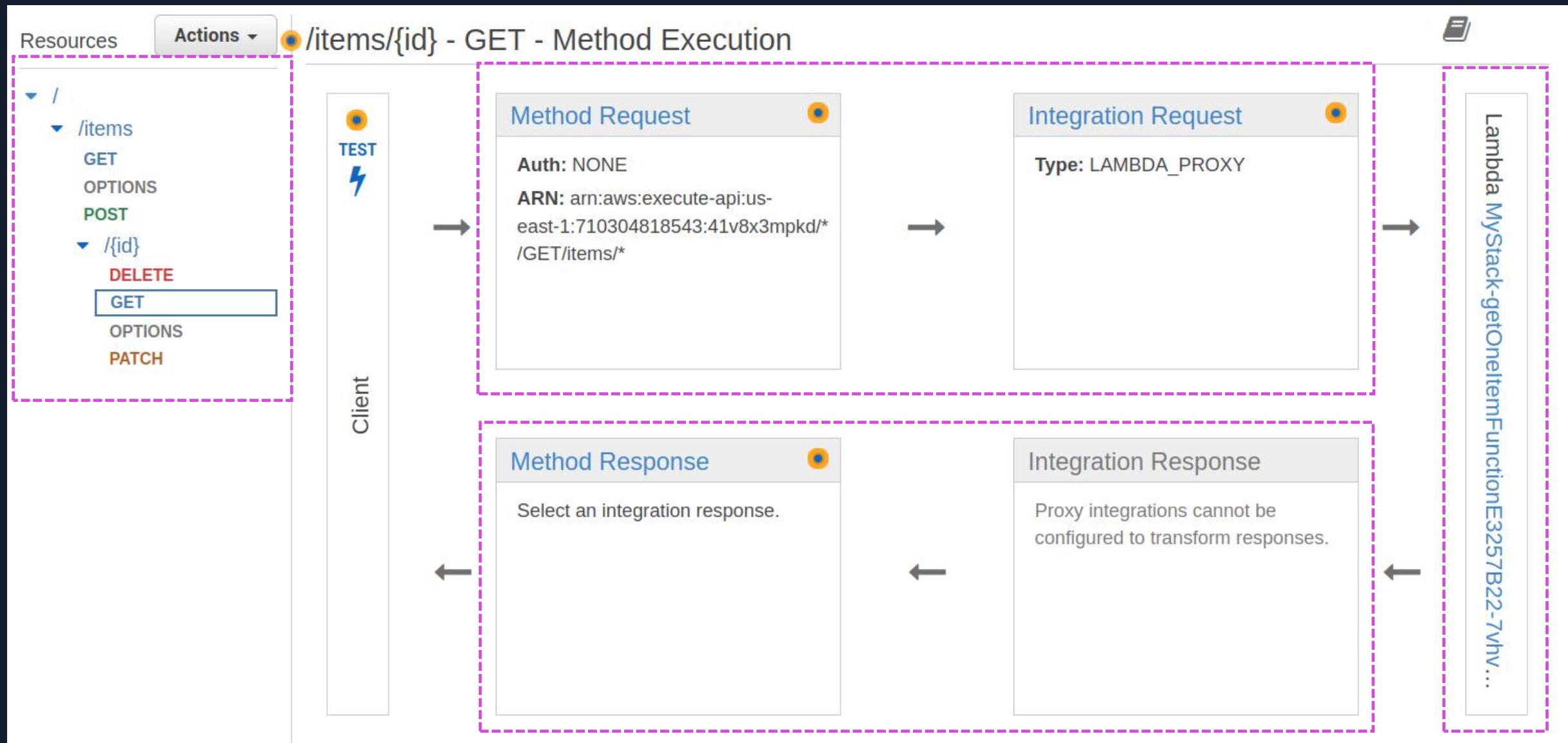
AWS Resources (API Gateway) create (POST)



