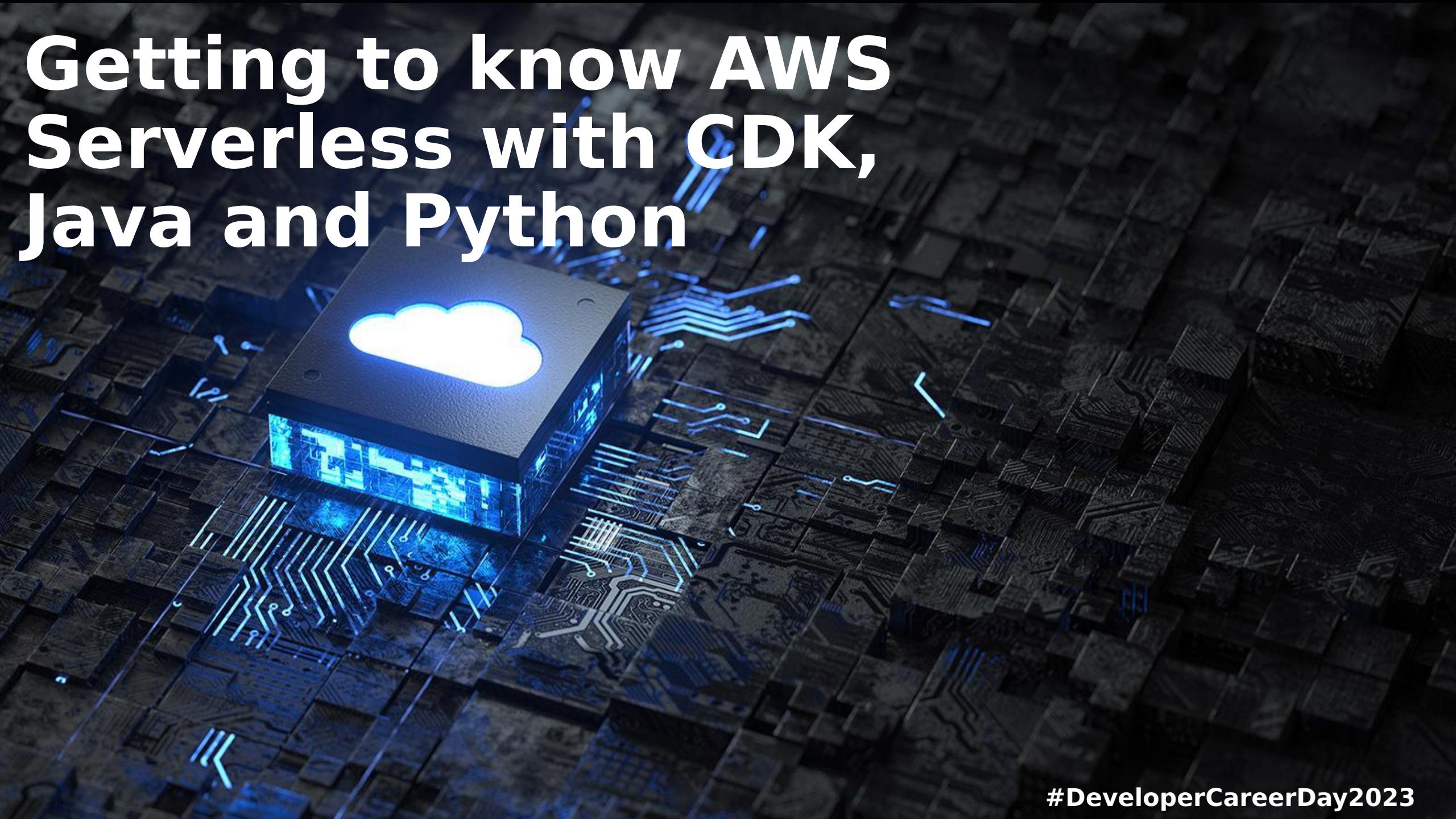


# Getting to know AWS Serverless with CDK, Java and Python





Me

# Weder Mariano de Sousa

Specialist Senior Java - GFT



GOJAVA



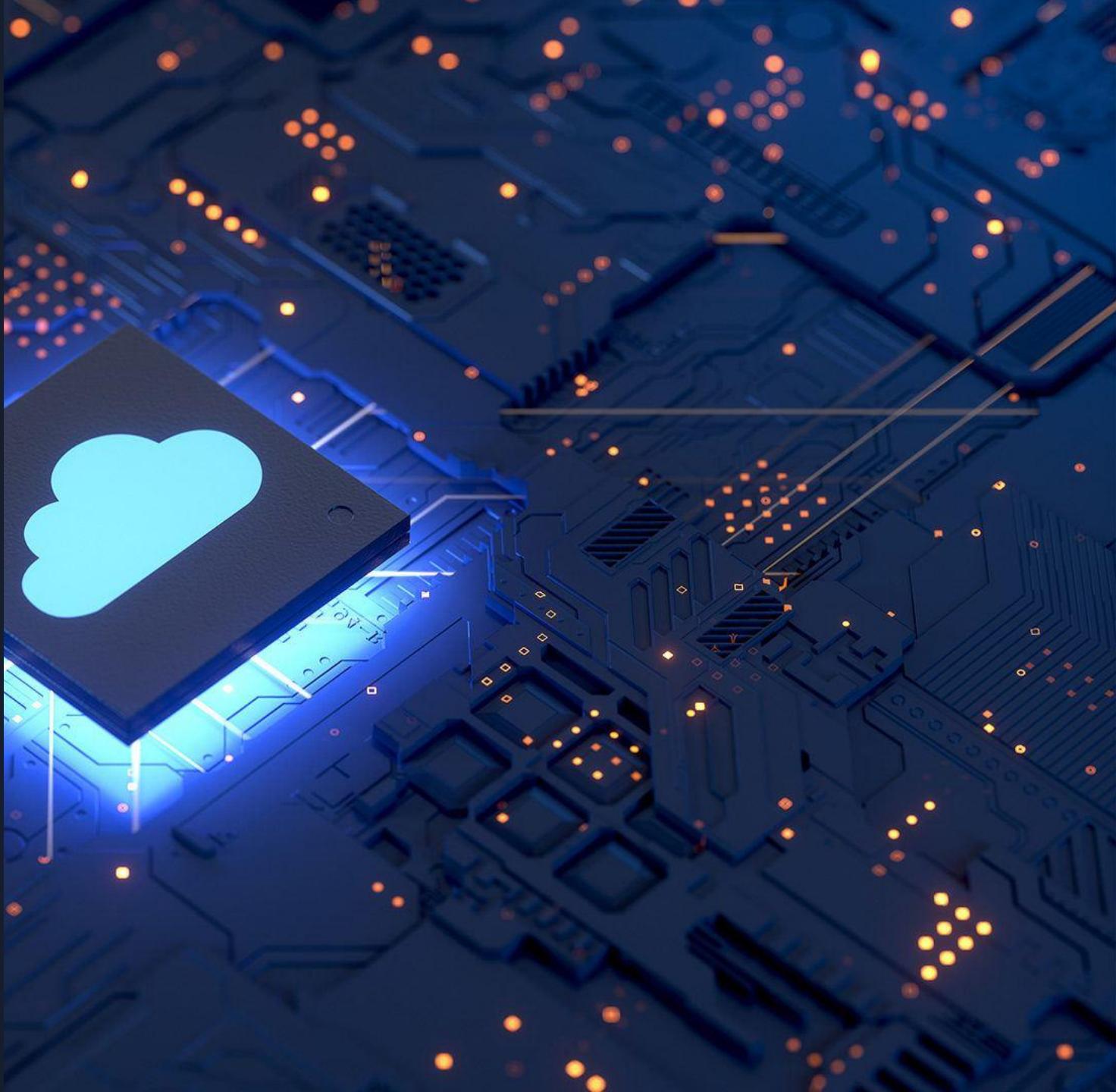
AWS User Group Goiânia



# Agenda

---

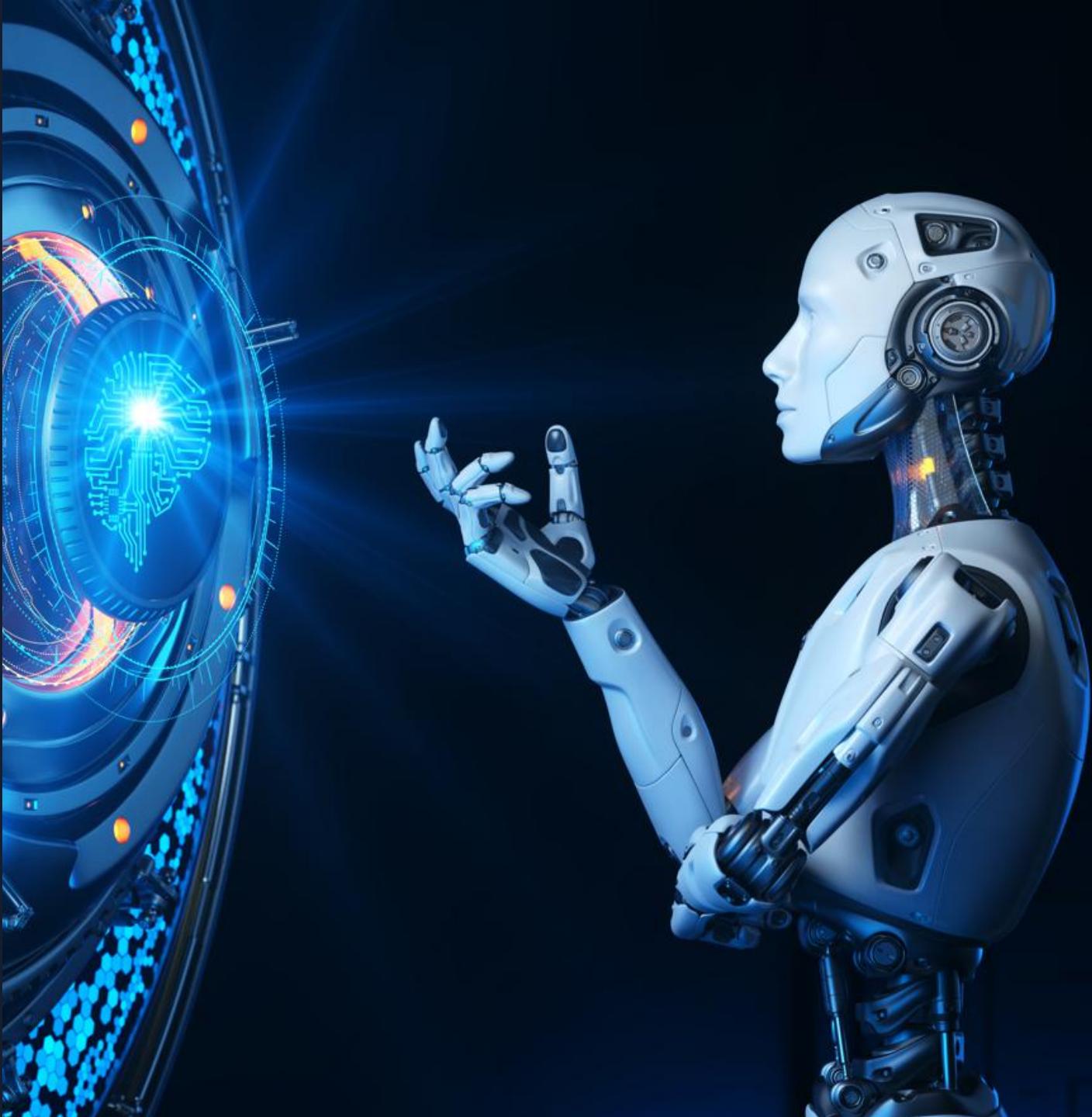
1. Question Artificial intelligence
2. Use Case
3. Requirement and Tools
4. Introduction About Serverless
5. Architecture Serverless
6. AWS CDK For Java and Python
7. Next Steps





# Question Artificial intelligence

Create a study routine focused on learning Serverless AWS?





# ChatGPT



**Week 1-2: Fundamentals and AWS Basics**

**Week 3-4: AWS Lambda**

**Week 5: AWS API Gateway**

**Week 6: AWS Step Functions**

**Week 7-8: Serverless Application Development**

**Week 9-10: Advanced Topics and Best Practices**

**Week 11-12: Real-world Projects and Certifications**

**Ongoing**

**Join AWS forums and communities.**

**Follow AWS blogs and attend webinars.**

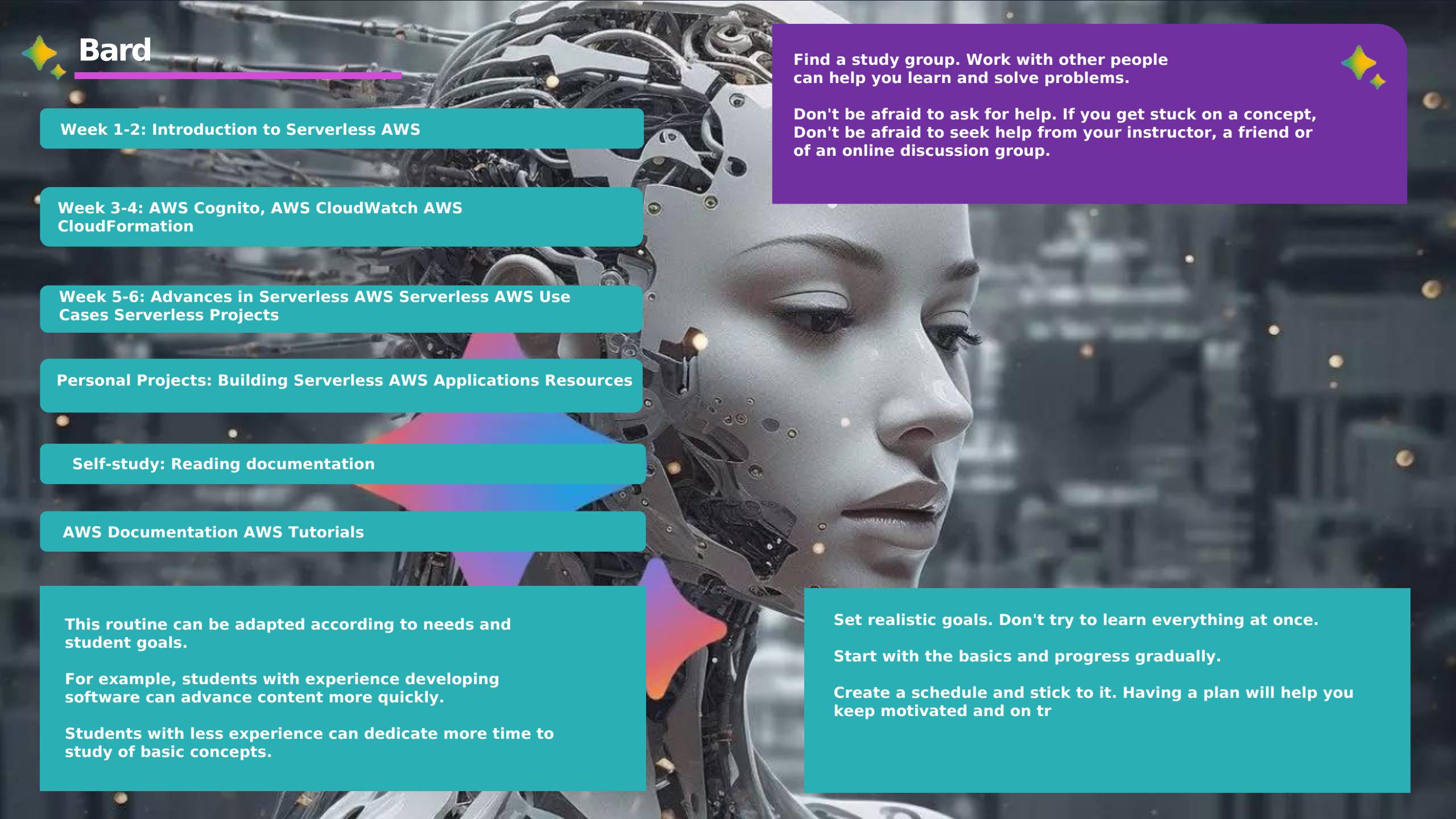
**Keep experimenting with new AWS services and features.**

**Stay updated with AWS documentation and best practices.**

**ChatGPT**

**Remember to adapt this study routine to your pace and background knowledge.**

**Regular practice and hands-on experience are key to becoming proficient in Serverless AWS.**



Week 1-2: Introduction to Serverless AWS

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

Week 5-6: Advances in Serverless AWS Serverless AWS Use Cases Serverless Projects

Personal Projects: Building Serverless AWS Applications Resources

Self-study: Reading documentation

AWS Documentation AWS Tutorials

This routine can be adapted according to needs and student goals.

For example, students with experience developing software can advance content more quickly.

Students with less experience can dedicate more time to study of basic concepts.

Find a study group. Work with other people can help you learn and solve problems.

Don't be afraid to ask for help. If you get stuck on a concept, Don't be afraid to seek help from your instructor, a friend or of an online discussion group.

Set realistic goals. Don't try to learn everything at once.

Start with the basics and progress gradually.

Create a schedule and stick to it. Having a plan will help you keep motivated and on tr

# Use Case

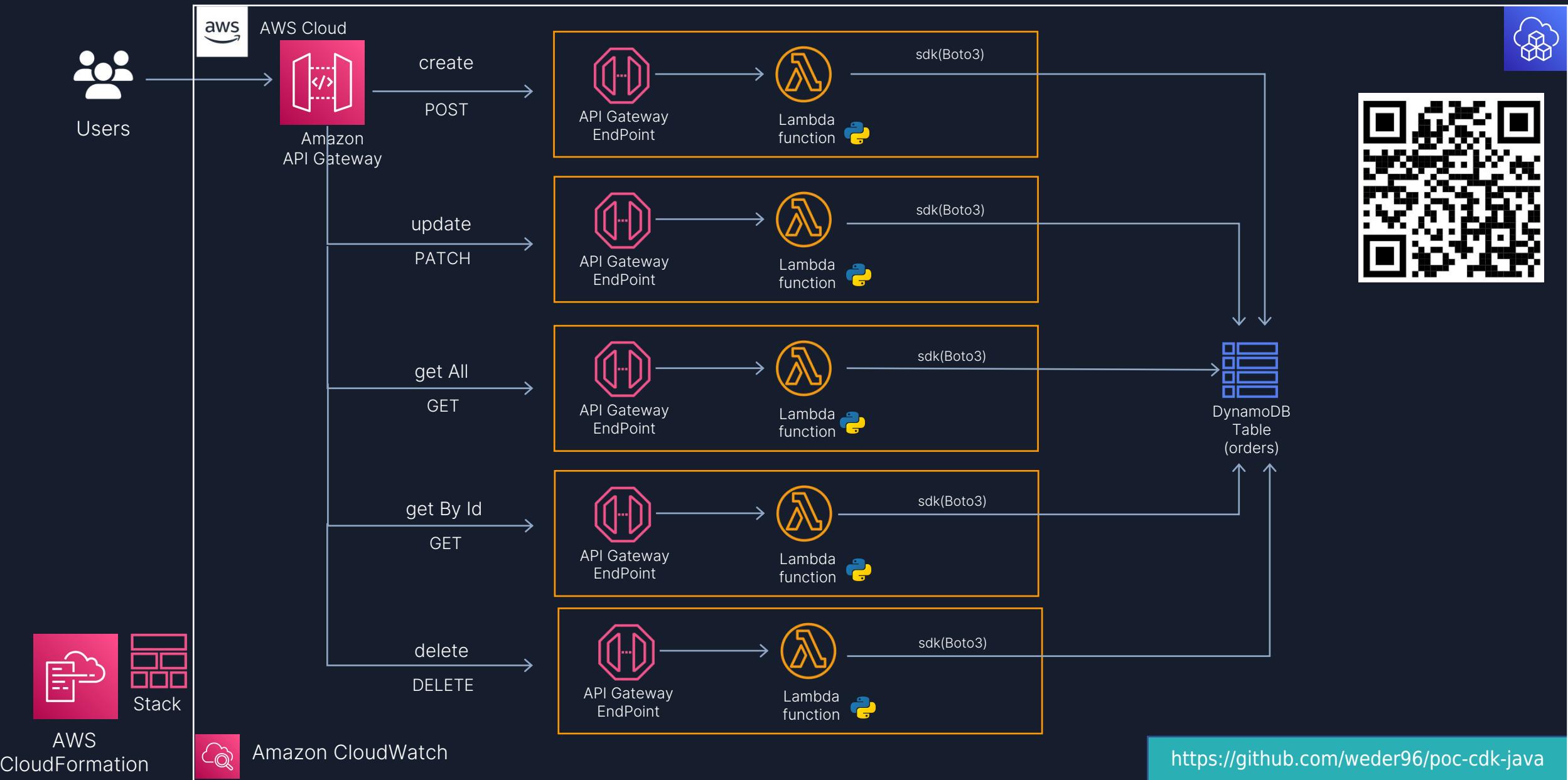
**CRUD - Register Items  
using AWS Serverless**

Don't worry  
about infrastructure

DON'T JUST WISH FOR IT  
**WORK FOR IT**



# Use Case AWS CDK - CRUD





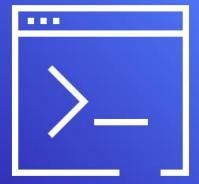
# Requirement and Tools



## Requirement and Tools

---

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



# 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.

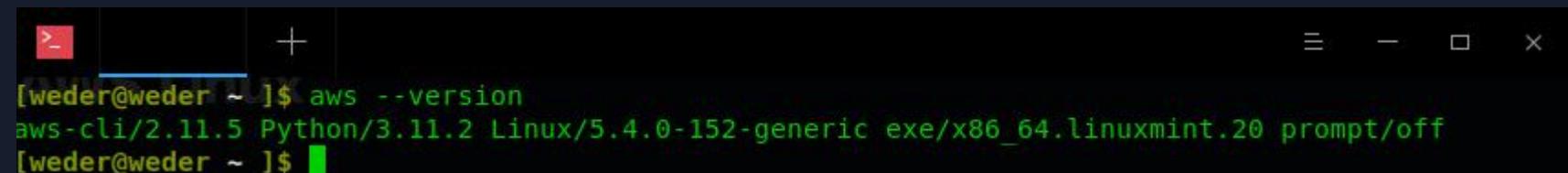
We provide the steps in one easy to copy and paste group based on whether you use 64-bit Linux or Linux ARM. See the descriptions of each line in the steps that follow.

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.

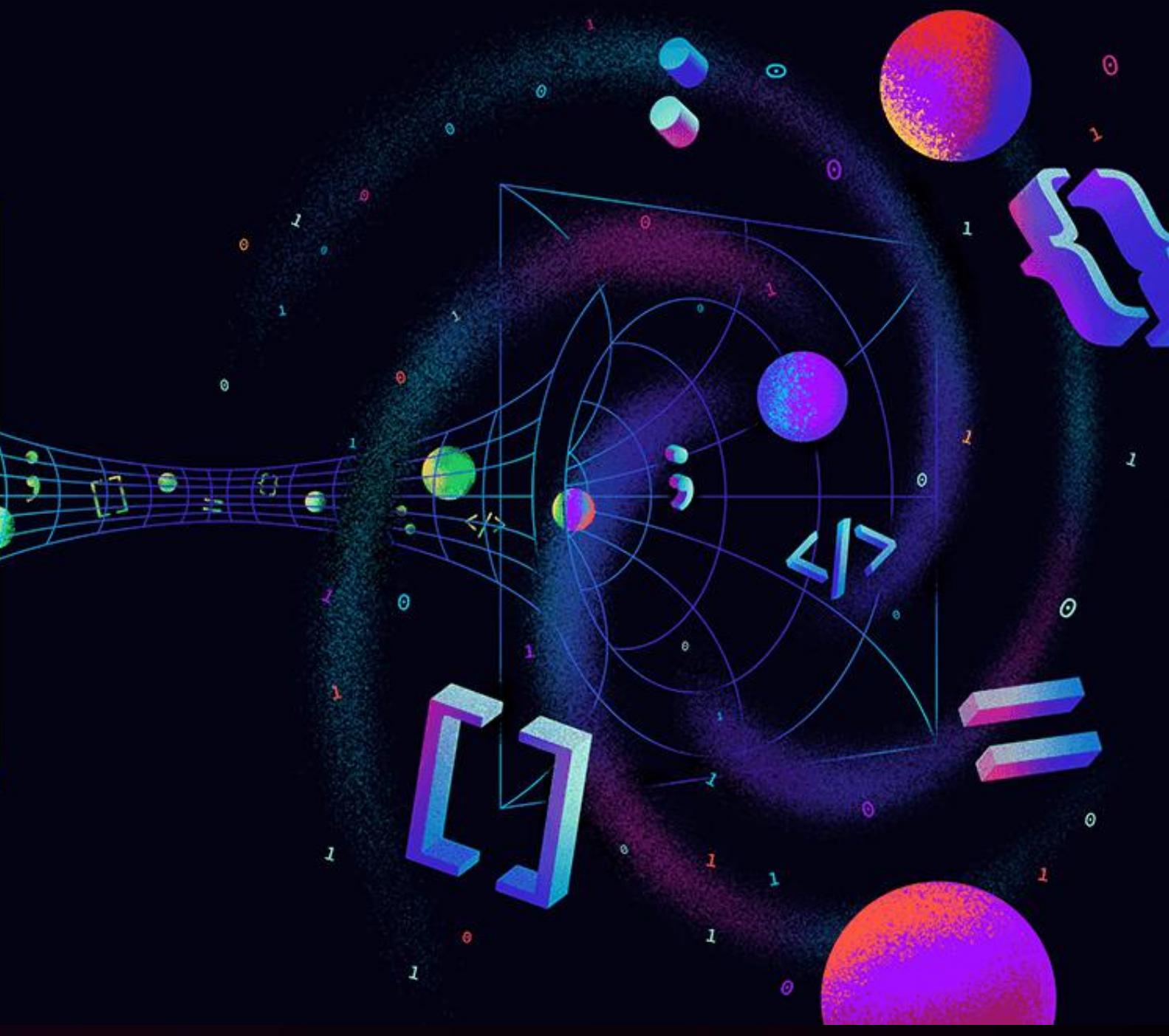
We can't guarantee that they contain the latest version of the AWS CLI.



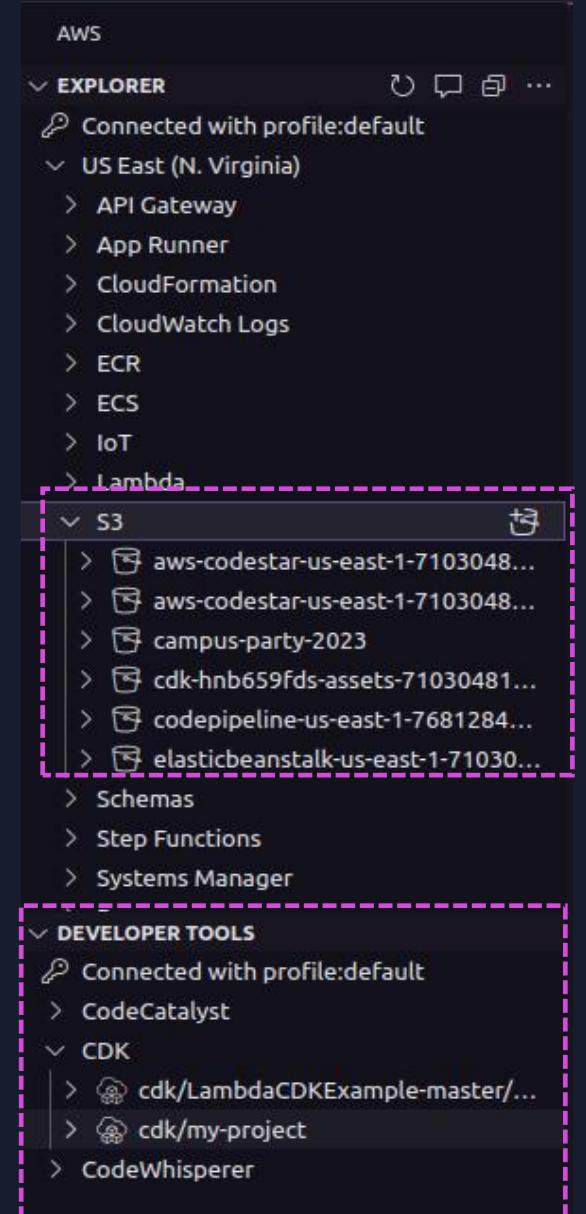
A screenshot of a terminal window titled 'Terminal'. The window shows the AWS CLI installed and running. The command 'aws --version' is entered, and the output is: 'aws-cli/2.11.5 Python/3.11.2 Linux/5.4.0-152-generic exe/x86\_64.linuxmint.20 prompt/off'. The terminal has a dark background with light-colored text. The title bar says 'Terminal'.



# AWS IDE Toolkits



# AWS Toolkit for Visual Studio Code



[https://aws.amazon.com/serverless/?nc1=h\\_ls](https://aws.amazon.com/serverless/?nc1=h_ls)  
<https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.aws-toolkit-vscode>

# What is Toolkits IDE AWS ?

Settings

Q+ plu

Plugins Marketplace Installed

Appearance & Behavior

- System Settings
- Data Sharing
- Updates
- Notifications

Keymap

Editor

- Inspections
- Live Templates
- Intentions
- Natural Languages
- Grammar and Style
- Plugins

Build, Execution, Deployment

- Build Tools
- Maven
- Importing
- Required Plugins

Languages & Frameworks

Tools

- External Tools
- Settings Sync

Search Results (17) Sort By: Relevance

AWS Toolkit

Amazon Web Services Plugin homepage ↗

Install 1.67-231

Big Data Tools Ultimate Install

Merloc - AWS Lambda Debugg... Install

AWS CodeArtifact + Maven Install

Elasticsearch Query - EDQL Install

DynamoDB Paid Install

Elasticsearch Paid Install

Cicclone Paid Install

Httpx Requests Ultimate Install

Vacuum Ultimate Install

Cloud

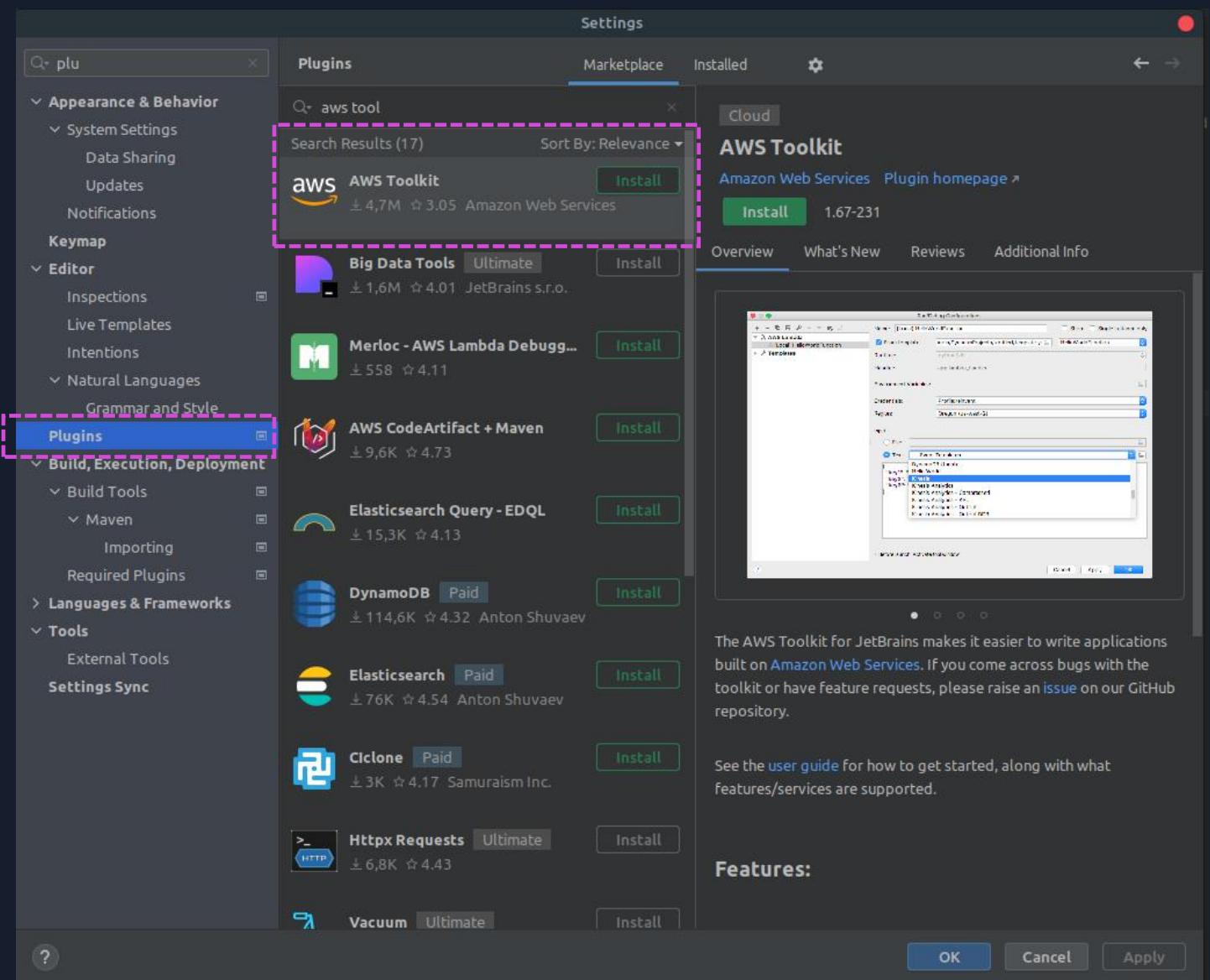
Overview What's New Reviews Additional Info

The AWS Toolkit for JetBrains makes it easier to write applications built on [Amazon Web Services](#). If you come across bugs with the toolkit or have feature requests, please raise an [issue](#) on our GitHub repository.

See the [user guide](#) for how to get started, along with what features/services are supported.

Features:

OK Cancel Apply



# What is Toolkits IDE 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 AWS Toolkit for IntelliJ IDEA interface. On the left, the Explorer sidebar lists various AWS services: App Runner, CloudFormation, CloudWatch Logs, DynamoDB, ECR, ECS, Lambda, S3, Schemas, SQS, and Resources. The S3 section is expanded, showing the 'campus-party-2023' bucket. On the right, the main window displays the contents of the 'campus-party-2023' bucket, listing three objects: AB177.jpg, AB188.jpg, and AB199.jpg. A dashed purple box highlights the S3 section in the Explorer and the bucket contents in the main pane. A large arrow points from the highlighted S3 section towards the main pane, indicating the integration between the two views.



# Introduction AWS Serverless



## What is “Serverless” ?

---

**It's not about having servers or  
the lack of servers**



<https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/SVS401-ri22.pdf>

# What is “Serverless” ?

## Consider “serverless” as:

A **mindset**, an approach, a **practice**, a **culture**, a way of working Focus on **business value**, rather than the enabling technology

## Benefits:

**Faster** time to market from **prototype** to **production**

**Rapid**, continuous experimentation and feedback

React and **deliver** business changes with a product **mindset**

**Serverless = the best way to build and run modern applications**

## “Serverless” architecture is an **operational model**

Minimize taking on ongoing operational tasks, outsource system administration tasks

Every non-serverless component = ongoing ops responsibility

Optimize for long-term maintainability

Serverless does not mean "No Ops"!

## Concentrate on the flow of data and events



---

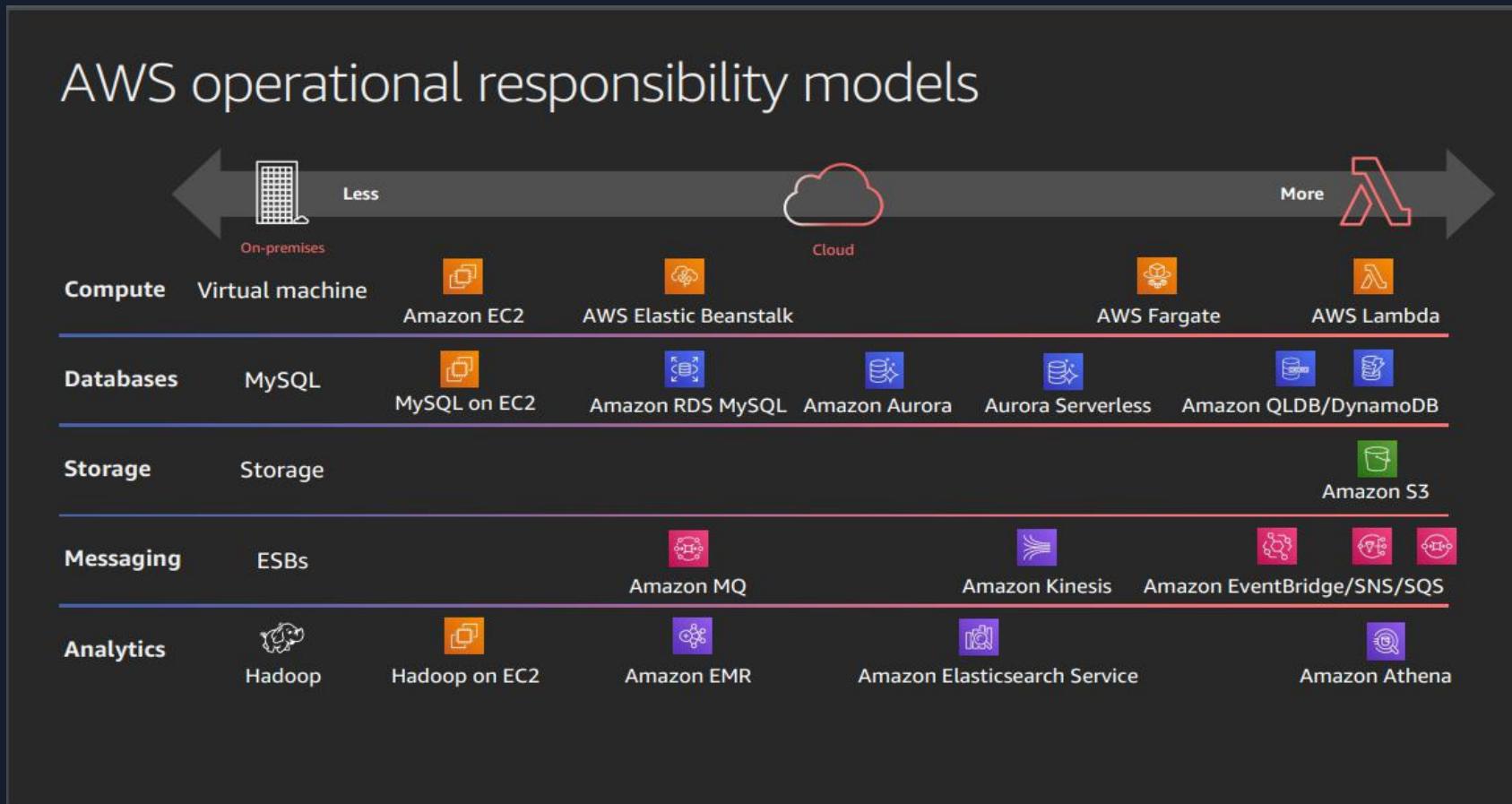
# **Events are the language of serverless applications.**

Dave Boyne  
AWS Serverless Developer Advocate

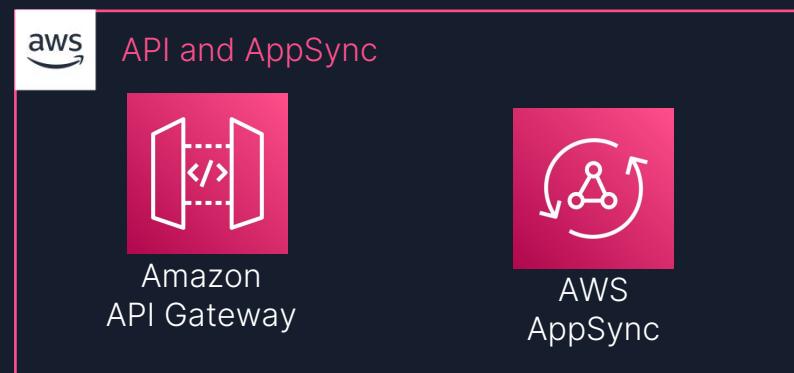
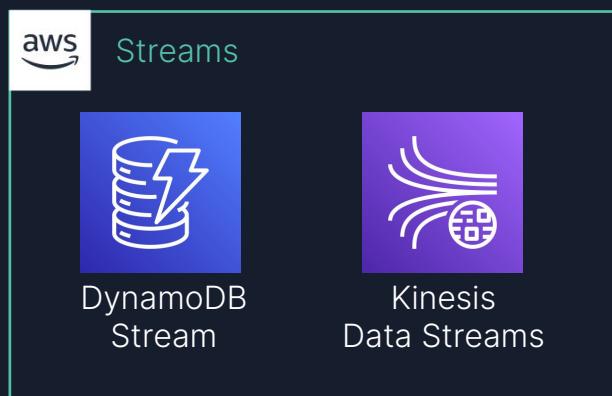
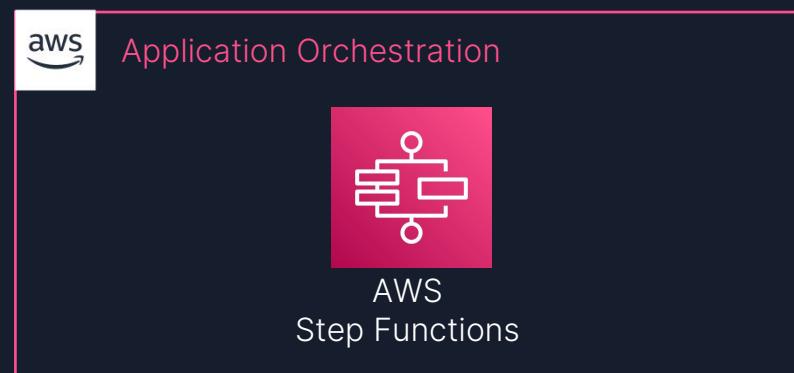
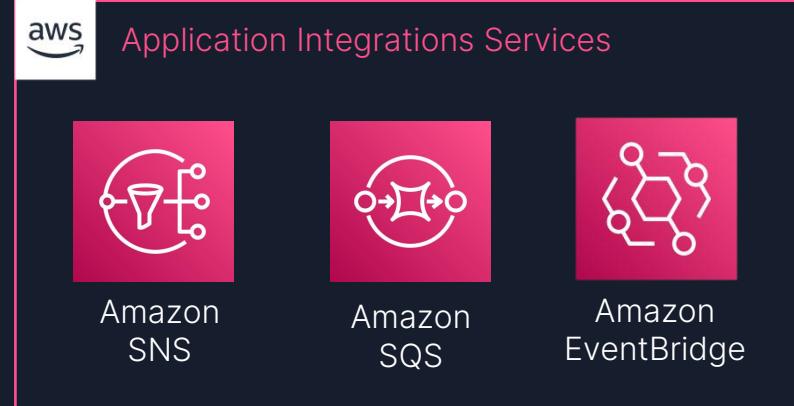
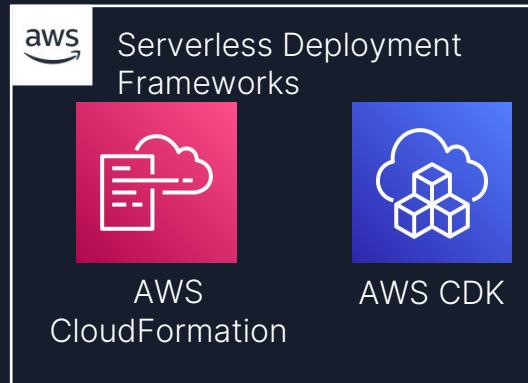


<https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/SVS401-ri22.pdf>

# Serverless Explained : AWS Operational Responsibility Model



# 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



# 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
```



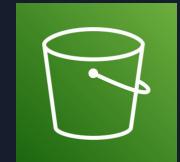
## Copy files to S3 bucket

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



## List S3 buckets

```
aws s3 ls
```



Amazon S3

Figure: 1 - describe using CLI Commands handling Objects in S3 service.

[https://github.com/weder96/joinCommunity2023/blob/main/learning/09\\_S3/README.md](https://github.com/weder96/joinCommunity2023/blob/main/learning/09_S3/README.md)



DynamoDB



Aurora



Neptune

# AWS Database



# 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**

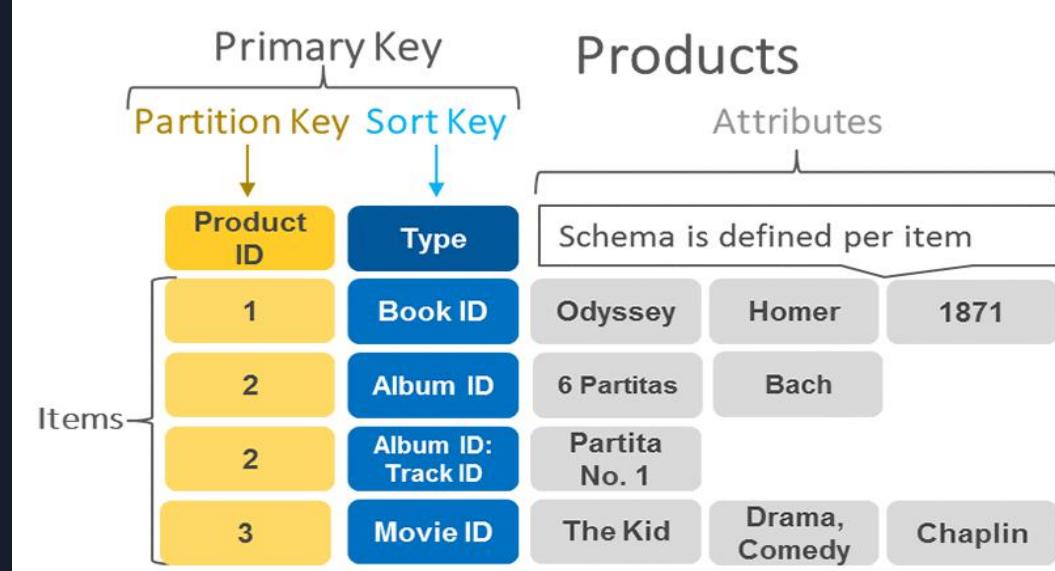


Figure: 1 - Partition Key and Sort Key (Primary key)

General information	
Partition key orderId (String)	Sort key orderDate (String)
Alarms <span style="color: green;"> ⓘ No active alarms</span>	Point-in-time recovery (PITR) <span style="color: blue;"> ⓘ</span> <span style="color: grey;"> ⓘ Off</span>
▶ Additional info	

Figure: 2 - Details table (Partition Key, Sort Key)

# DynamoDB Python(Boto3)

Files

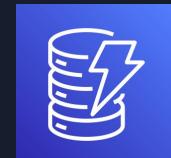
weder96 learning serverless with cli commands

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
```