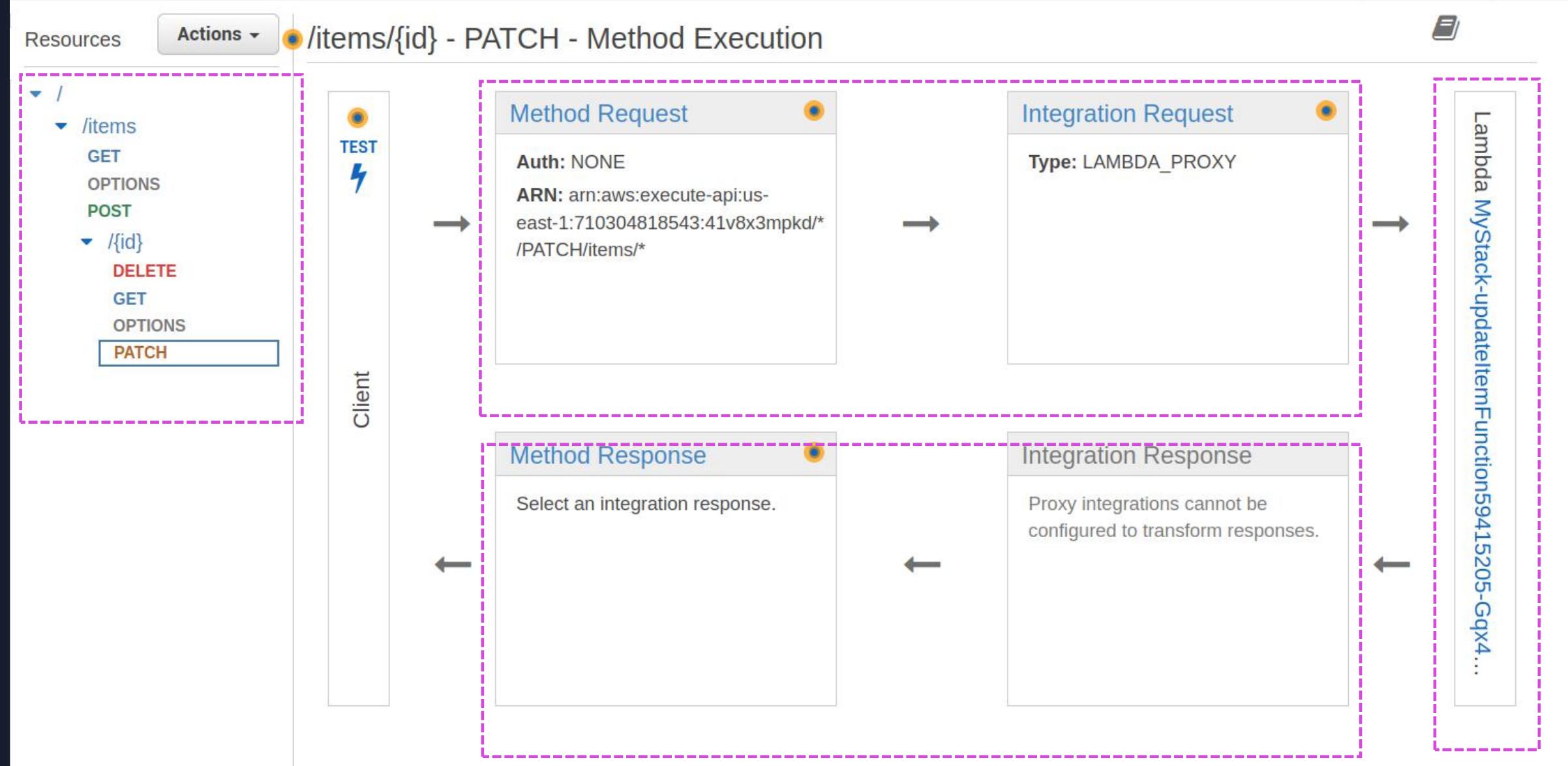
AWS Resources (API Gateway) create /id (DELETE)



AWS Resources (API Gateway) create /id (GET)



AWS Resources (API Gateway) create /id (PATCH)



AWS Resources create (Lambdas)

Lambda > Functions

Functions (30)					Last fetched 23 seconds ago	C	Actions ▾	Create function	
<input type="text"/> Filter by tags and attributes or search by keyword						<	1	>	⚙️
Function name	Description	Package type	Runtime	Last modified					
MyStack-deleteItemFunction2918B1B0-ZEjurzRhyTEP	Lambda deleteItem	Zip	Python 3.10	16 hours ago					
MyStack-getOneItemFunctionE3257B22-7vhvfNEQGTxk	Lambda getOneItem	Zip	Python 3.10	20 hours ago					
MyStack-updateItemFunction59415205-Gqx4MKt4wOxX	Lambda updateItem	Zip	Python 3.10	4 days ago					
MyStack-createItemFunction8D47E48A-6av4PrReEqoU	Lambda createItem	Zip	Python 3.10	4 days ago					
MyStack-getAllItemsFunction0B7A913E-eAYHHix0qcMN	Lambda getAllItems	Zip	Python 3.10	4 days ago					