Octotree

- learning
- 01\_start
- 02\_calc
- 03\_logStream
- 04\_Name
- 05\_Greeting
- 06\_function\_with\_url
- 07\_product
- 08\_dynamodb
  - crud
    - creatitem.py
    - deleteitem.py
    - getAllItems.py
    - getOneItem.py
    - updateItem.py
  - README.md
- 09\_S3



DynamoDB

Figure: 1 - Function Lambda (using Python 3.10), CRUD DynamoDB

[https://github.com/weder96/joinCommunity2023/tree/main/learning/08\\_dynamodb](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

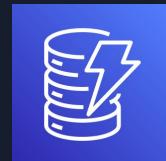
### CLI

**Create table DynamoBD**

```
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 DynamoBD**

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

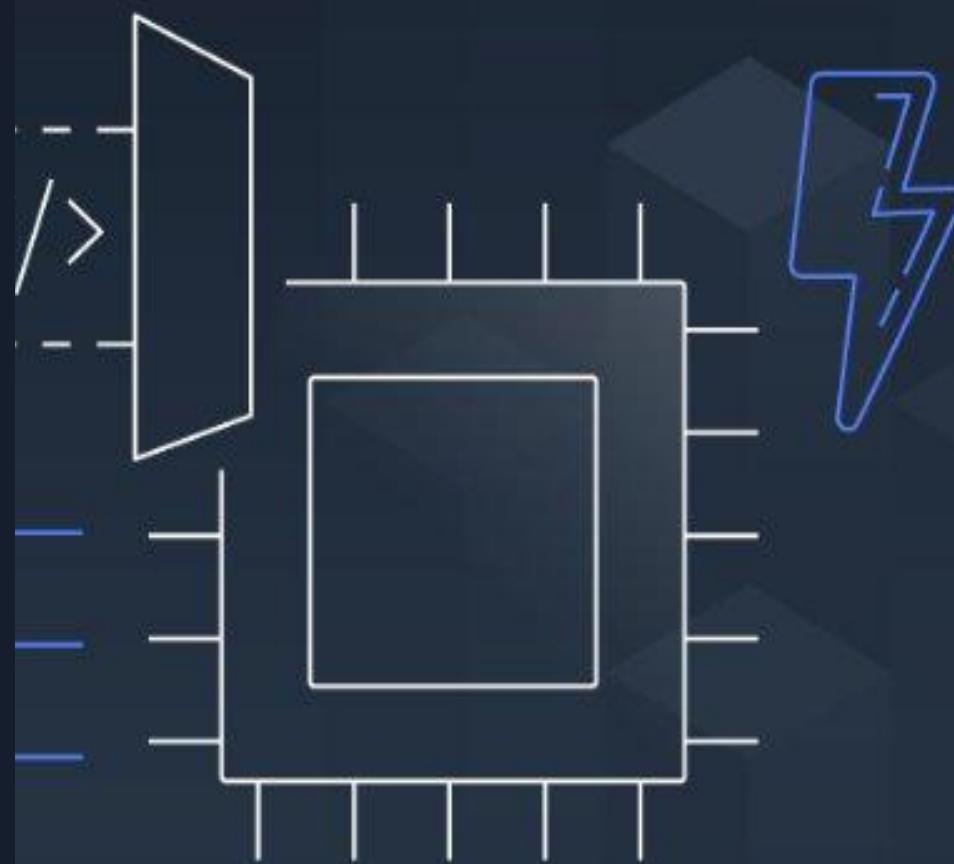


DynamoDB

Figure: 1 - create table AWS CLI Commands



# AWS 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 CLI Commands

joinCommunity2023 / learning / 10\_ApiGateway /

**Create a role**

We use the [create-role](#) command :

**create the lambda role**

```
aws iam create-role --role-name apigw-lambda-role-py --assume-role-policy-document 'file://lambda-role-policy.json' -
```

Or

```
aws iam create-role --role-name apigw-lambda-role-py --assume-role-policy-document '{"Version": "2012-10-17", "Statement": {}}
```

**Attach the execution policy to it**

We use the [attach-role-policy](#) command :

**attach the AWSLambdaBasicExecutionRole policy to the lambda role**

**attach-role AWSLambdaBasicExecutionRole managed policy:**

```
aws iam attach-role-policy --role-name apigw-lambda-role-py --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

**Zip file Linux**

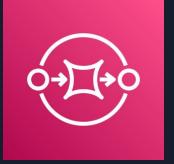


Amazon API Gateway

Figure: 1 - create API Gateway AWS CLI Commands (Role, Lambda, Resources, Permissions, putMethodResponse)



SNS



SQS



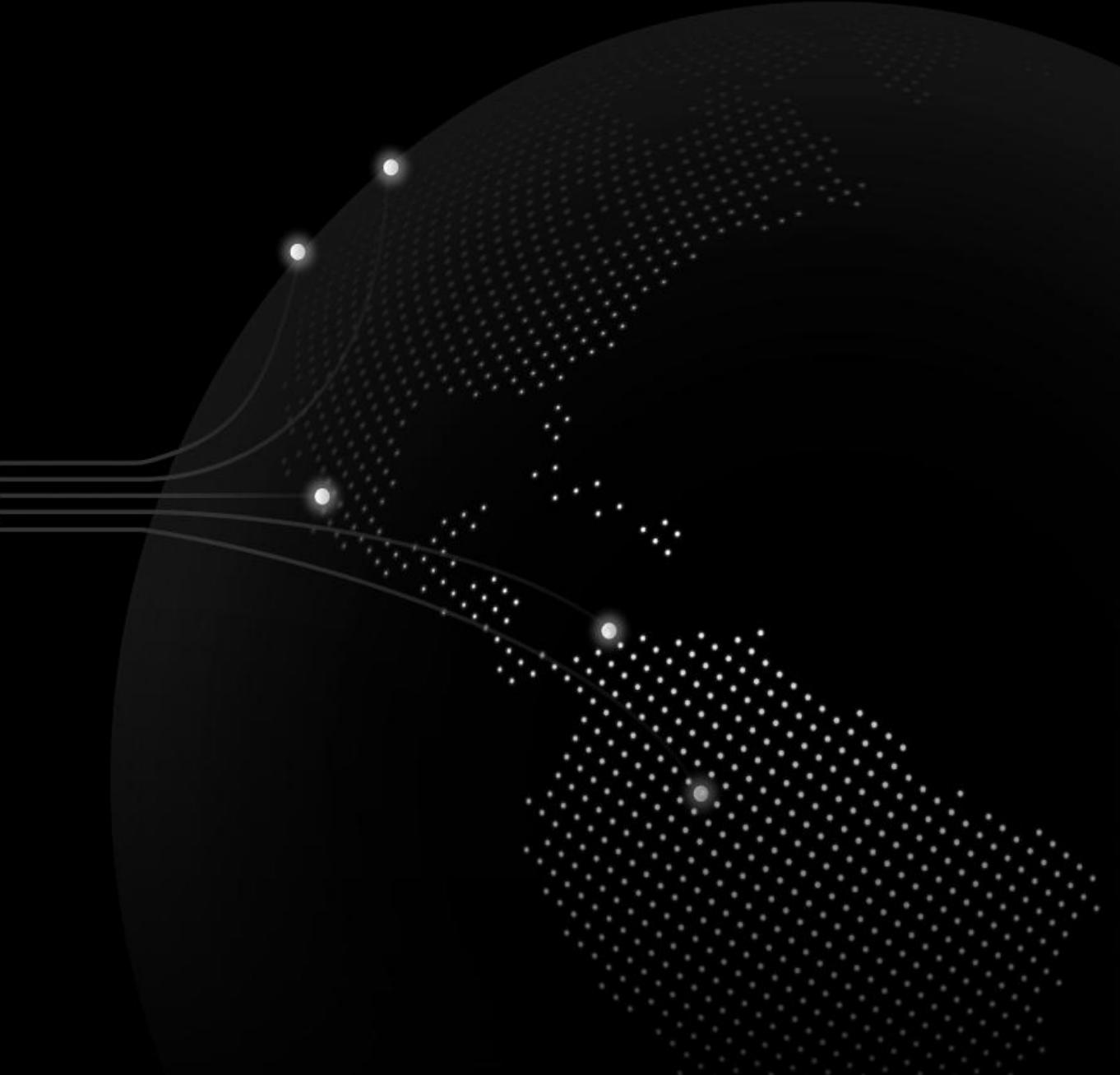
EVENTBRIDGE

# 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

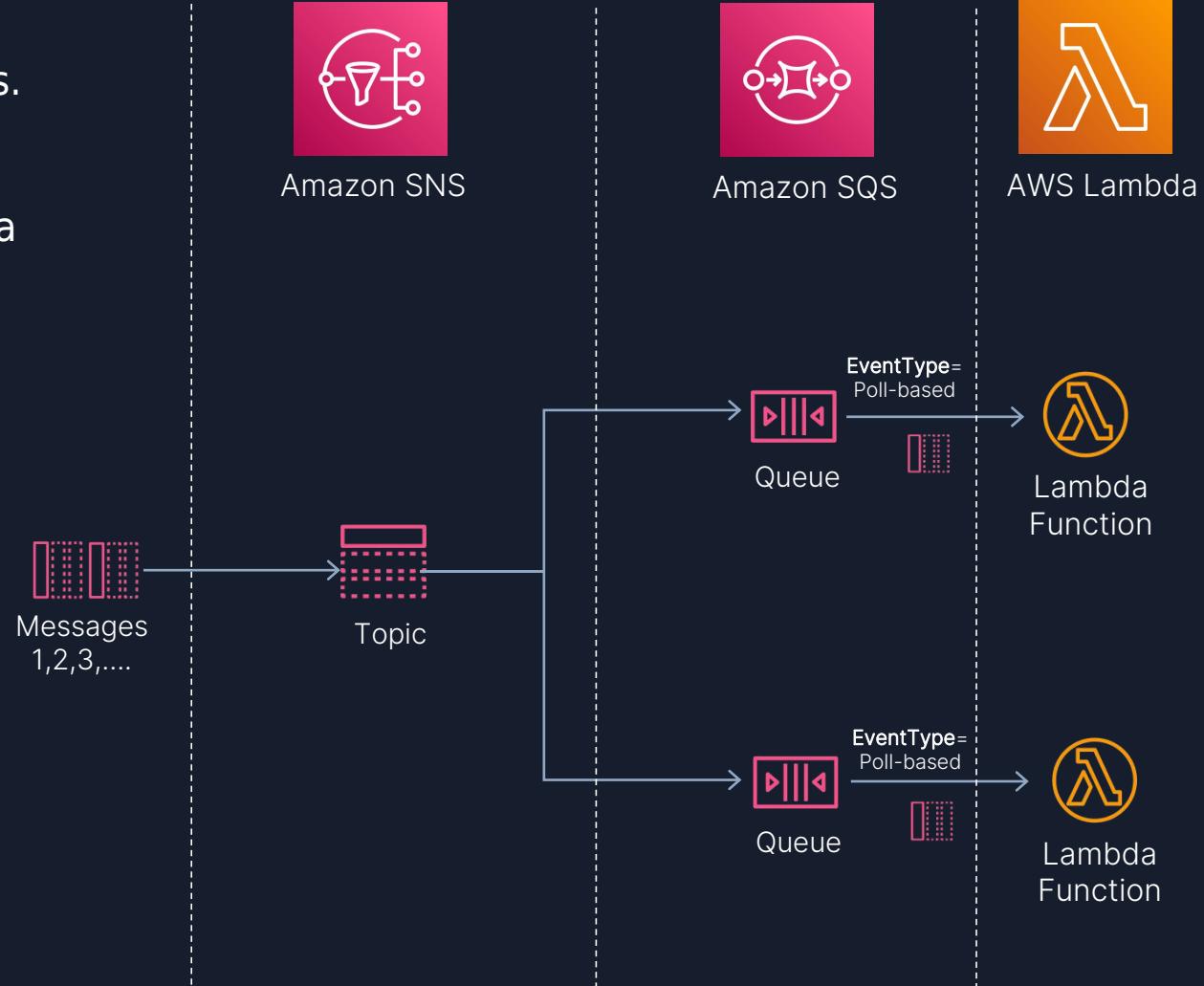
Amazon SNS can send notifications to specified users via SMS and email.

## 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 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

Figure: 1 - create SNS AWS CLI Commands (Lambda, topic , subscribe , permission)

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



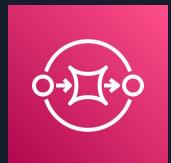
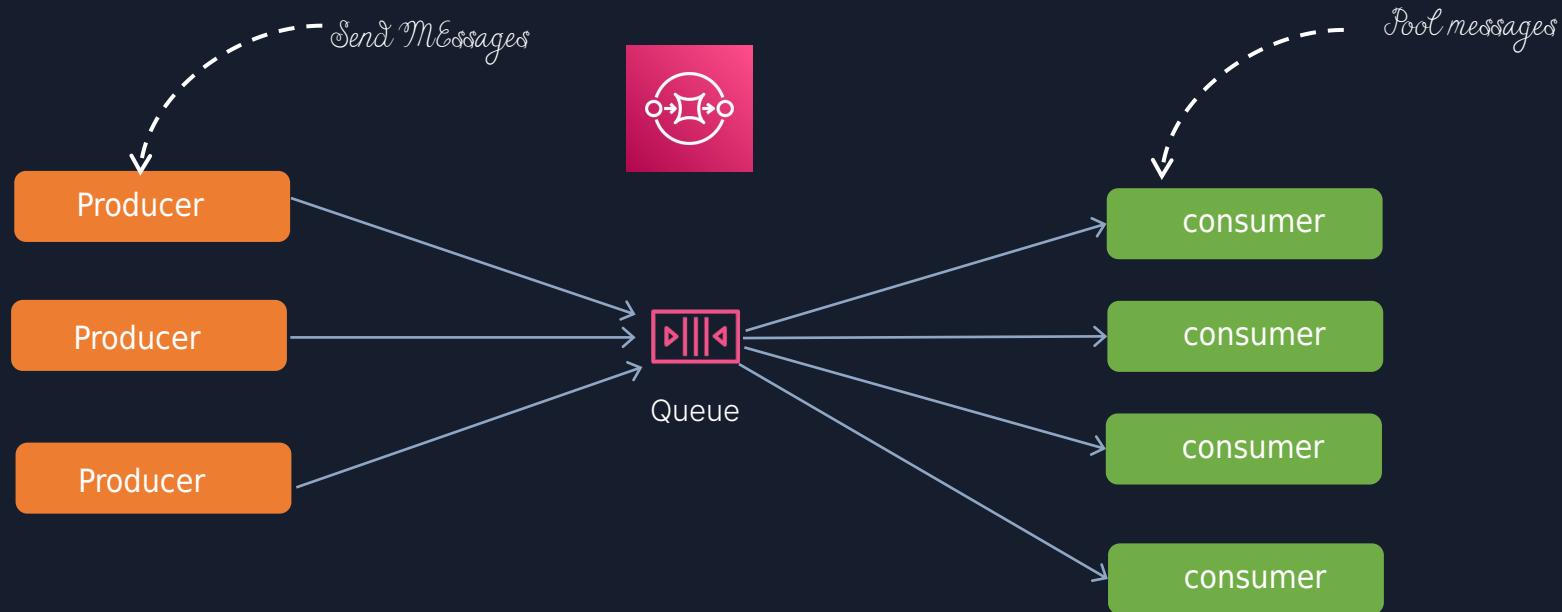
Amazon SQS

[https://aws.amazon.com/sns/?nc1=h\\_ls](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](https://aws.amazon.com/sns/?nc1=h_ls)

# 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

Figure: 1 - create SQS AWS CLI Commands (create, poll , send messages , permission, purge)

[https://github.com/weder96/joinCommunity2023/blob/main/learning/16\\_lambda\\_sqs/README.md](https://github.com/weder96/joinCommunity2023/blob/main/learning/16_lambda_sqs/README.md)



# Amazon EventBridge



# What is Amazon EventBridge ?

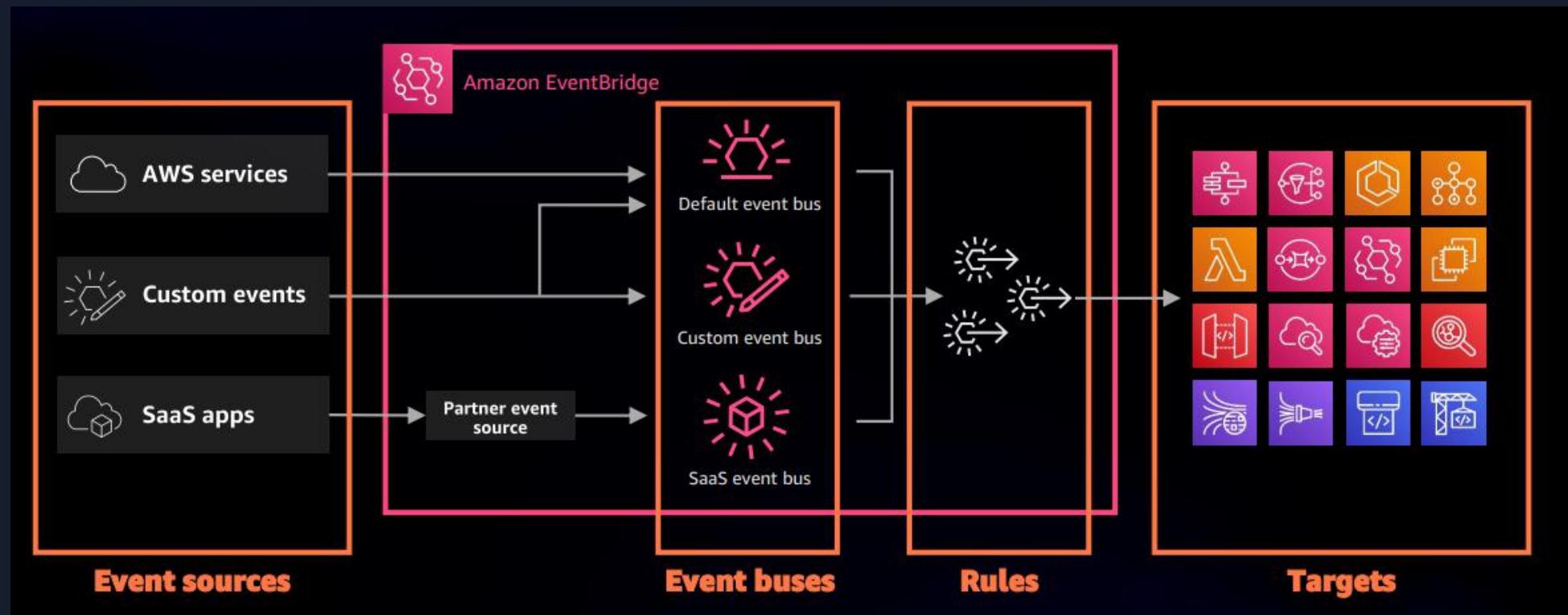
Serverless **event bus** service for AWS services

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**



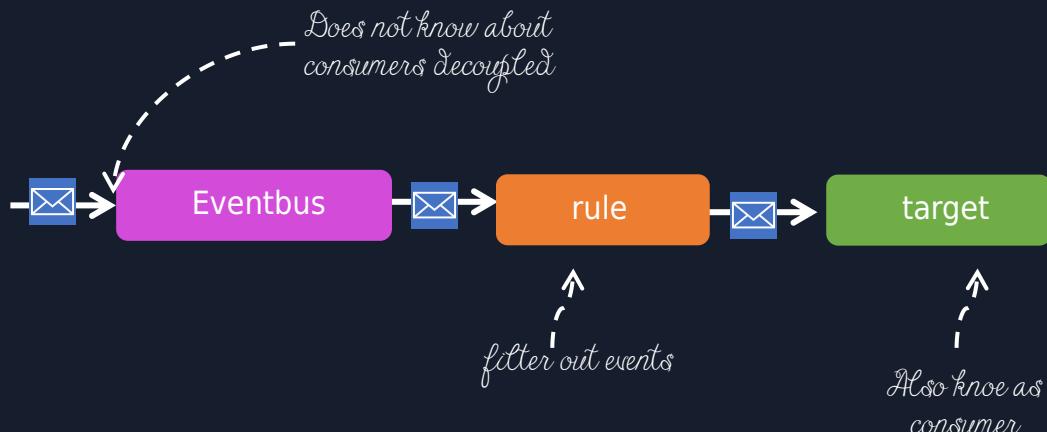
# What are events?

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"  
        }  
    }  
}
```

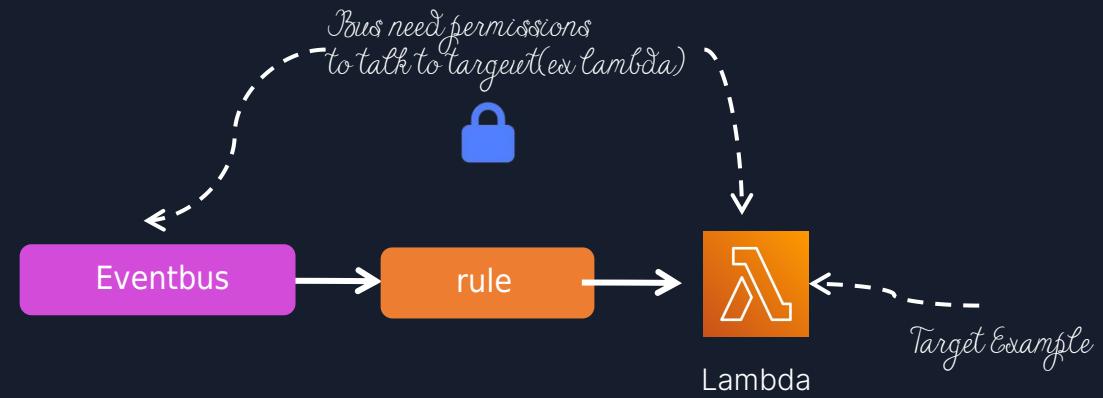
Figure: 1 - JSON event details is Custom(You define)

# 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

The screenshot shows a GitHub repository interface. On the left, the file tree displays various projects like S3, API Gateway, WebSocket, Lambda, and EventBridge. The README.md file is selected and shown on the right.

**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

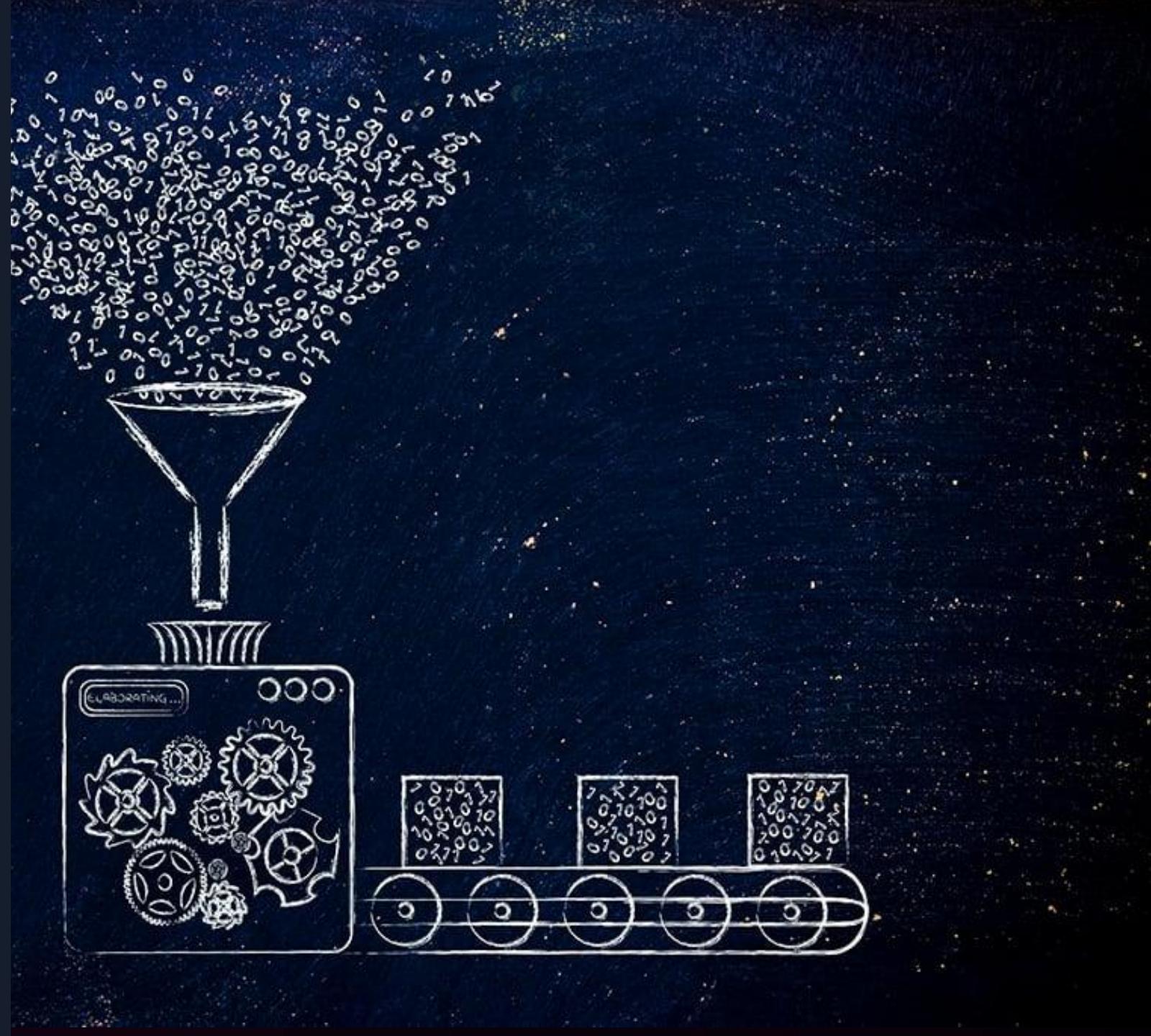
Figure: 1 - create EventBridge AWS CLI Commands (create, putTarget, putRule , permission, putEvents)

[https://github.com/weder96/joinCommunity2023/blob/main/learning/19\\_eventbridge/README.md](https://github.com/weder96/joinCommunity2023/blob/main/learning/19_eventbridge/README.md)



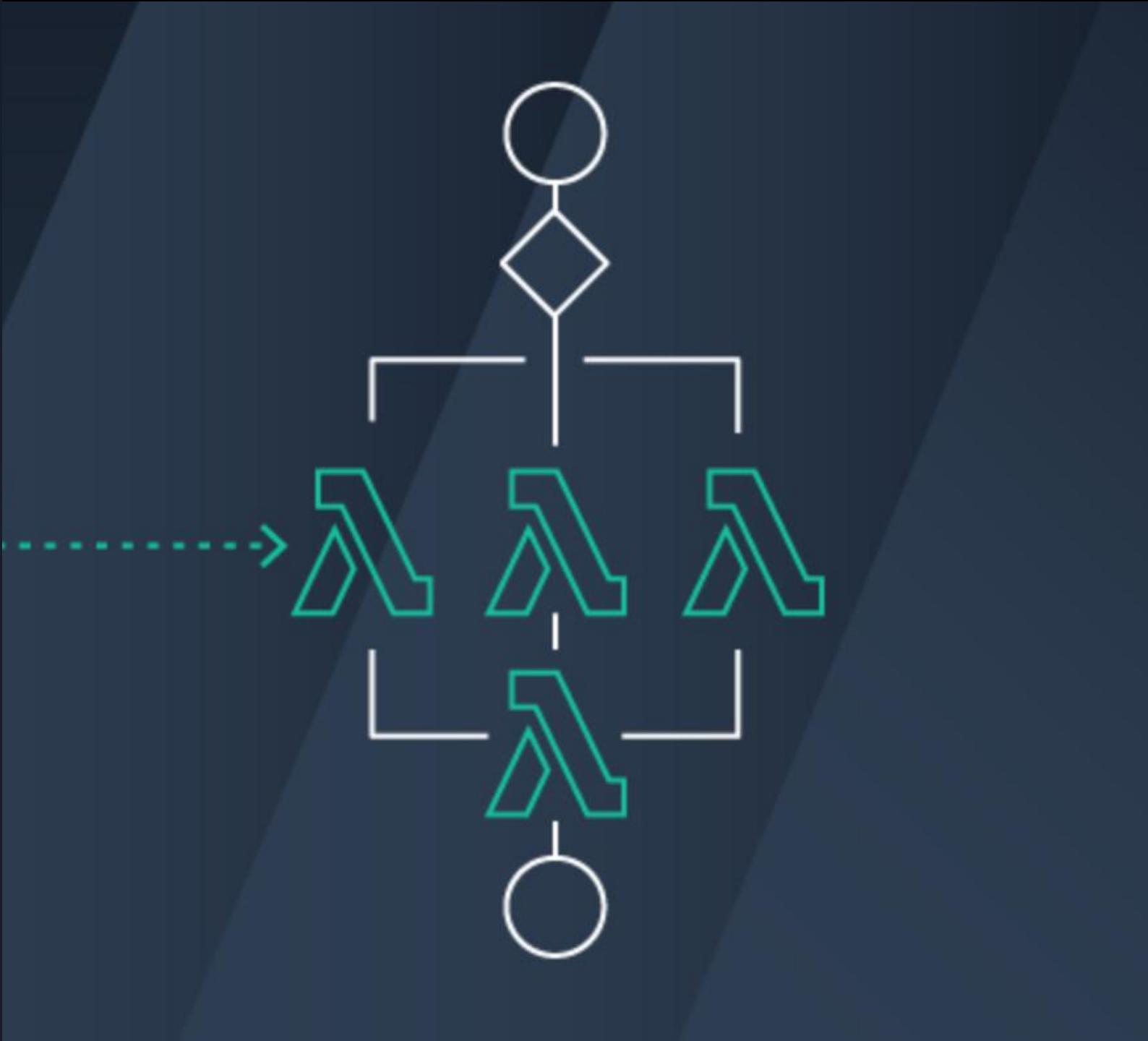
STEP FUNCTIONS

# AWS Application Orchestration





# AWS Step Functions



# 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.



# Create a Serverless Workflow with AWS Step Functions and AWS Lambda

Step Functions > State machines > CallCenterStateMachine > Edit

## Edit CallCenterStateMachine

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",
```

Cancel **Start execution** Save Export Layout

The screenshot shows the AWS Step Functions Studio interface. On the left, there is a code editor with the state machine definition in Amazon States Language (ASL). On the right, there is a visual representation of the state machine. The visual representation consists of several states connected by arrows: Start (yellow circle) leads to Open Case (rectangle), which leads to Assign Case (rectangle). Assign Case leads to Work on Case (rectangle). Work on Case leads to Is Case Resolved (rectangle). From Is Case Resolved, two paths emerge: one leading to Escalate Case (rectangle) and another leading to Close Case (rectangle). Escalate Case leads to Fail (rectangle). Both Close Case and Fail lead to End (yellow circle). There are also 'Workflow Studio' and 'New' buttons at the top right of the visual representation area.



Figure: 1 - StepFunctions Studio(create, view, execute the state Machine )

<https://aws.amazon.com/pt/tutorials/create-a-serverless-workflow-step-functions-lambda/>

# Amazon Step Functions CLI Commands and (Python) Boto3

The screenshot shows a GitHub repository interface for a file named README.md. The repository path is `joinCommunity2023 / learning / 21_step_function / README.md`. The file was last updated by `weder96` and has 123 lines of code. A note indicates that the code is 55% faster with GitHub Copilot.

The README.md file contains the following content:

## 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
```

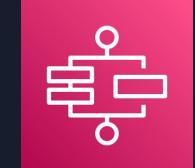
#### Execute state machines

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

A purple dashed box highlights the directory structure on the left, which includes:

- 11\_WebSocket
- 12\_APIGateway\_lambda\_dyna...
- 15\_lambda sns
- 16\_lambda\_sqs
- 17\_dynamodb\_stream
- 18\_kinesis\_stream
- 19\_eventbridge
- 20\_event\_driven
- 21\_step\_function (selected)
- stepFunctionHelloWord
  - app.py
  - input.json
  - stateMachineDefinition.json
- stepFunctionsWithLambda
  - README.md
- 22\_cdk\_infra
- 23\_iam\_sdk\_python
- pdfs
- ~presentation\_join\_community...

Figure: 1 - StepFunctions CLI Commands(create, update, execute the state Machine) and create lambda



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

# 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



# Lambda Function Code

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

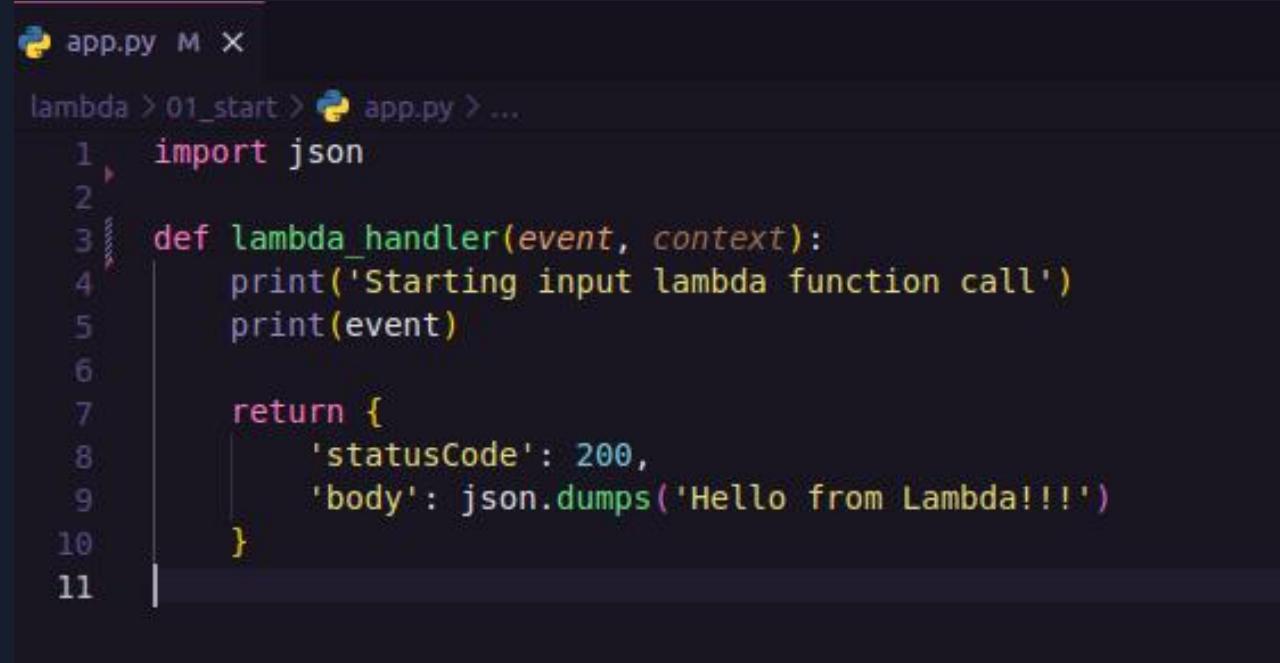
Invoke function directly **using the Lambda API**, or configure an AWS service or resource to invoke your function.

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
```