AWS Resources create (DynamoDB)

DynamoDB > Tables

Tables (1) [Info](#)

C Actions ▾ Delete Create table

Find tables by table name Any tag key Any tag value < 1 > ⚙

Na...	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode
<input type="checkbox"/> orders	<input checked="" type="checkbox"/> Active	orderId (S)	orderDate (S)	0	<input type="checkbox"/> Off	Provisioned (5)	Provisioned (5)

DynamoDB > Explore items > orders

Tables (1) X orders Autopreview View table details

Any tag key Any tag value Find tables by table name < 1 > ⚙

Scan or query items Expand to query or scan items.

Completed. Read capacity units consumed: 0.5

Items returned (1)

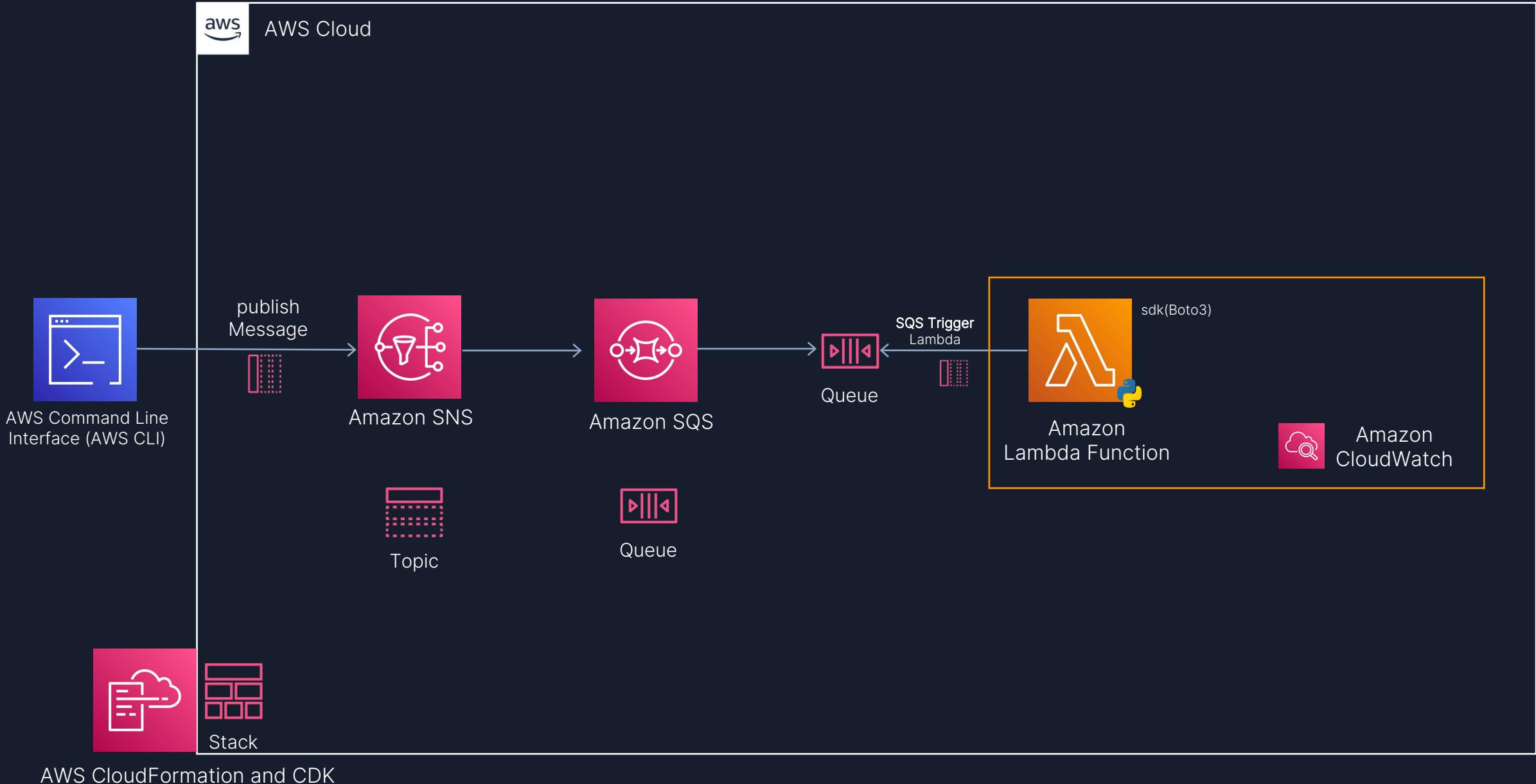
C Actions ▾ Create item < 1 > ⚙

orderId (String)	orderDate (String)	desc	Email	Name
db9f81d5-fcfe-4bc4-a201-9051175dc256	2023-09-12	Xiomi read...	weder.sous...	Weder So...

AWS Resources create (IAM Role)

Roles (37) <small>Info</small>			
<small>An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.</small>			
<input type="checkbox"/>	Role name	Trusted entities	Last activity
<input type="checkbox"/>	AWSServiceRoleForSupport	AWS Service: support (Service-Linker)	-
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linker)	-
<input type="checkbox"/>	cdk-hnb659fds-cfn-exec-role-710304818543-us-east-1	AWS Service: cloudformation	3 days ago
<input type="checkbox"/>	cdk-hnb659fds-deploy-role-710304818543-us-east-1	Account: 710304818543	3 days ago
<input type="checkbox"/>	cdk-hnb659fds-file-publishing-role-710304818543-us-east-1	Account: 710304818543	3 days ago
<input type="checkbox"/>	cdk-hnb659fds-image-publishing-role-710304818543-us-east-1	Account: 710304818543	-

Proof of Concepts AWS CDK - Using SNS, SQS and Lambda



Proof of Concepts AWS CDK