# 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"  
    }  
  ]  
}
```

Figure: 1 - Policy(JSON), set Permissions for execute lambda or lambda invoke AWS Services

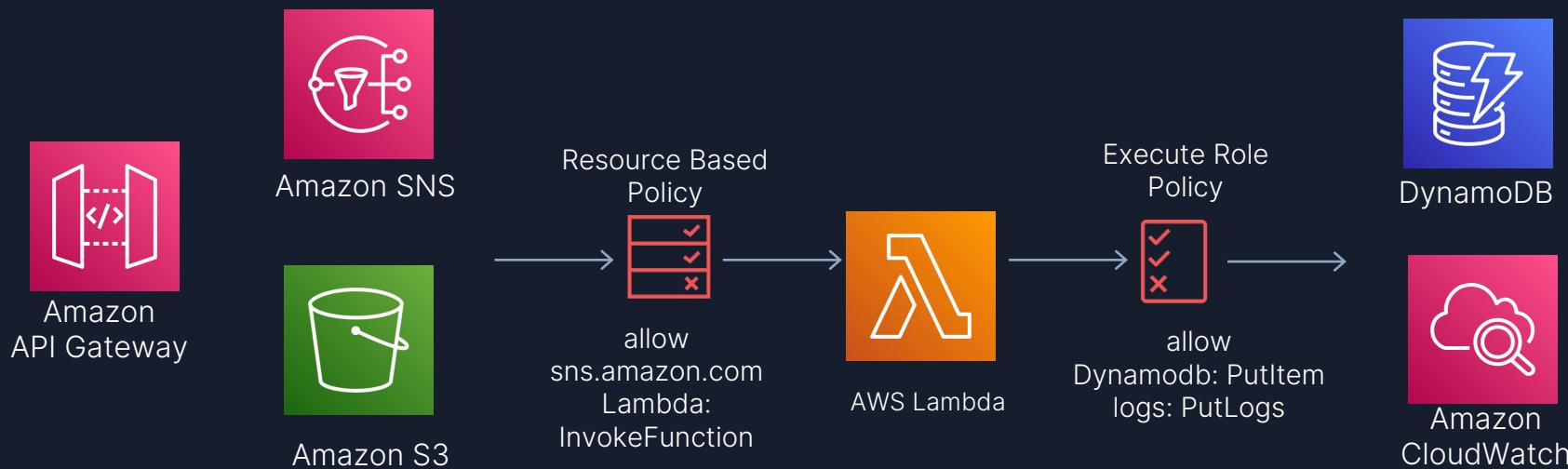
# AWS Lambda Resource-based Policy

## 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.

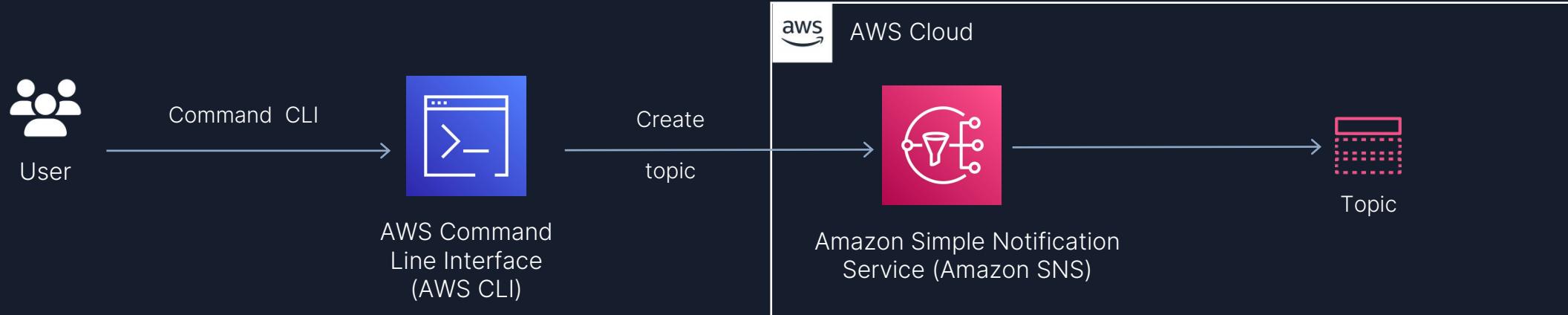




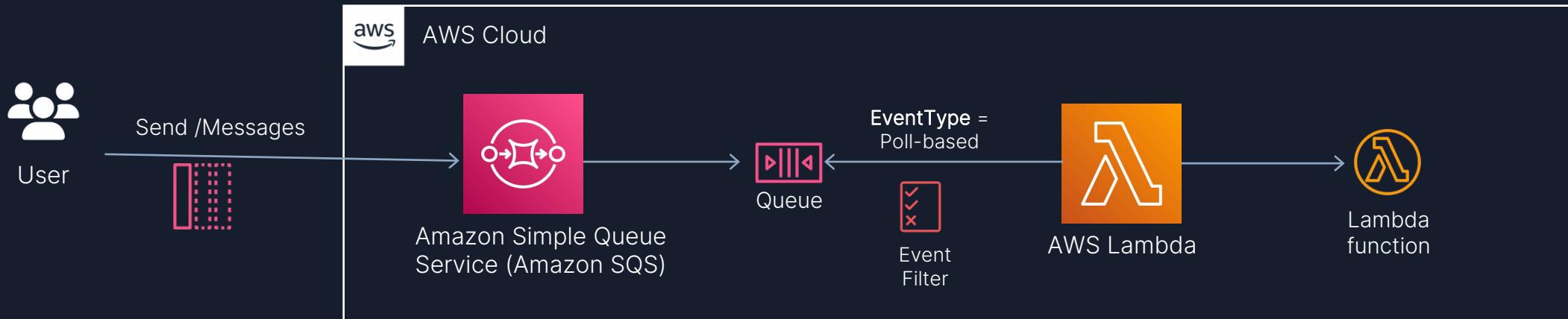
# Architecture Serverless



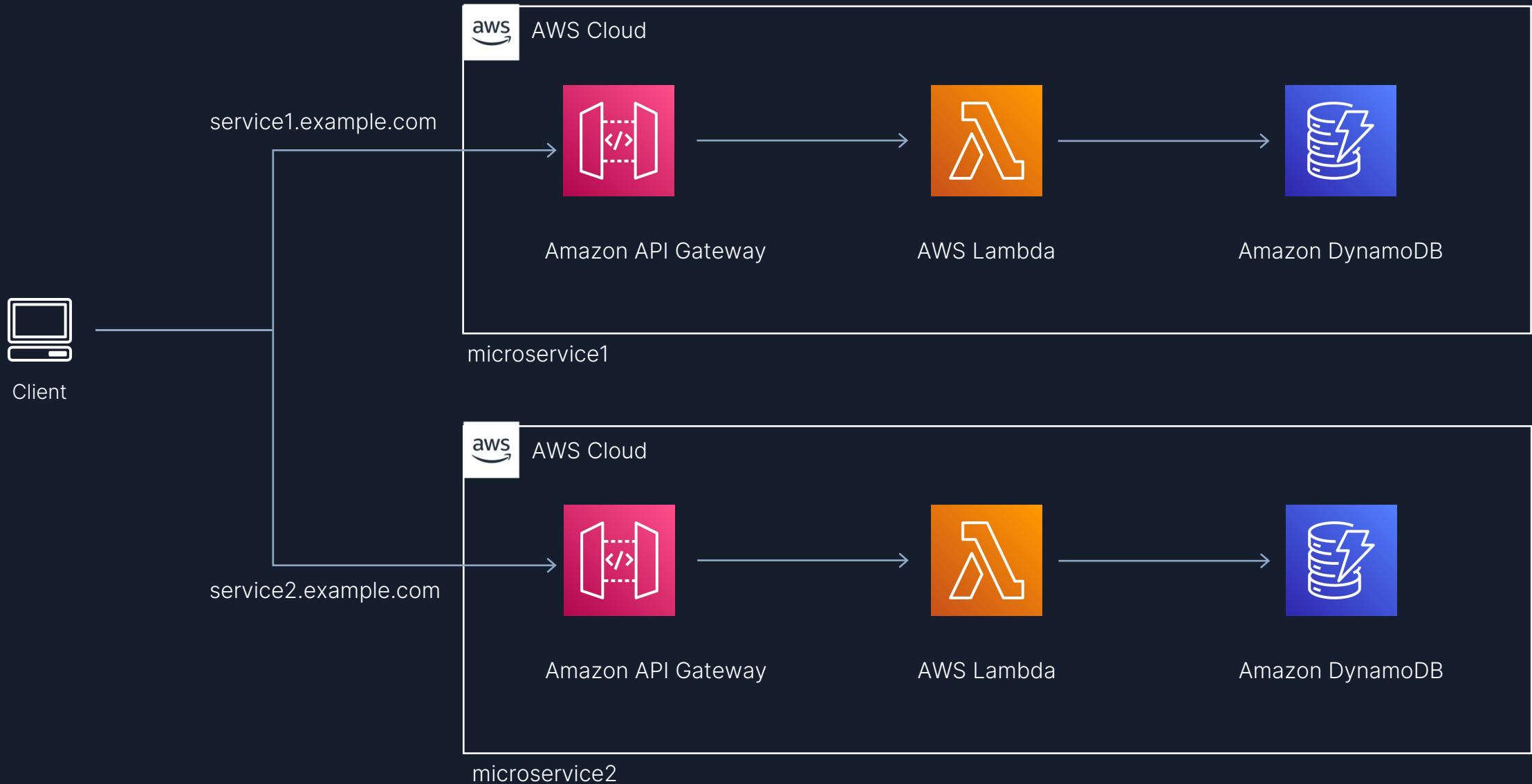
# Amazon SNS with AWS Management Console



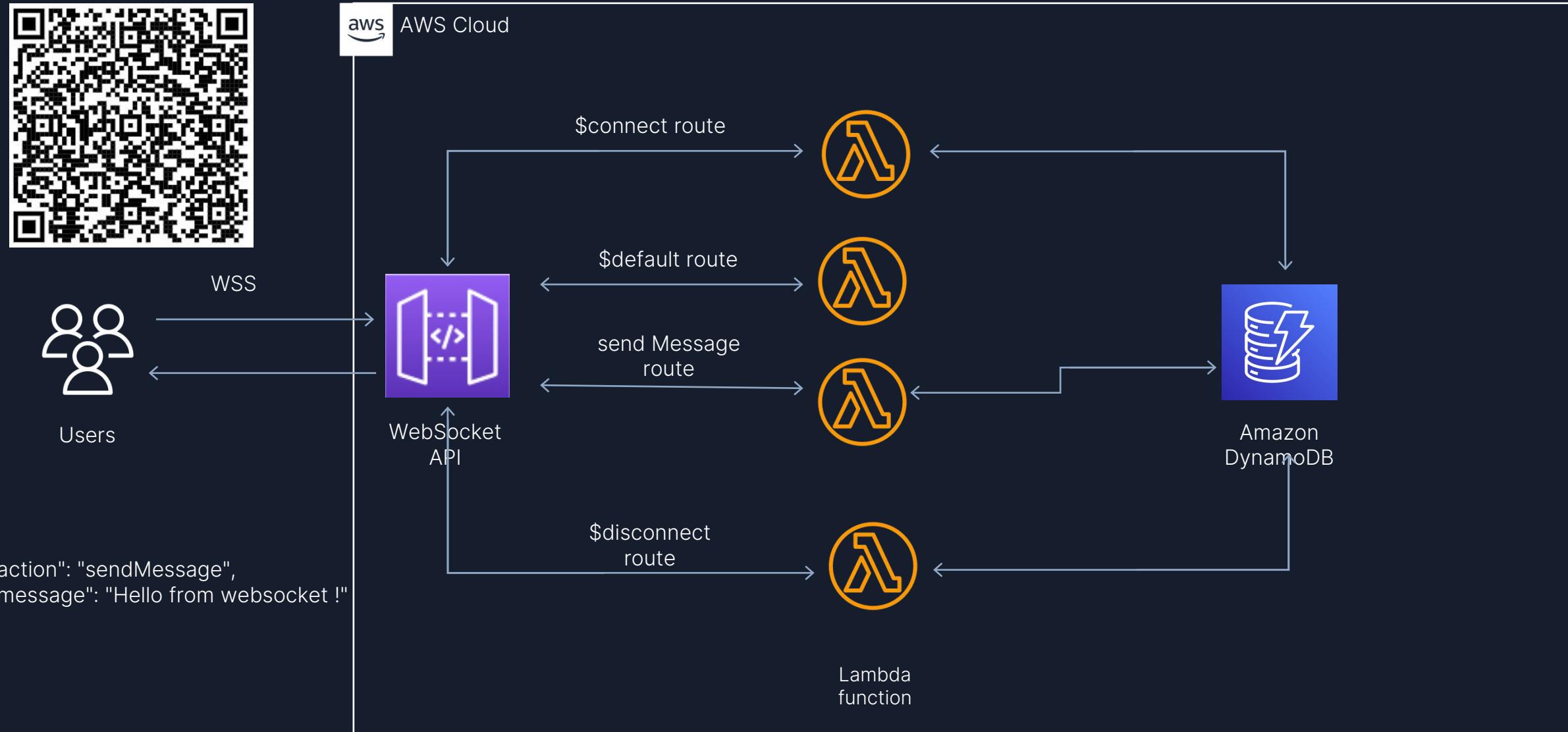
# Amazon SQS Queue Polling From AWS Lambda



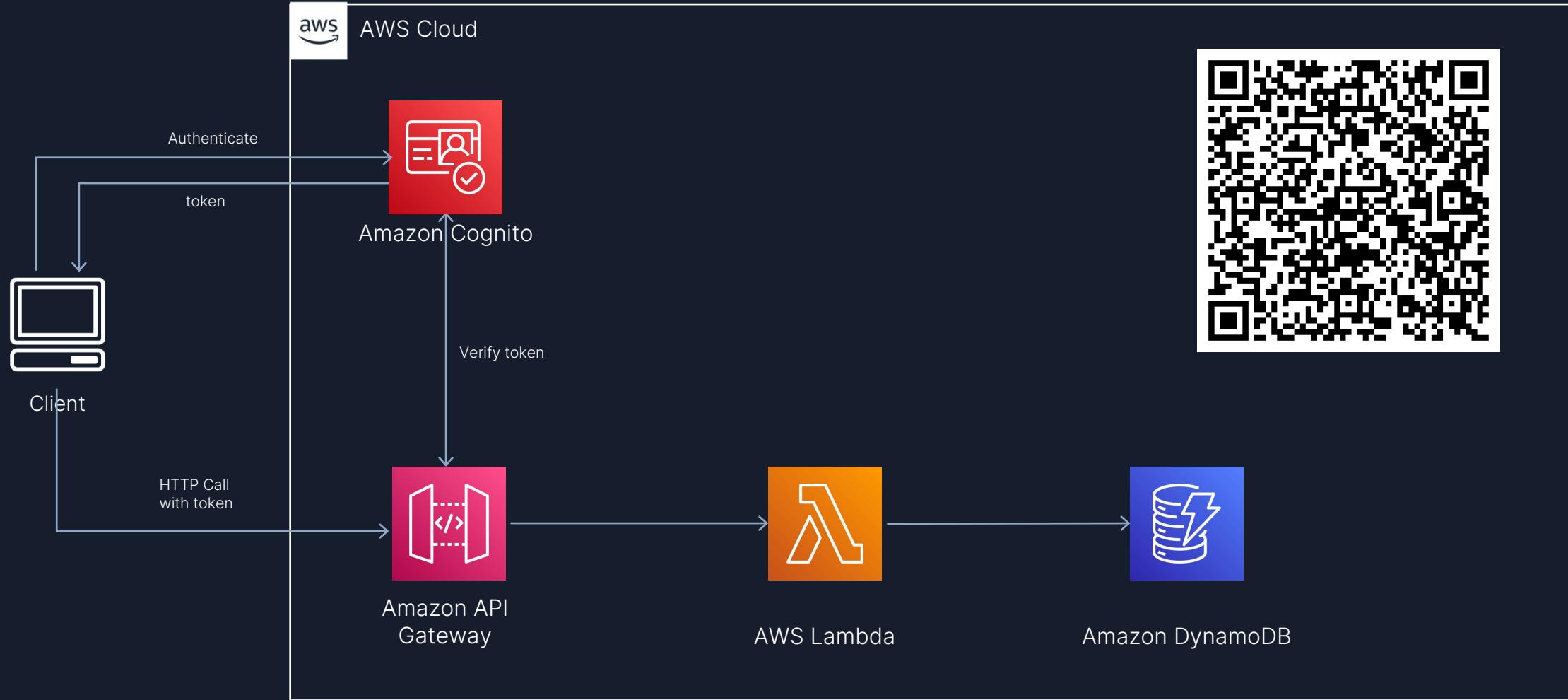
# Microservices “REST”



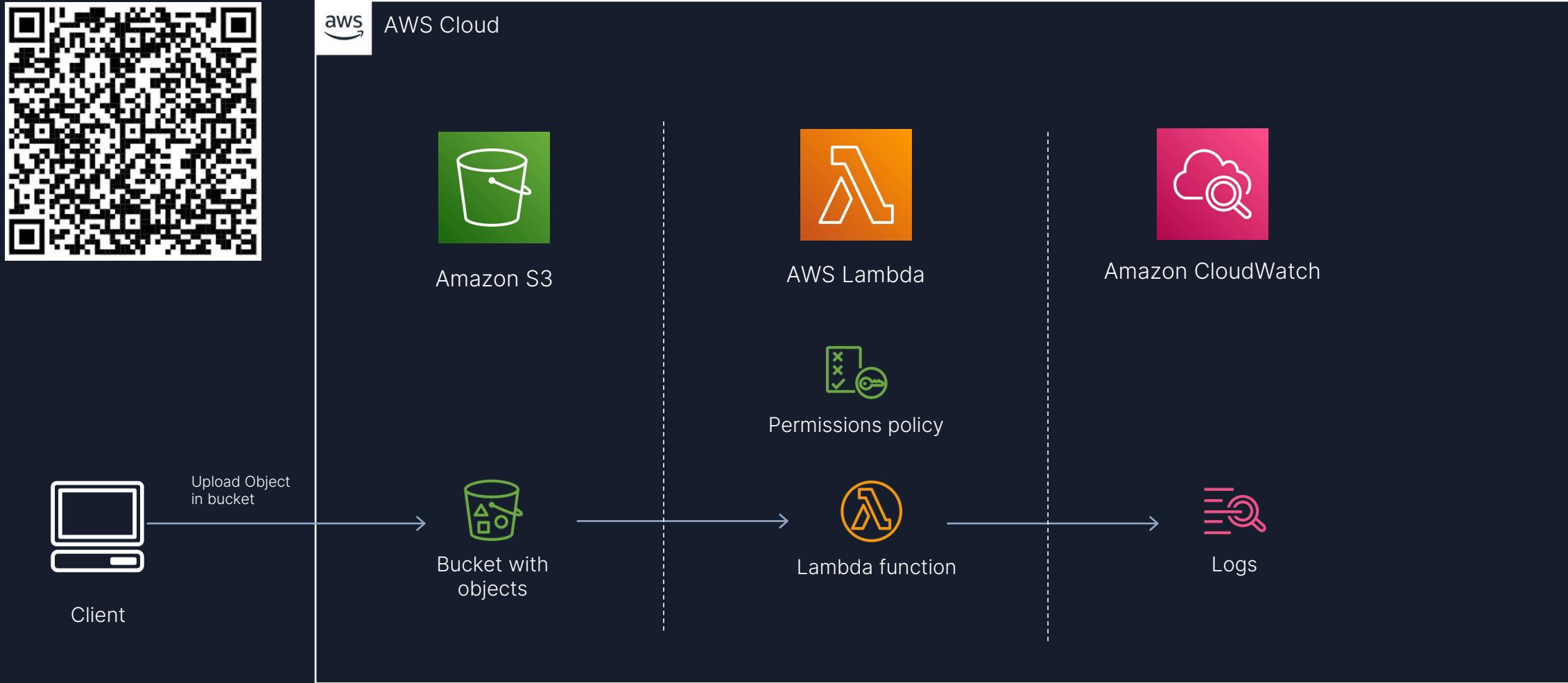
# Build Serverless Chat App with a WebSocket API and Lambda



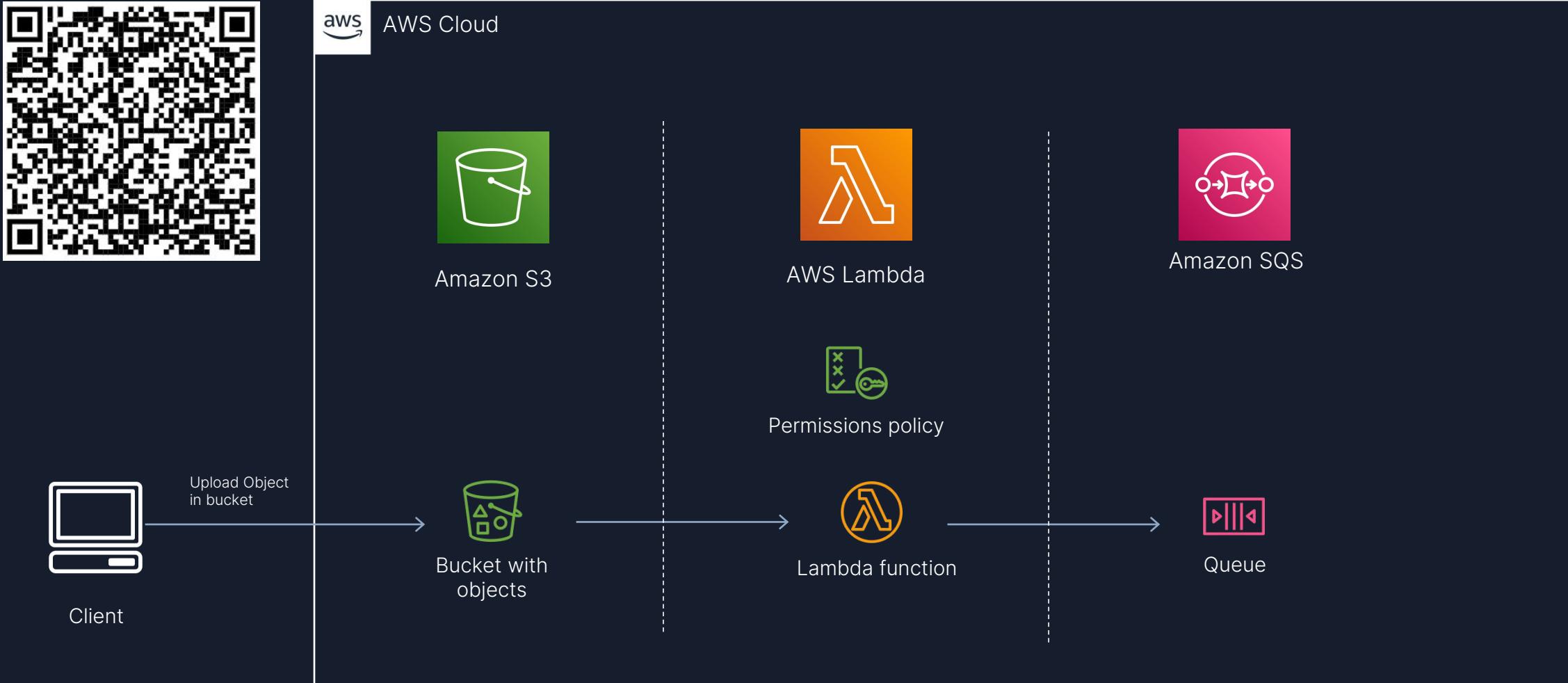
# Secure your API Gateway with Amazon Cognito User Pools



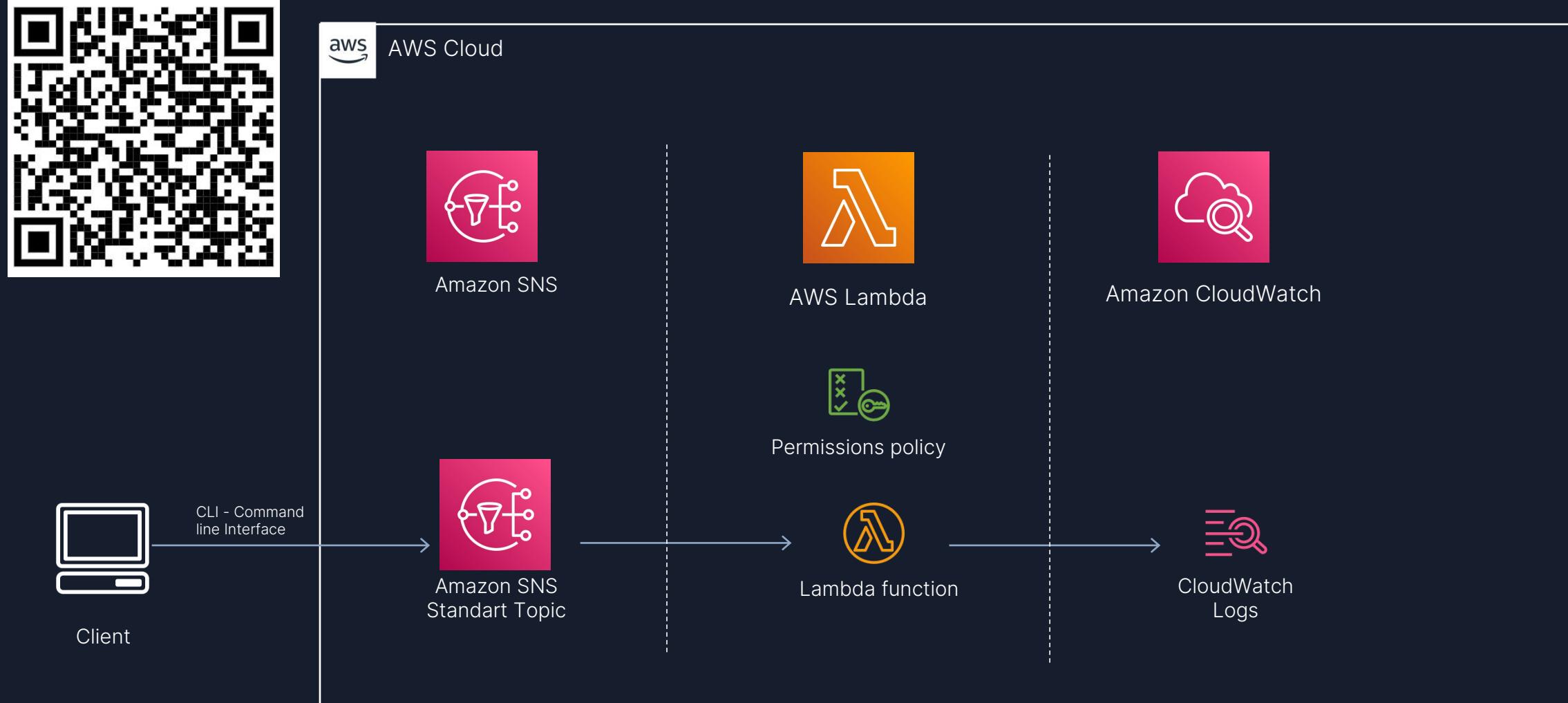
# Using an Amazon S3 trigger to invoke a Lambda function



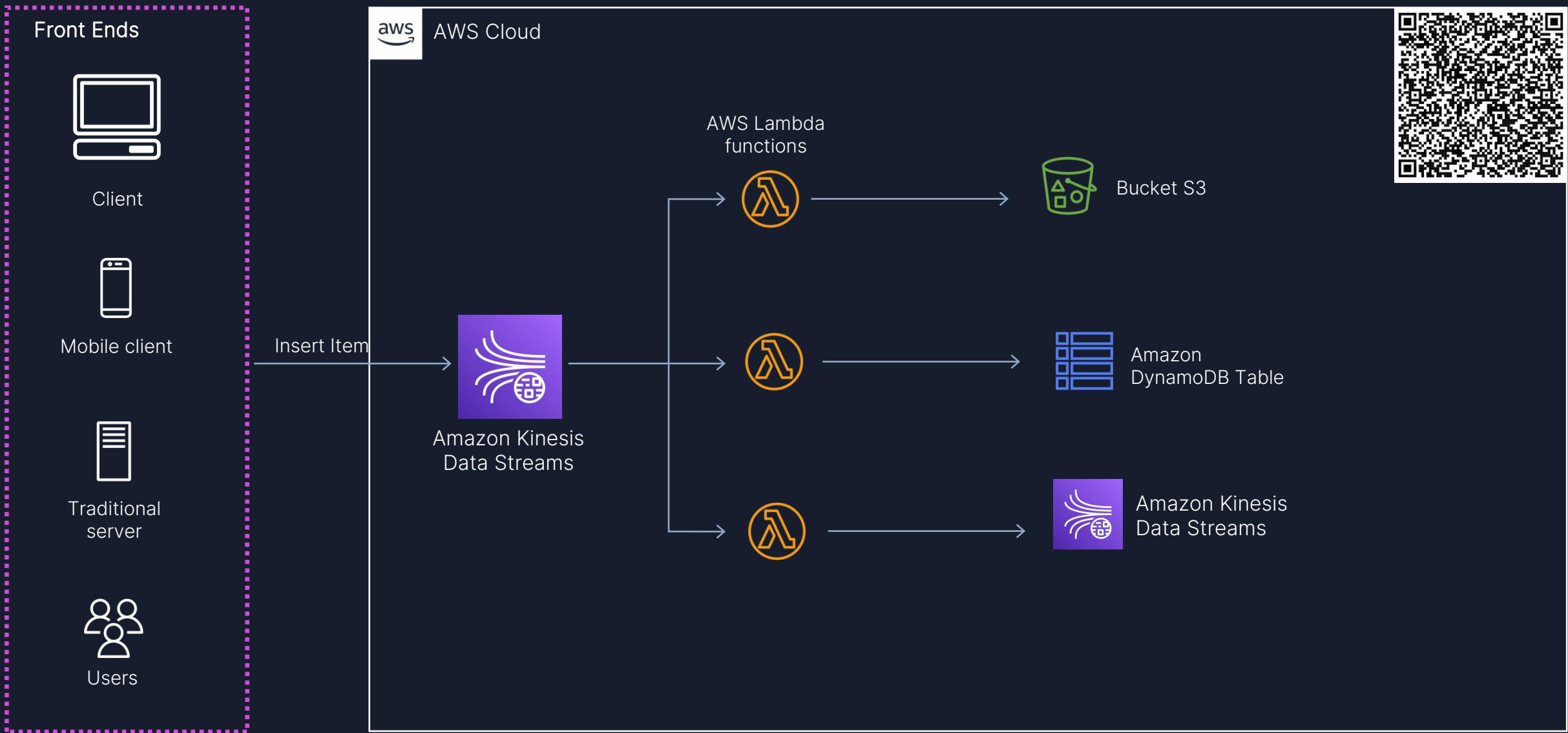
# AWS Lambda Destination to SQS - DLQ Case



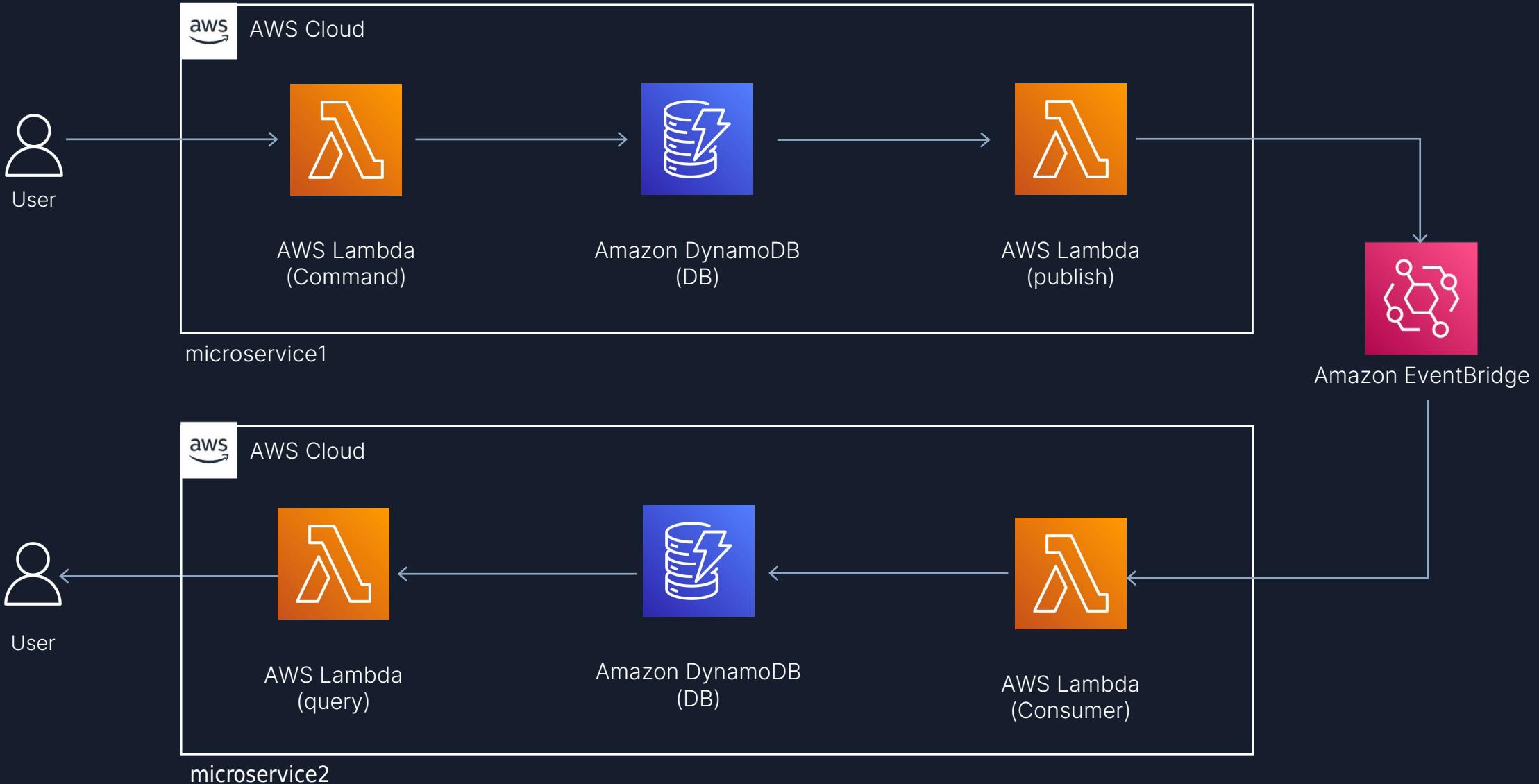
# Using an Amazon SNS to send message to Lambda function



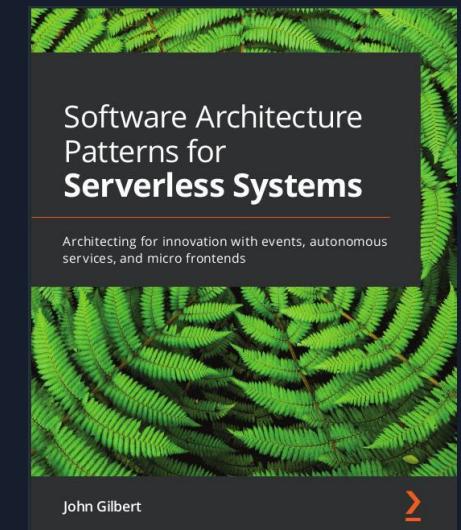
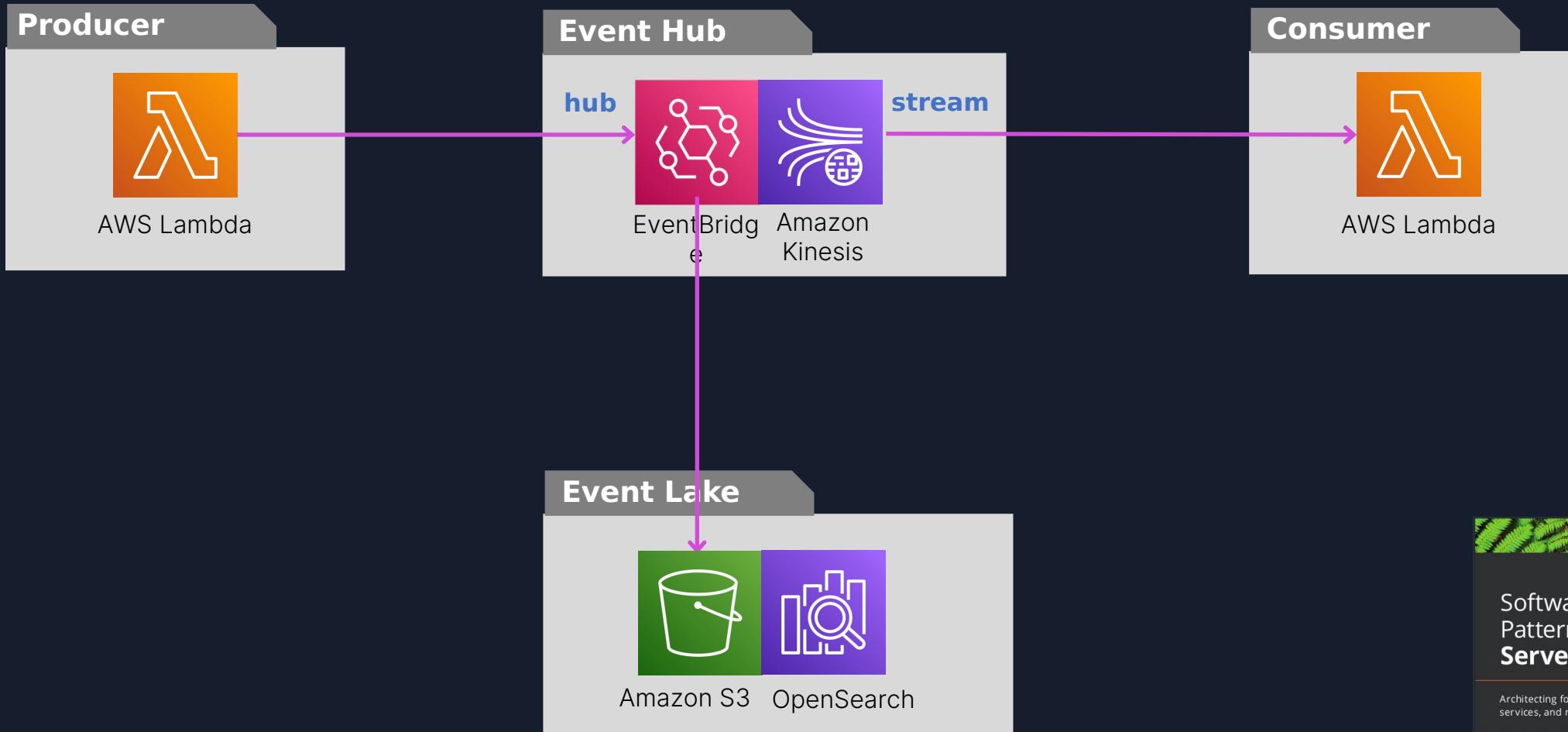
# Processing performance with Amazon Kinesis Data Streams and AWS Lambda



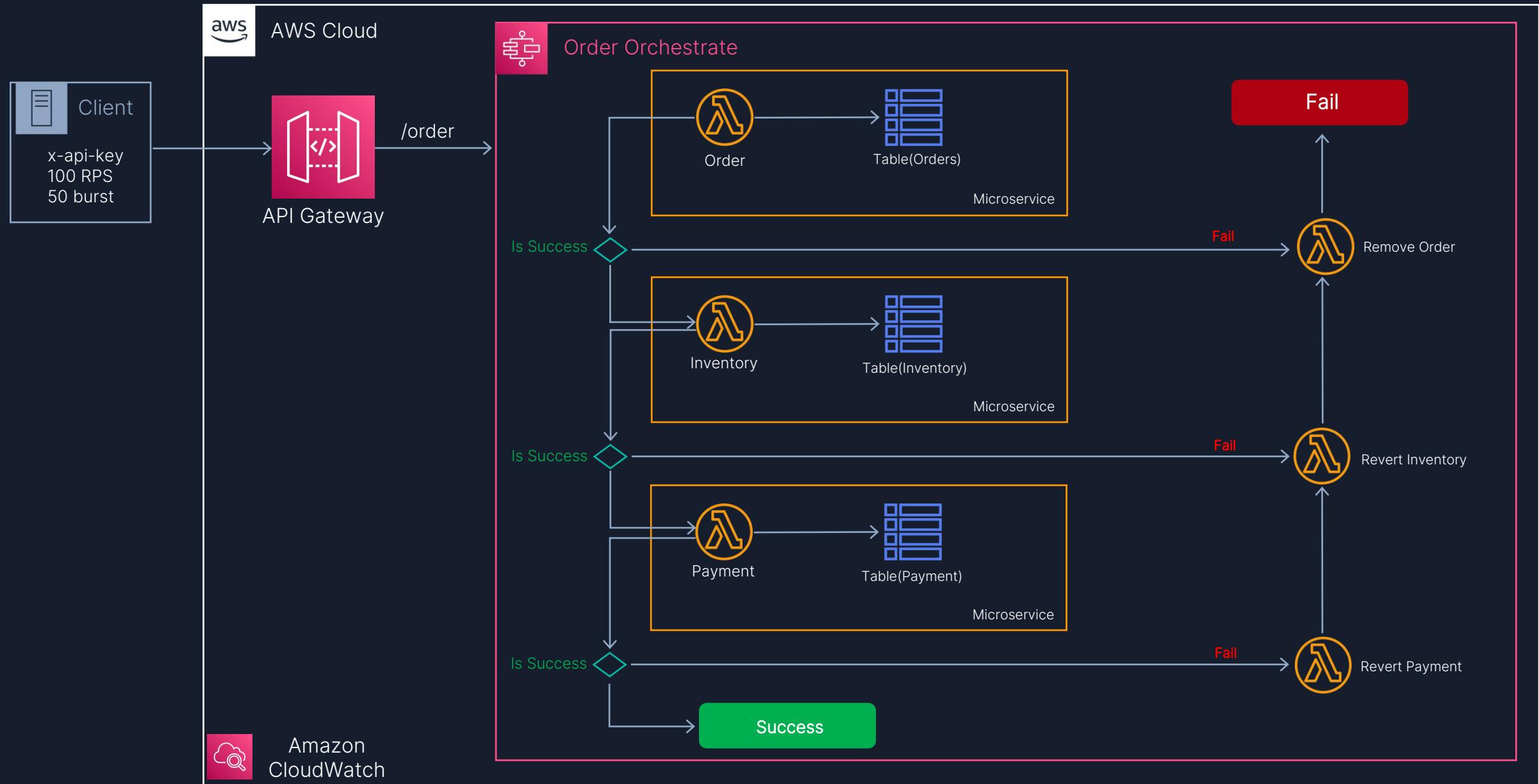
# Command, Publish, Consume, Query (CPCQ)



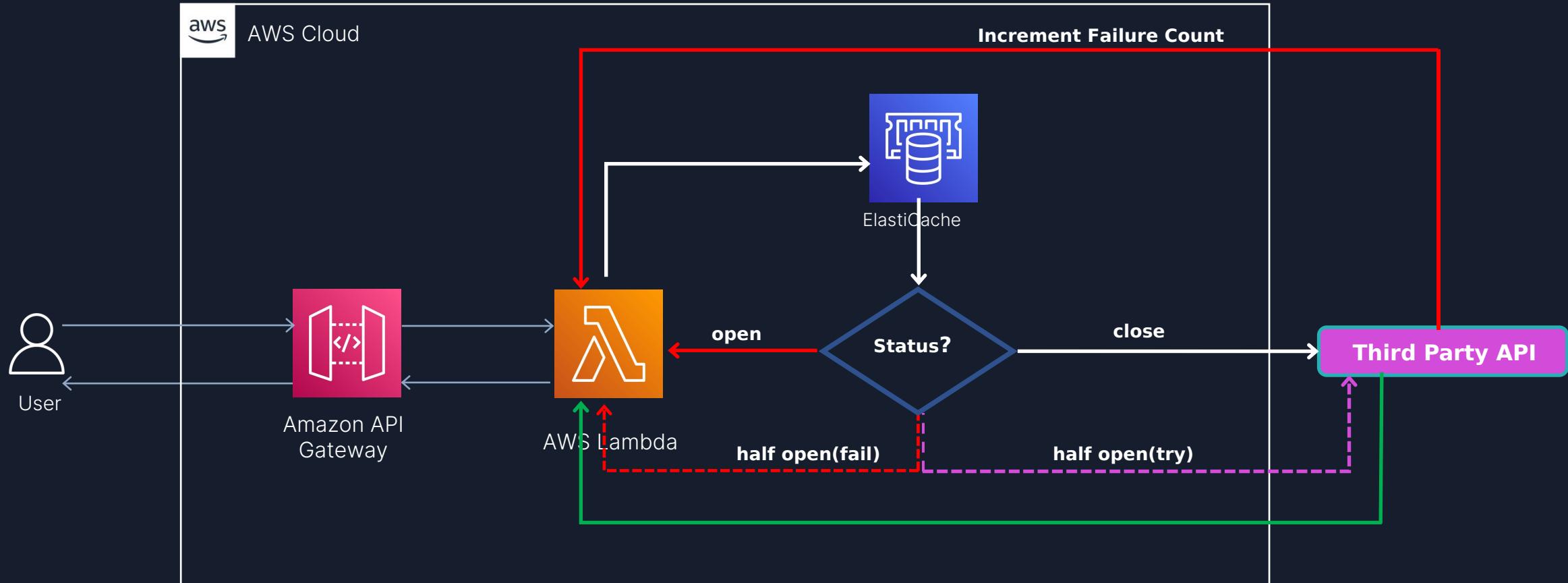