The purpose of this PoC is to understand how to use the CDK **to generate the infra(JAVA)** of a **SNS, SQS(DeadLetter) and lambdas(Python) with lambda trigger** its advantages and limitations of **traditional use with the creation of a Cloudformation(SAM) template**.

Files

main

Go to file

my-cdk

lambda

lambdaSqs

app.py

node_modules

postman

src

Octree

main/java/com/myorg

CreateBucket.java

CreateDynamoDb.java

CreateLambdaCron.java

CreateSqsAndSns.java

MyCdkApp.java

MyStack.java

test/java/com/myorg

.gitignore

README.md

poc-cdk-java / my-cdk / src / main / java / com / myorg / CreateSqsAndSns.java

Code Blame 103 lines (81 loc) · 4.14 KB Code 55% faster with GitHub Copilot

```
public class CreateSqsAndSns {  
    public void cdkSqs(MyStack myStack){  
        // 🚧 create queue  
        Queue dQueue = Queue.Builder.create(myStack, "cdk-dead-letter-queue-id")  
            .retentionPeriod(Duration.days(7))  
            .build();  
  
        DeadLetterQueue deadLetterQueue = DeadLetterQueue.builder().queue(dQueue)  
            .maxReceiveCount(1)  
            .build();  
  
        Queue sQueue = Queue.Builder.create(myStack, "cdk-upload-queue-id")  
            .deadLetterQueue(deadLetterQueue)  
            .visibilityTimeout(Duration.seconds(300))  
            .build();  
  
        // 🚧 create sns topic  
        Topic snsTopic = Topic.Builder.create(myStack,"cdk-sns-topic")  
            //.topicName("cdk-sns-topic")  
            //.fifo(true)  
            .build();  
  
        // 🚧 create Subscription add Topic  
        SqsSubscription sqsSubscription = SqsSubscription.Builder.create(sQueue).build();  
        snsTopic.addSubscription(sqsSubscription);  
  
        // 🚧 create Lambda Function  
        SingletonFunction function = createLambdaFunction(myStack);  
        function.addEventSource(SqsEventSource.Builder.create(sQueue).build());  
    }  
}
```

<https://github.com/weder96/poc-cdk-java>



Next Steps



The AWS Step Functions Workshop



The AWS Step Functions Workshop

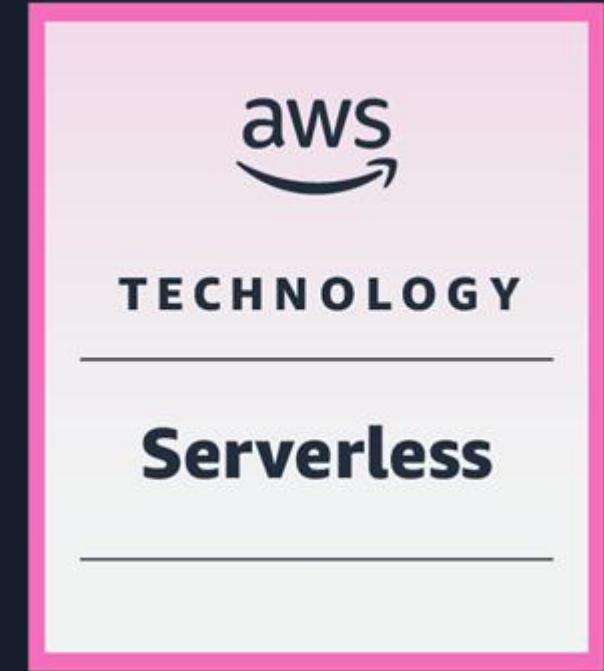


© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

<https://catalog.workshops.aws/stepfunctions/en-US>

AWS Serverless Digital Learning Badges

[AWS Skill Builder](#)



<https://aws.amazon.com/blogs/compute/introducing-new-aws-serverless-digital-learning-badges/>

Want to Try?

Serverless Security Workshop



Serverless Security Workshop

Serverless Security Workshop



Important

The purpose of the workshop is to provide a starter API which **does NOT follow many security best practices** on purpose. The tutorial modules guide you to identify security gaps in the starter app, and implement protection measures for them. Furthermore, the modules **do not cover ALL** the security measures that should be applied. After completing all modules, we recommend you to explore additional protections, such as ensuring the principle of least privilege. See the **Extra Credit** section for more details.

In this workshop, you will learn techniques to secure a serverless application built with AWS Lambda, Amazon API Gateway and RDS Aurora. We will cover AWS services and features you can leverage to improve the security of a serverless applications in 5 domains:

1. identity & access management
2. code
3. data
4. infrastructure
5. logging & monitoring

You'll start by deploying a simple serverless application that allows third party companies to submit unicorn customizations. This will help Wild Rydes receive ad revenue and allow third party companies to market their brand leveraging Wild Rydes's popularity.

<https://catalog.us-east-1.prod.workshops.aws/workshops/026f84fd-f589-4a59-a4d1-81dc543fc30/en-US>

Want to Try?

Serverland AWS Lambda Fundamentals



AWS Lambda Fundamentals



s12d.com/lambda-fundamentals

<https://s12d.com/lambda-fundamentals>

Want to Try?

Serverless airline - Multiple patterns/practices



The image displays four sequential screenshots of a mobile application interface for a flight booking service:

- Screenshot 1: Search Screen**
Title: Flight App
Text: "Where next?"
Fields: "Departure airport: LGW", "Arrival airport: MAD", "Pick a date: Wed, 24 Apr 2019".
Button: "SEARCH FLIGHTS >"
- Screenshot 2: Flight Selection Screen**
Title: Flight App
Text: "Select your flight"
List:
 - DEPARTURE LGW London Gatwick → 16 JAN 2019 MAD Madrid Barajas
Time: 08:00 Duration: 2h15m Arrival: 11:15
Price: 400 EUR Flight No: #1812
 - DEPARTURE LGW London Gatwick → 16 JAN 2019 MAD Madrid Barajas
Time: 10:30 Duration: 2h15m Arrival: 13:45
Price: 200 EUR Flight No: #1813
 - DEPARTURE LGW London Gatwick → 16 JAN 2019 MAD Madrid Barajas
Time: 12:00 Duration: 2h15m Arrival: 15:15
Price: 1000 EUR Flight No: #1814
- Screenshot 3: Review Selection Screen**
Title: Flight App
Text: "Review your selection"
List:
 - DEPARTURE LGW London Gatwick → 16 JAN 2019 MAD Madrid Barajas
Time: 08:00 Duration: 2h15m Arrival: 11:15
Price: 400 EUR Flight No: #1812
- Screenshot 4: User Profile Screen**
Title: Flight App
Text:
 - Heitor F. Lessa (purple)
 - 4,554,234 Points
 - 10% Next Tier Progress

Payment details:
Name: Name on card
Country:
Postcode:
Card number: 1234 1234 1234 1234
Expiry date: MM / YY
CVC: CVC

Preferences:
Dietary requirements
Luggage

Buttons: "SIGN OUT"

<https://github.com/aws-samples/aws-serverless-airline-booking>

Want to Try? CDK - Workshop



AWS CDK
Workshop

Search

English

Prerequisites

- TypeScript Workshop
- Python Workshop
- .NET Workshop
- Java Workshop
- Go Workshop
- Construct Hub
- Congrats!