# Event hub



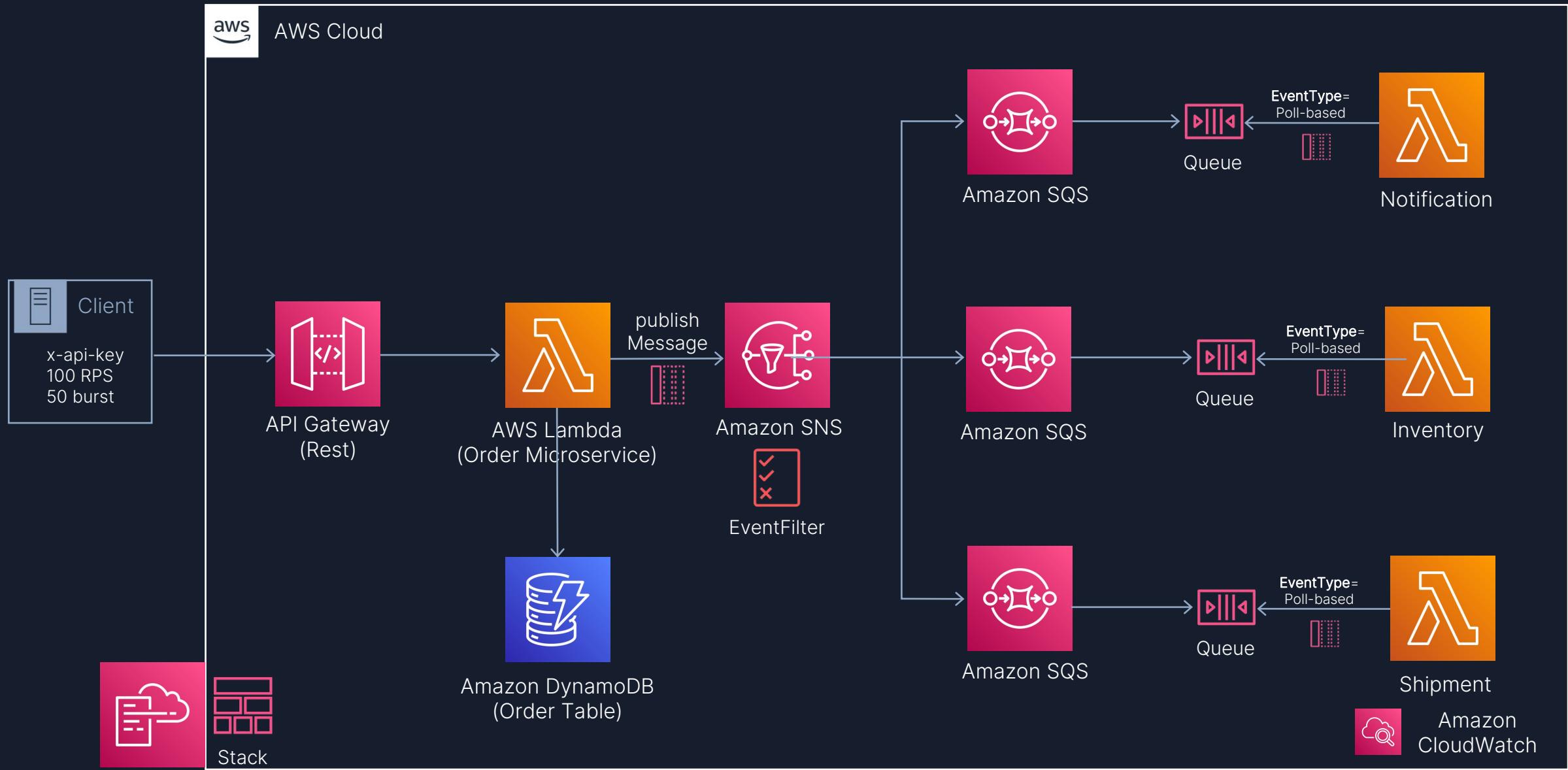
# Saga Pattern For Orchestrate Distributed Transaction (AWS StepFunctions)



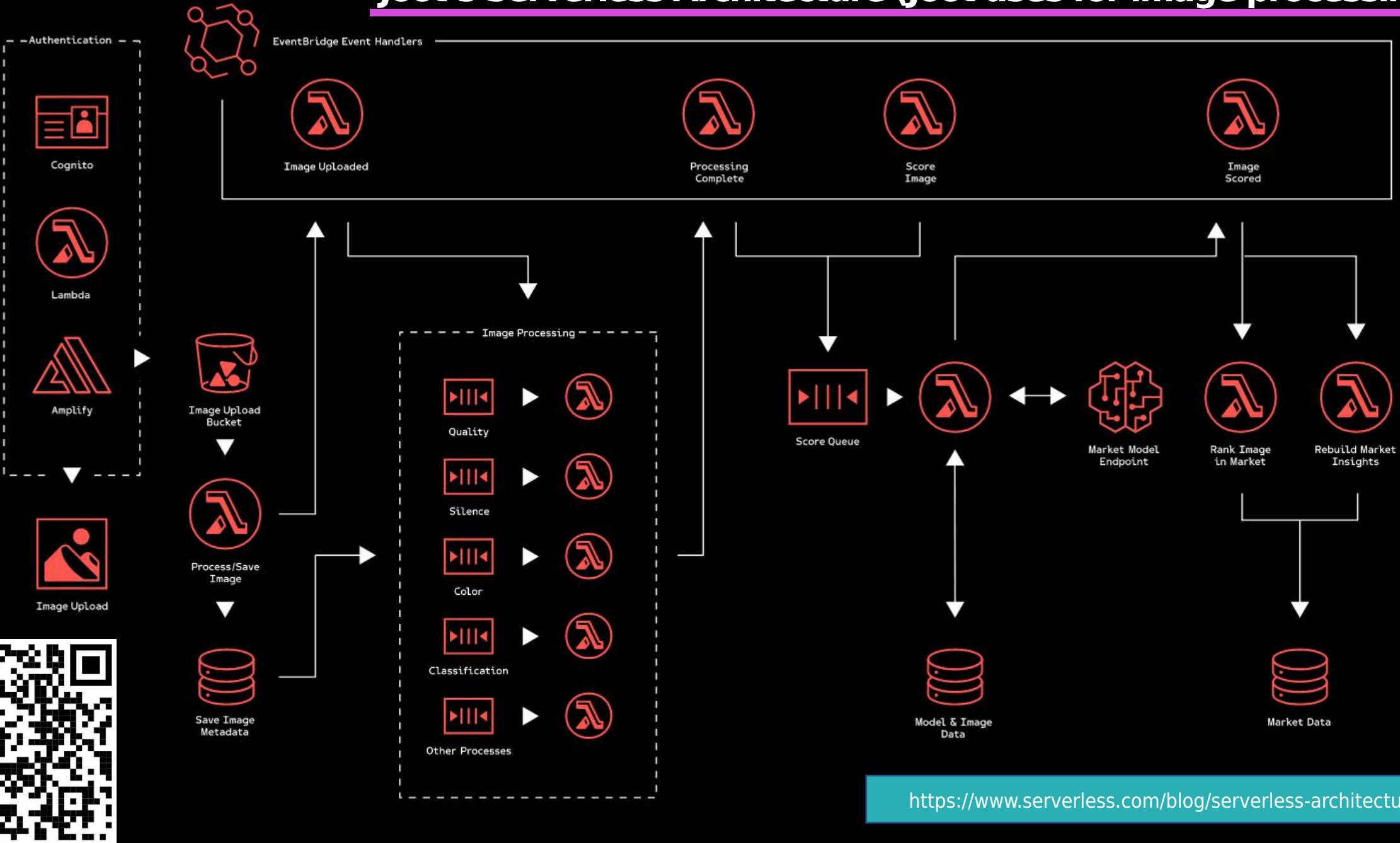
# Circuit Breaker and Many More by Jeremy Daly



# Fan-Out Serverless Architectures Using SNS, SQS and Lambda



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



<https://www.serverless.com/blog/serverless-architecture>



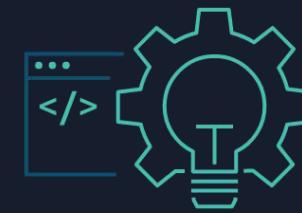
# AWS CDK for Java and Python



## 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](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

Success Info Warn Error

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

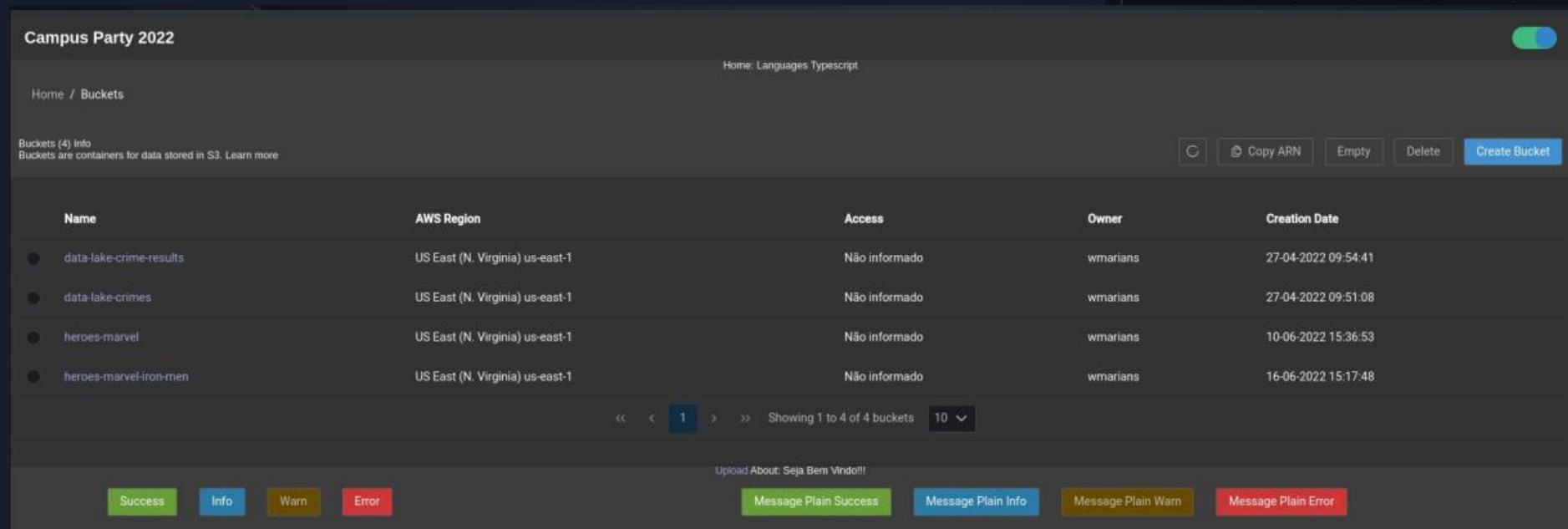


Figure: 1 - FrontEnd (Architecture React)

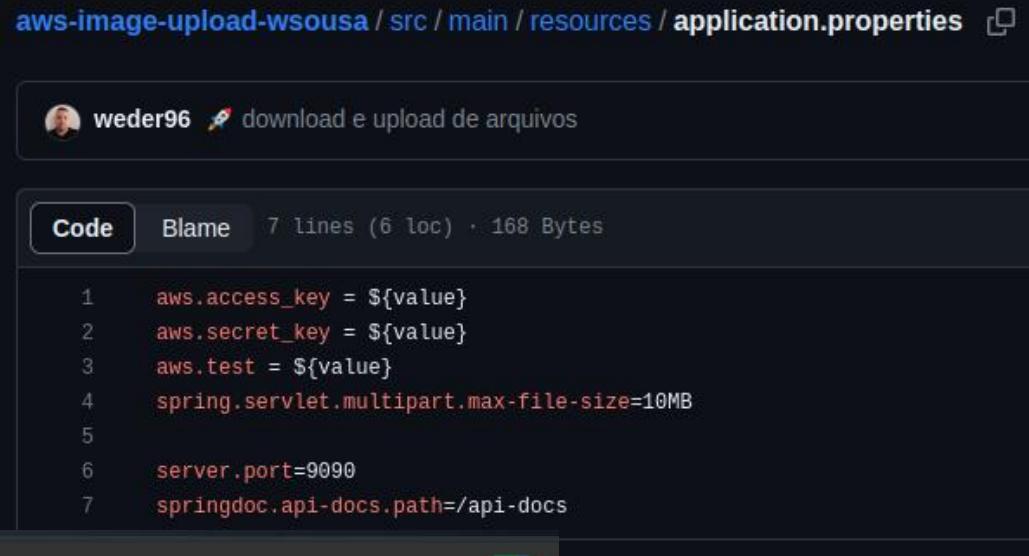
Figure: 2 - BackEnd (Architecture Spring boot(Java))

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

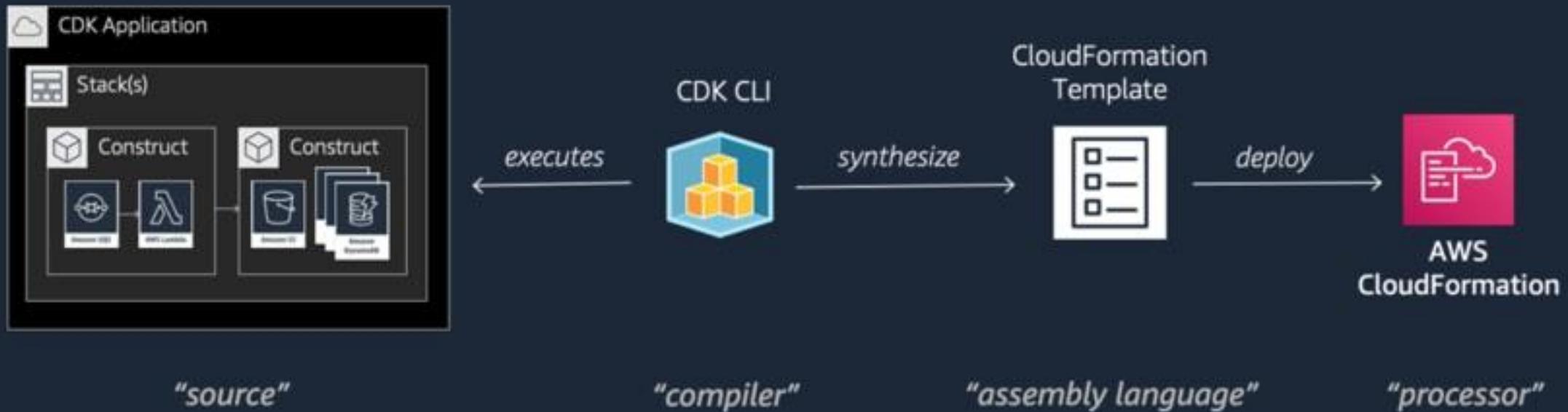
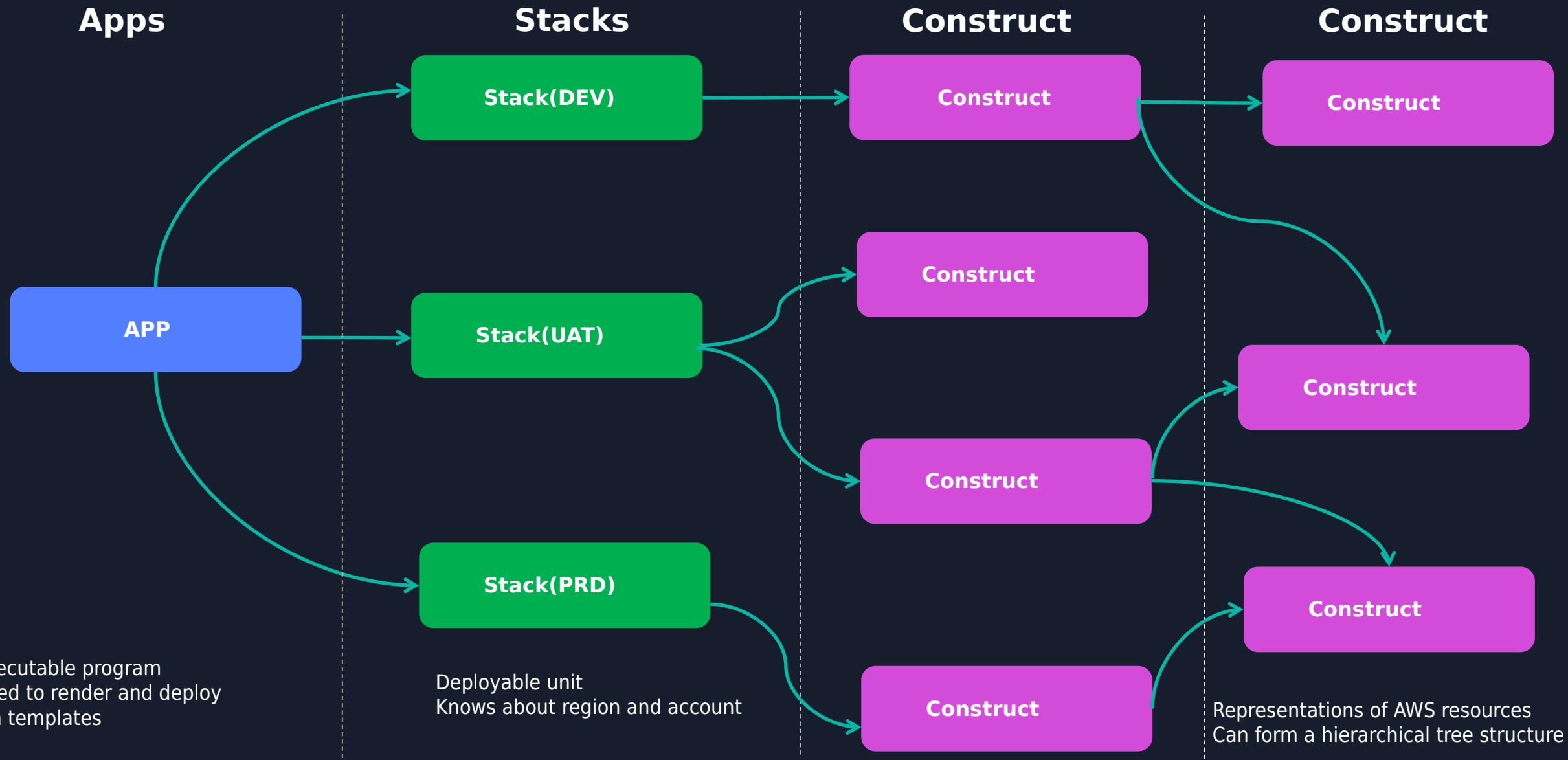


Figure: 1 - How Lifecycle CDK Development

# CDK Code Structure



# 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.

The screenshot shows a GitHub repository interface for a project named "poc-cdk-java". The repository structure is as follows:

- Files** tab selected.
- Code** tab selected for the file `CreateDynamoDb.java`.
- Blame**: 171 lines (142 loc) · 8.25 KB
- Code 55% faster with GitHub Copilot**
- Raw** button.

The code in `CreateDynamoDb.java` is as follows:

```
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 }
```

The repository structure includes:

- my-cdk** folder:
  - Lambda** folder (Python files): `app.py`, `createItem.py`, `deleteItem.py`, `getAllItems.py`, `getOneItem.py`, `updateItem.py`. These files are highlighted with a dashed purple border.
- node\_modules** folder.
- postman** folder.
- src** folder:
  - main/java/com/myorg** folder:
    - CreateDynamoDb.java** (highlighted with a dashed purple border).
    - `CreateLambdaCron.java`
    - `MyCdkApp.java`
    - `MyStack.java`
- test** folder.

Icons on the right side of the code editor include a QR code, a blue cloud icon with a cube, and a Java logo.

Figure: 1 - Project CDK (Java 11), folder lambda(Python 3.10)

<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 ~ ]$ █
```

Figure: 1 - Checking node version

## 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
```

Figure: 2 - install aws-cdk

## Verifying AWS CDK Installation

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

Figure: 3 - checking aws-cdk version install

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

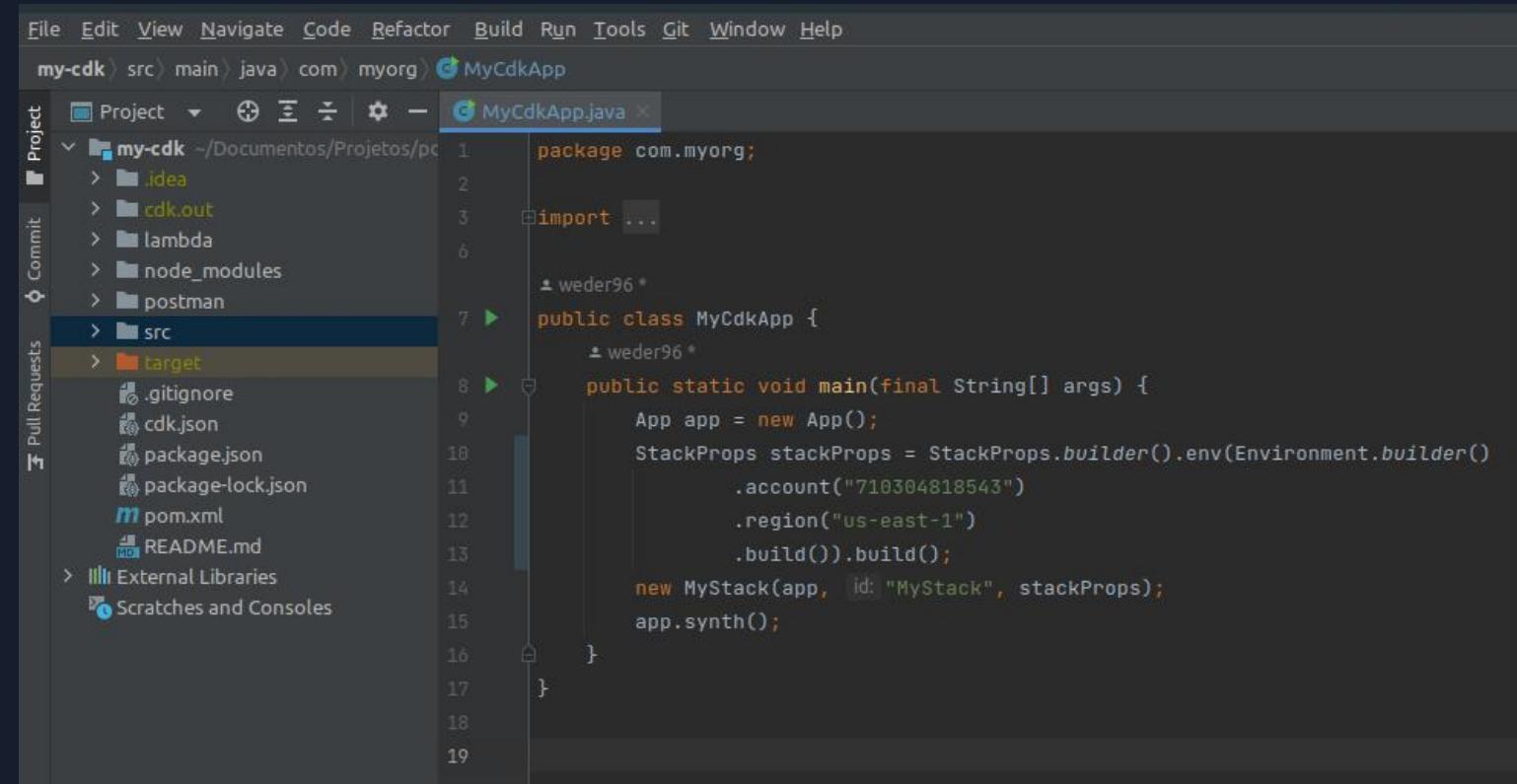
# Initializing a CDK Project



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
```

Figure: 1 - Create project CDK (Java)



The screenshot shows a Java code editor with the file `MyCdkApp.java` open. The code defines a class `MyCdkApp` that creates an `App` object and a `StackProps` object, then builds and synthesizes a stack named `MyStack`. The project structure on the left includes files like `.gitignore`, `cdk.json`, and `pom.xml`.

```
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();
    }
}
```

Figure: 2 - Structure project CDK (Java)

# 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.



Figure: 1 - validated Account and Region to created Stack

```
⠄ 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).
```

Figure: 2 - Enviroment checking and found (Account and Region)

# 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 titled "Terminal". The window has a dark background with light-colored text. At the top, there's a menu bar with options: Arquivo, Editar, Ver, Pesquisar, Terminal, Ajuda. Below the menu, the prompt shows "[weder@weder ~ ]\$ cdk synth". The terminal is currently empty, showing only the command entered.

Figure: 1 - Command aws-cdk generate AWS Cloudfromation(YAML)

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 terminal window showing a large block of CloudFormation YAML code. The code defines a REST API endpoint named "itemsApi" with a specific ID. It includes properties like "Metadata" and "Properties" (which contains a long URL), and outputs like "itemsApiEndpoint". Parameters include "BootstrapVersion" and "Rules". The code is very long and detailed, showing the full structure of the API definition.

Figure: 2 - File AWS Cloudfromation(YAML) Generated command aws-cdk

# 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
```

Figure: 1 - Checking AWS Cloudformation Generated

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)?

Figure: 2 - Showing services and policies that will be created

# Deploying the App



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

```
...Stack: deploying... [4/4]
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
```

Figure: 1 - Showing cdk creating all services aws

```
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
```

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

Figure: 2 - Showing services created(ARN)

## 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.

A screenshot of a terminal window titled "Terminal". The window has a dark background with light-colored text. At the top, there is a menu bar with options: Arquivo, Editar, Ver, Pesquisar, Terminal, and Ajuda. Below the menu bar, the title bar shows the user's name "[weder@weder ~ ]\$". The main area of the terminal contains the command "cdk destroy" entered by the user.

Figure: 1 - Command aws-cdk destroy

A screenshot of a terminal window showing the output of the "aws-cdk destroy" command. The text in the terminal reads:  
Are you sure you want to delete: MyStack (y/n)? y  
MyStack: destroying... [1/1]  
MyStack: destroyed  
The last line, "MyStack: destroyed", is highlighted with a red dashed rectangle.

Figure: 2 - Showing aws-cdk destroy success (necessary confirmation(Y))

# 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: "Local" and "Local (2)". The "Local (2)" tab is active and displays the output of the "cdk diff" command. The output lists numerous AWS resources (ApiGateway::Stage, Resource, Lambda::Permission, Method) with their names, types, and unique identifiers. A pink dashed box highlights the "Outputs" section at the bottom, which contains a single output entry: "Output itemsApi/Endpoint itemsApiEndpoint8392E274: {"Value": {"Fn::Join": ["", [{"Ref": "itemsApi28111E1C"}], ".exec".

```
[+] 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 itemsApiitemsidOPTIONS62B09100

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

Figure: 2 - Showing diff and list add or removes resources

# 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.

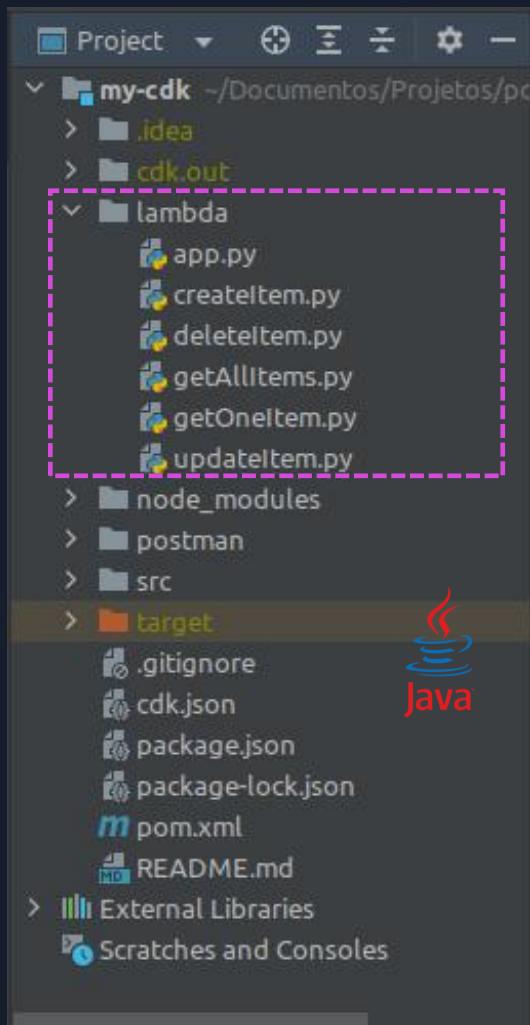


Figure: 1 - Code Lambda Python

```
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
```

Figure: 2 - Code Lamda insert order at table orders(fields method put\_item)

# 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**.

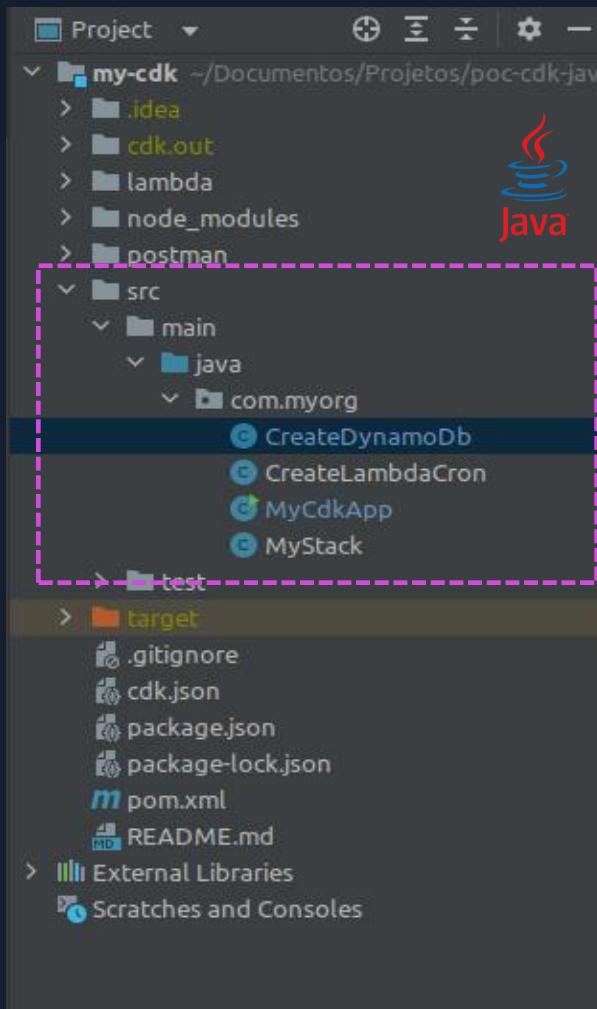


Figure: 1 - Code CDK(Java)

The screenshot shows the code editor in IntelliJ IDEA with the file 'CreateDynamoDb.java' open. The code is written in Java and uses the AWS CDK framework. It defines a private method 'getLambdaFunctionProps' that takes several parameters: 'MyStack', 'Role', 'Map<String, String>', 'String', 'String', and 'String'. The method body starts with a try block that reads a lambda function's content from a file and builds a 'FunctionProps' object. It then returns the built object. If an exception occurs, it prints the stack trace. Finally, it returns null. The code editor shows syntax highlighting for Java and AWS CDK constructs. To the right of the code, there is a Java logo icon. The bottom of the screen shows other standard IDE components like 'External Libraries' and 'Scratches and Consoles'.