Submit a correction

CDK on GitHub

CDK on StackOverflow

CDK on Gitter

Welcome Developers!

Hey there, and thanks for joining us! Hope you can't wait to play with this new thing we call the "AWS Cloud Development Kit" or in short, the AWS CDK.

The AWS CDK is a new software development framework from AWS with the sole purpose of making it fun and easy to define cloud infrastructure in your favorite programming language and deploy it using AWS CloudFormation.

So what are we going to build? Nothing too fancy...

We'll spend some time setting up your development environment and learning a little about how to work with the CDK Toolkit to deploy your app to an AWS environment.

Then, you'll write a little "Hello, world" Lambda function and front it with an API Gateway endpoint so users can call it via an HTTP request.

Next, we'll introduce the powerful concept of **CDK constructs**. Constructs allow you to bundle up a bunch of infrastructure into reusable components which anyone can compose into their apps. We'll walk you through writing your own construct.



<https://cdkworkshop.com/>

Want to Try?

CDK Patterns



aws
CDK Patterns

[About](#)[Find A Pattern](#)

CDK Patterns is more than "just AWS CDK examples"

Check Out Our 4 Content Distribution Platforms:



Star 2,130

Follow

YouTube

The Practical Dev

<https://cdkpatterns.com/>

Want to Try?

hands-on.cloud

The screenshot shows the homepage of hands-on.cloud. At the top, there's a dark header with social media links (Facebook, YouTube, Twitter) and the site's logo, "hands-on.cloud". Below the header is a navigation bar with links: "Python Boto3 Tutorials", "Courses (In progress)", "AWS Certifications", "How-To Articles", "Write For Us", and a "serverless" dropdown menu. The main content area features a large, semi-transparent background image of a hand typing on a keyboard. Overlaid on this image is the site's title "Hands-On.Cloud" in a large, bold, dark blue serif font, followed by the subtitle "Tutorials, How-Tos for Cloud Engineers" in a smaller, lighter font. Below this, the word "serverless" is prominently displayed in a large, bold, dark blue sans-serif font. A search bar at the bottom left contains the word "SERVERLESS" and a magnifying glass icon. At the bottom right, a teal button provides the website URL: <https://hands-on.cloud/>. The overall design is clean and professional, focusing on cloud computing and developer resources.

Resources

<https://cdkworkshop.com>

<https://github.com/aws-samples/aws-cdk-examples>

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

<https://aws.amazon.com/pt/developer/language/java/>

<https://docs.aws.amazon.com/toolkit-for-jetbrains/latest/userguide/setup-toolkit.html>

<https://aws.amazon.com/pt/intellij/>

https://docs.aws.amazon.com/code-library/latest/ug/java_2_code_examples.html

https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/patterns/deploy-a-ci-cd-pipeline-for-java-microservices-on-amazon-ecs.html

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

<https://aws.amazon.com/pt/blogs/compute/java-17-runtime-now-available-on-aws-lambda/>

<https://www.slideshare.net/AmazonWebServices/java-on-aws>

<https://www.jrebel.com/blog/aws-java-application-setup>

<https://www.slideshare.net/VadymKazulkin/adopting-java-for-the-serverless-world-at-jax-2022>

<https://towardsaws.com/deploy-spring-boot-application-to-aws-ec2-using-docker-f359e7ad2026>

<https://aws.amazon.com/pt/blogs/developer/stepfunctions-fluent-api/>

<https://aws.amazon.com/blogs/compute/java-17-runtime-now-available-on-aws-lambda/>

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

About the Speaker

Weder Mariano de Sousa

Specialist Senior - GFT

Q & A



AWS
community
builder

Technician **System Development**

Graduated **Computer Science**

Post Graduate in **Midias UFG**

Post Graduate in **Information Security**

<https://www.linkedin.com/in/wedermarianodesousa/>

<https://github.com/weder96>

<https://twitter.com/weder96>

<https://dev.to/weder96>



GOJava



AWS User Group Goiânia



THANK YOU