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

Figure: 2 - Method Create Lambda function using CDK(Java)

# 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(Groups.asList(new String[] {"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"}))
        .resources(Groups.asList(new String[] {"arn:aws:logs:*:*:*"})).build();

    PolicyDocument policyDocument = PolicyDocument.Builder.create()
        .statements(Groups.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();
}
```



Figure: 1 - Code CDK(Java), creating Statements(effect, actions, resources) for Role

# 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;  
}
```

Figure: 3 - Code CDK(Java), props DynamoDB(names, partitionKey, sortKey)

```
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;  
}
```

Figure: 3 - Code CDK(Java), creating lambda Env

```
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);  
}
```

Figure: 1 - Code CDK(Java), creating DynamoDB, setting permission lambda functions



# 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;  
}
```

Figure: 3 - Code CDK(Java), reading file folder lambda

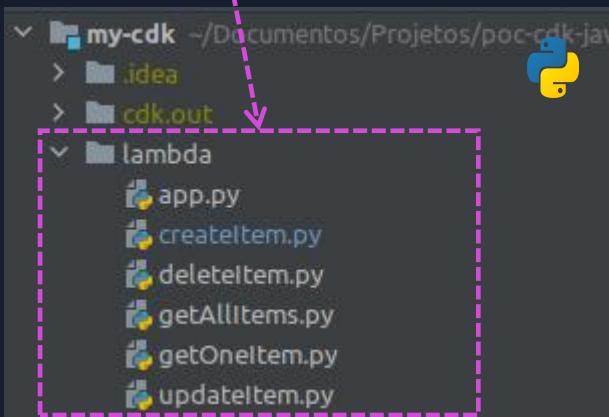


Figure: 4 - files folder lambda

```
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;  
}
```

Figure: 2 - Code CDK(Java), reading file python(py), settings(memorySize, role, timeout, version Python)

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

Figure: 1 - Code CDK(Java), created lambda Functions

# Code IAC (infrastructure as Code) - Create API Gateway



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

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

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

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

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

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

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

Figure: 1 - Code CDK(Java), created API Gateway

```
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<>();
```

Figure: 2 - Code CDK(Java), created Cors API Gateway

# Postman Call API Gateway

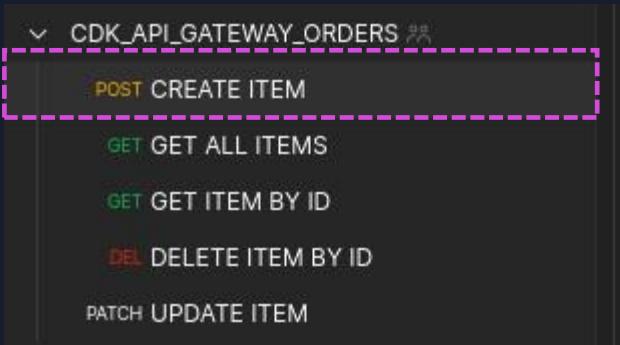


Figure: 1 - Collection Postman

The screenshot shows the 'POST /CREATE ITEM' request details in Postman. The request method is 'POST' and the URL is <https://kp90ktq1qb.execute-api.us-east-1.amazonaws.com/prod/items>. The 'Body' tab is selected, showing the raw JSON payload:

```
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 } ]
```

Figure: 2 - add item table order

# Postman Call API Gateway

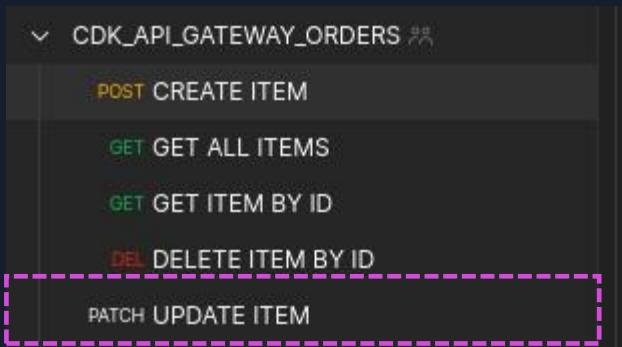


Figure: 1 - Collection Postman

The screenshot shows the 'PATCH UPDATE ITEM' request details in Postman. At the top, the method 'PATCH' is selected, and the URL is set to 'https://kp90ktq1qb.execute-api.us-east-1.amazonaws.com/prod/items/c0fe2aa0-a1d0-4d86-8091-ff76dda73086'. The 'Body' tab is active, showing a JSON payload:

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

Figure: 2 - update item table order

# Postman Call API Gateway

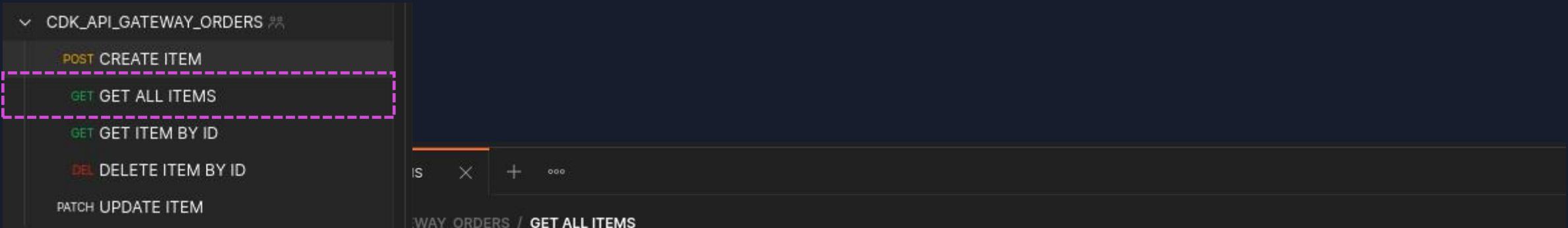


Figure: 1 - Collection Postman

The screenshot shows the 'GET ALL ITEMS' request details in Postman:

- Method: GET
- URL: https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items
- Params tab selected
- Query Params table:

KEY	VALUE	DESCRIPTION
- Body tab selected
- JSON response (Pretty):

```
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": "Xiomai readmi 12C",
10         "Name": "Weder Sousa2"
11     }
]
```

Figure: 2 - Get All items table order

# Postman Call API Gateway

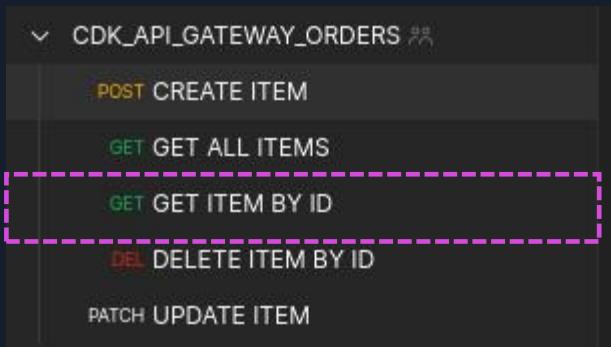


Figure: 1 - Collection Postman

A screenshot of the Postman application interface showing the details for the 'GET GET ITEM BY ID' endpoint. The URL is https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items/0d16a630-c89c-4b1f-b03d-aa179c737d76. The 'Body' tab is selected, displaying a JSON response with a single item. The JSON is shown in Pretty format:

```
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": "Xiomai readmi 12C"  
21      },  
22      "Name": {  
23        "S": "Weder Sousa2"  
24      }  
25    }  
26  ]  
27}  
28
```

Figure: 2 - Get One item by orderId table order

# Postman Call API Gateway

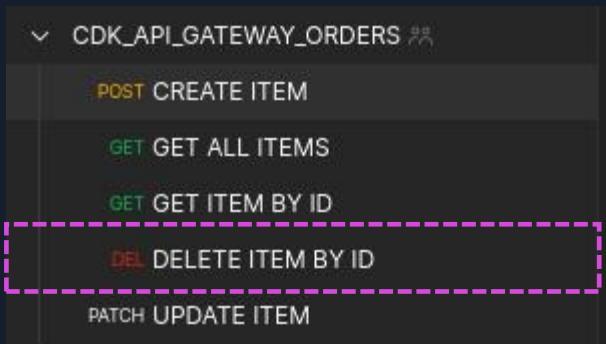


Figure: 1 - Collection Postman

The screenshot shows the main Postman interface for the "DELETE ITEM BY ID" endpoint. The URL is set to `https://41v8x3mpkd.execute-api.us-east-1.amazonaws.com/prod/items/0d16a630-c89c-4b1f-b03d-aa179c737d76`. The "Body" tab is selected, showing a JSON response:

```
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 }
```

Figure: 2 - Delete item by orderId table order

# Lambda Function Python(Boto3) - Create Item

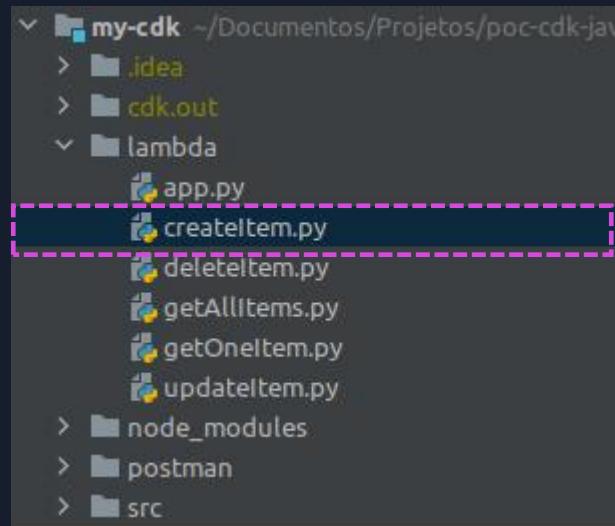


Figure: 1 - Lambda functions Folder

```
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
```

Figure: 2 - Put item table order (code Python boto3)

# Lambda Function Python(Boto3) - Update Item

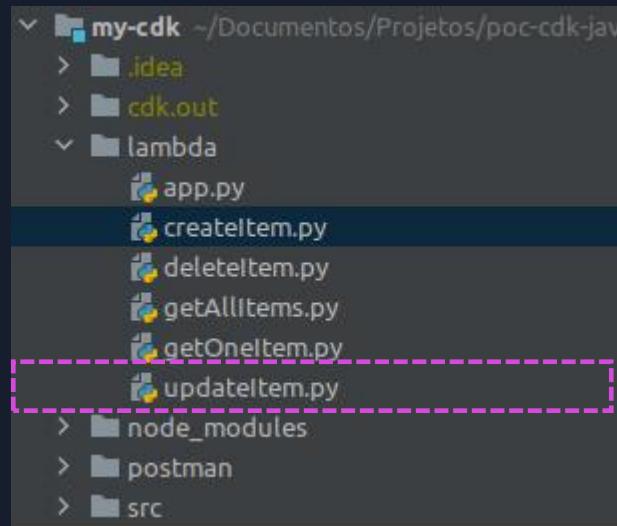


Figure: 1 - Lambda functions Folder

```
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
```

Figure: 2 - Update item table order (code Python boto3)

# Lambda Function Python(Boto3) - Get All Item

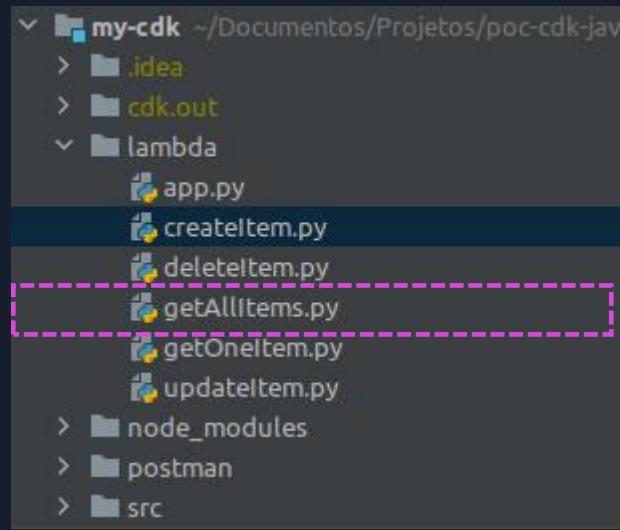


Figure: 1 - Lambda functions Folder

```
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
```

Figure: 2 - Get All items table order (code Python boto3)

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

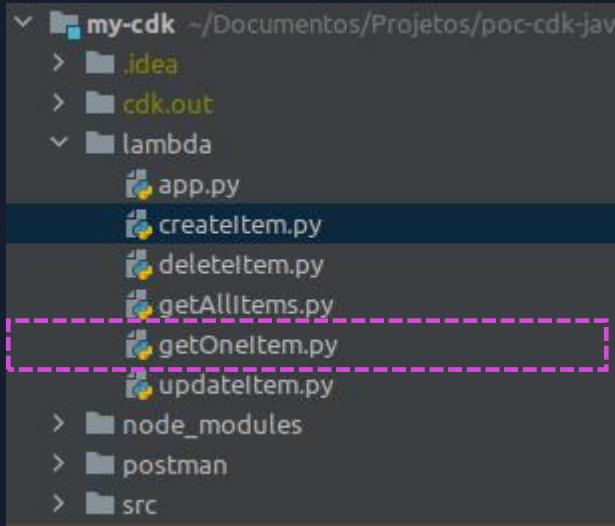


Figure: 1 - Lambda functions Folder

```
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
```

Figure: 2 - Get One item(by orderId) table order (code Python boto3)

# Lambda Function Python(Boto3) - Delete Item By Id

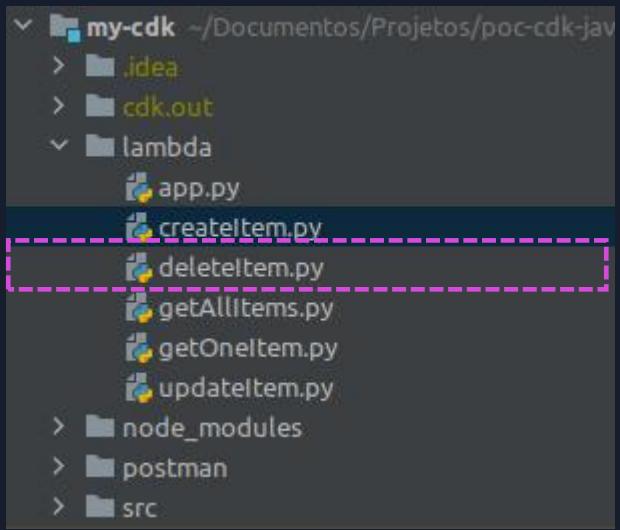
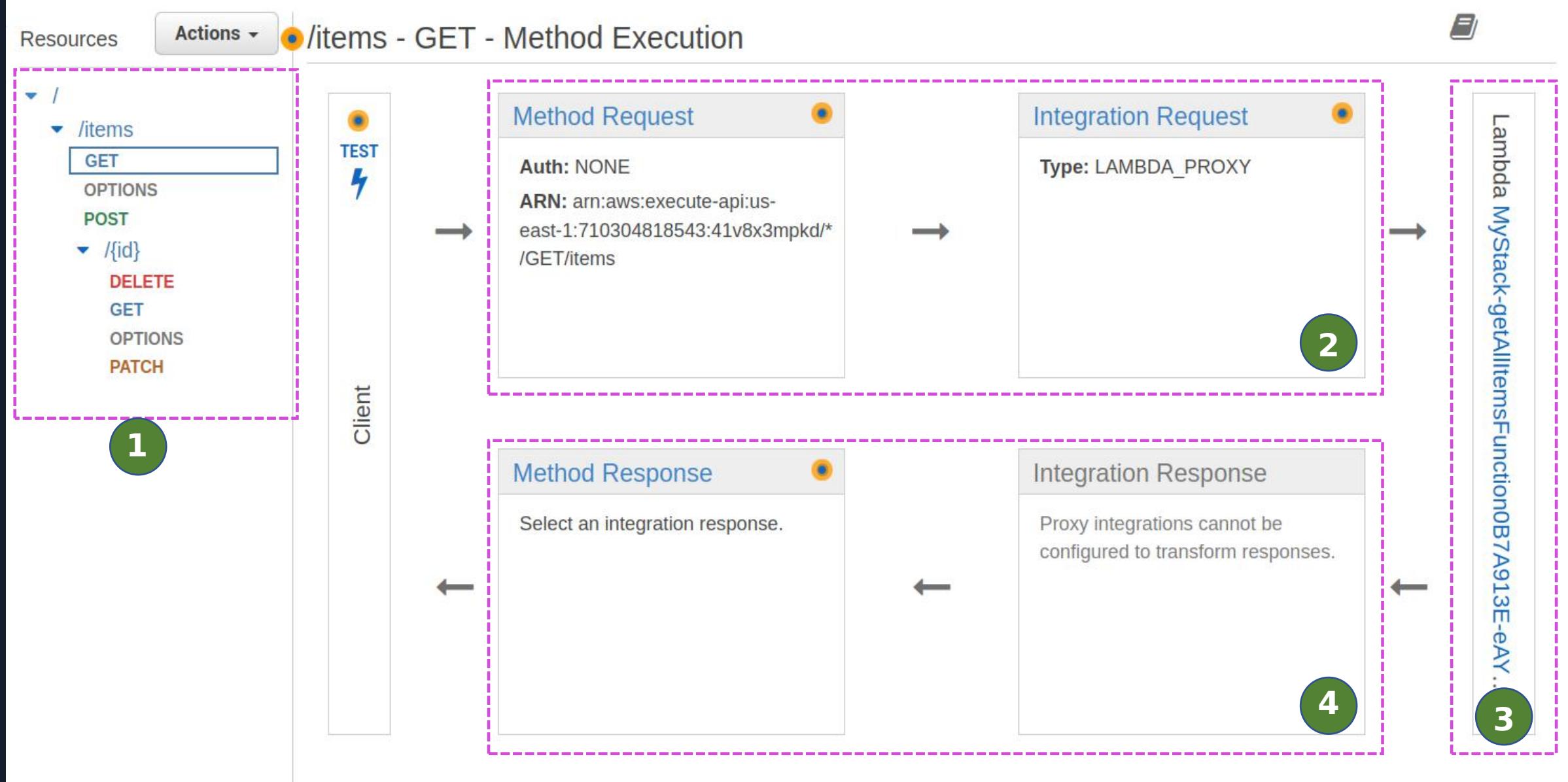


Figure: 1 - Lambda functions Folder

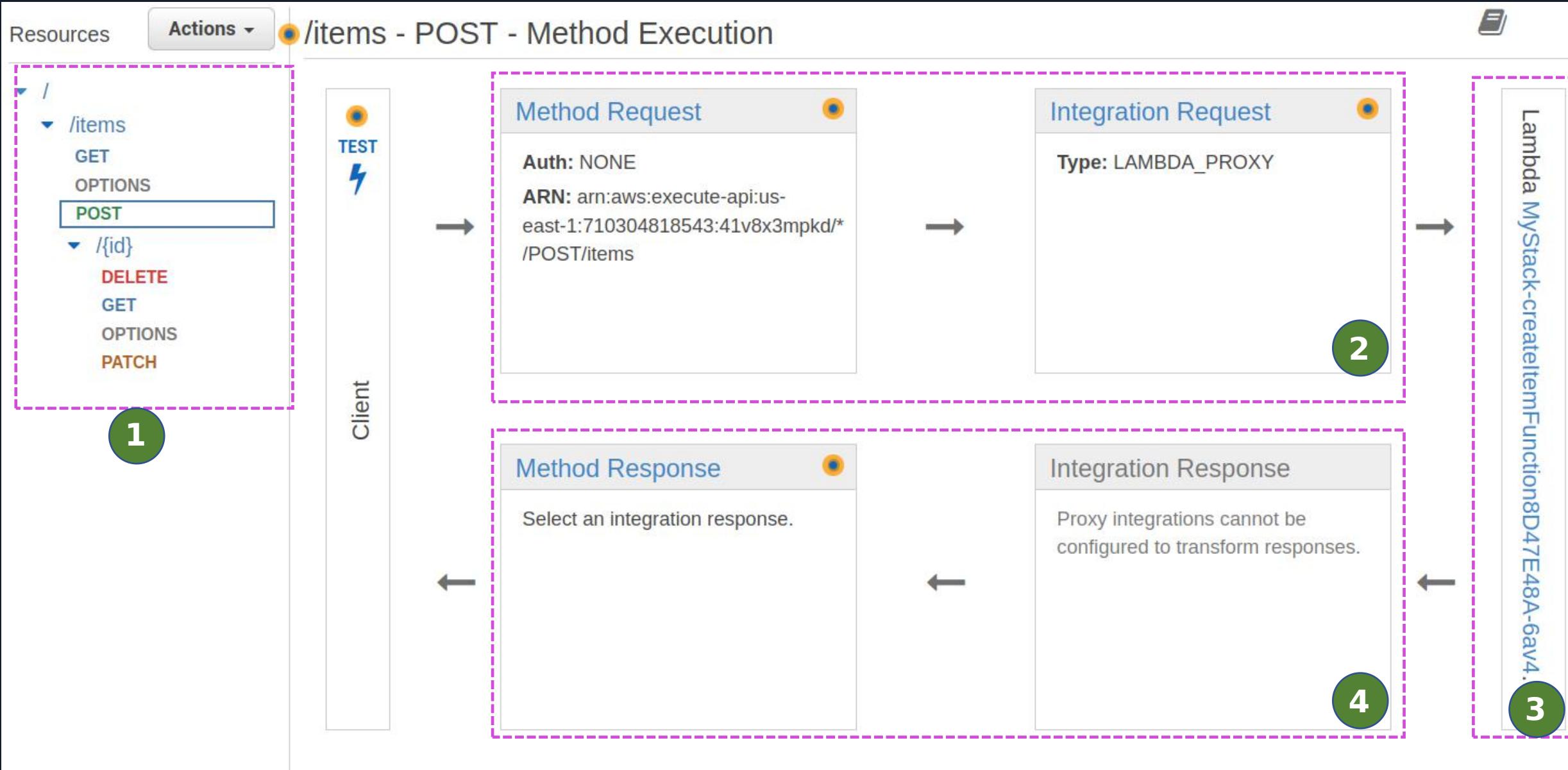
```
1 import json
2 import boto3
3 from boto3 import dynamodb
4 from boto3.dynamodb.conditions import Key, Attr
5
6
7 def lambda_handler(event, context):
8     print(event['pathParameters']['id'])
9
10    idParam = event['pathParameters']['id']
11    client = boto3.client('dynamodb')
12
13    item = 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)
```

Figure: 2 - Delete item(by orderId) table order (code Python boto3)

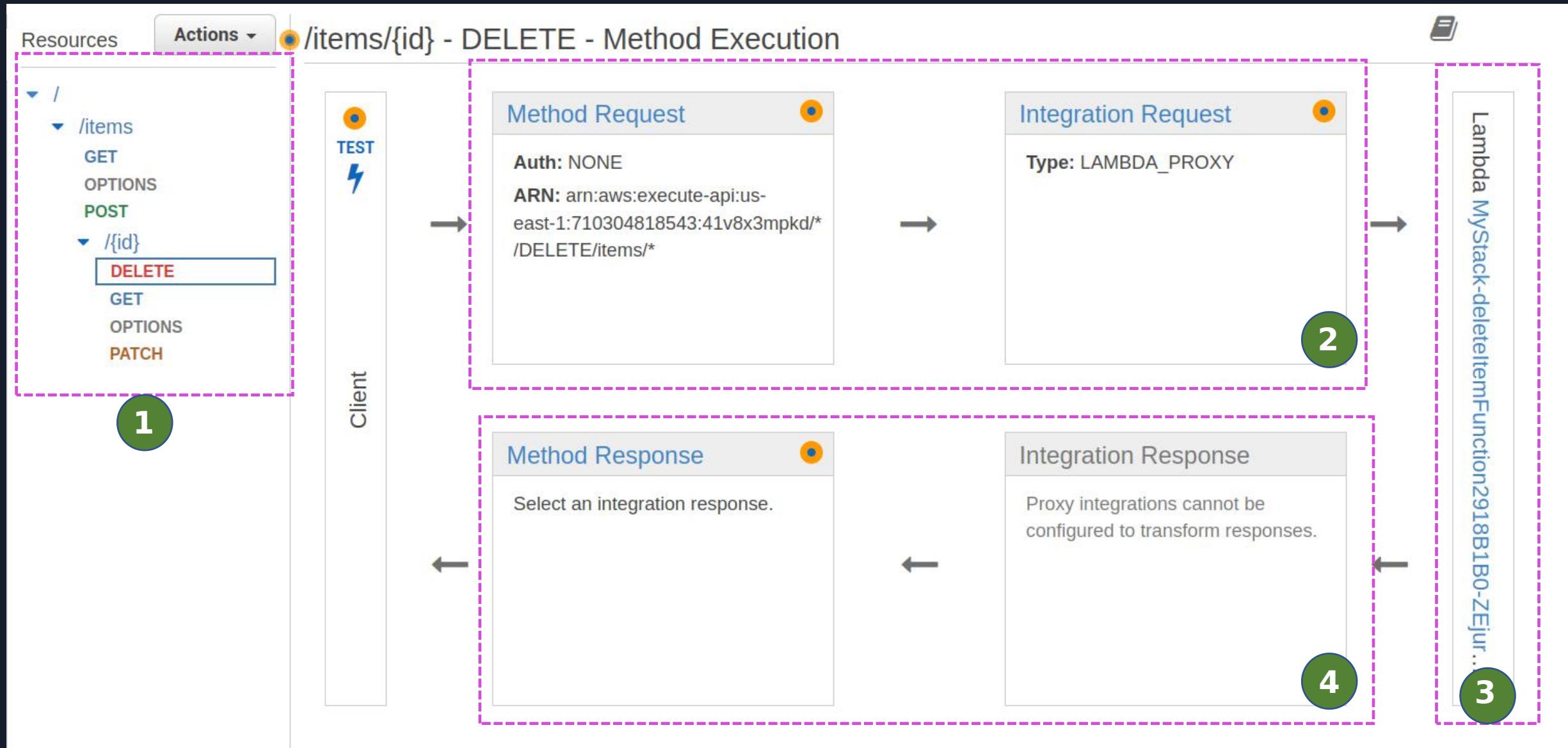
# AWS Resources (API Gateway) create(GET)



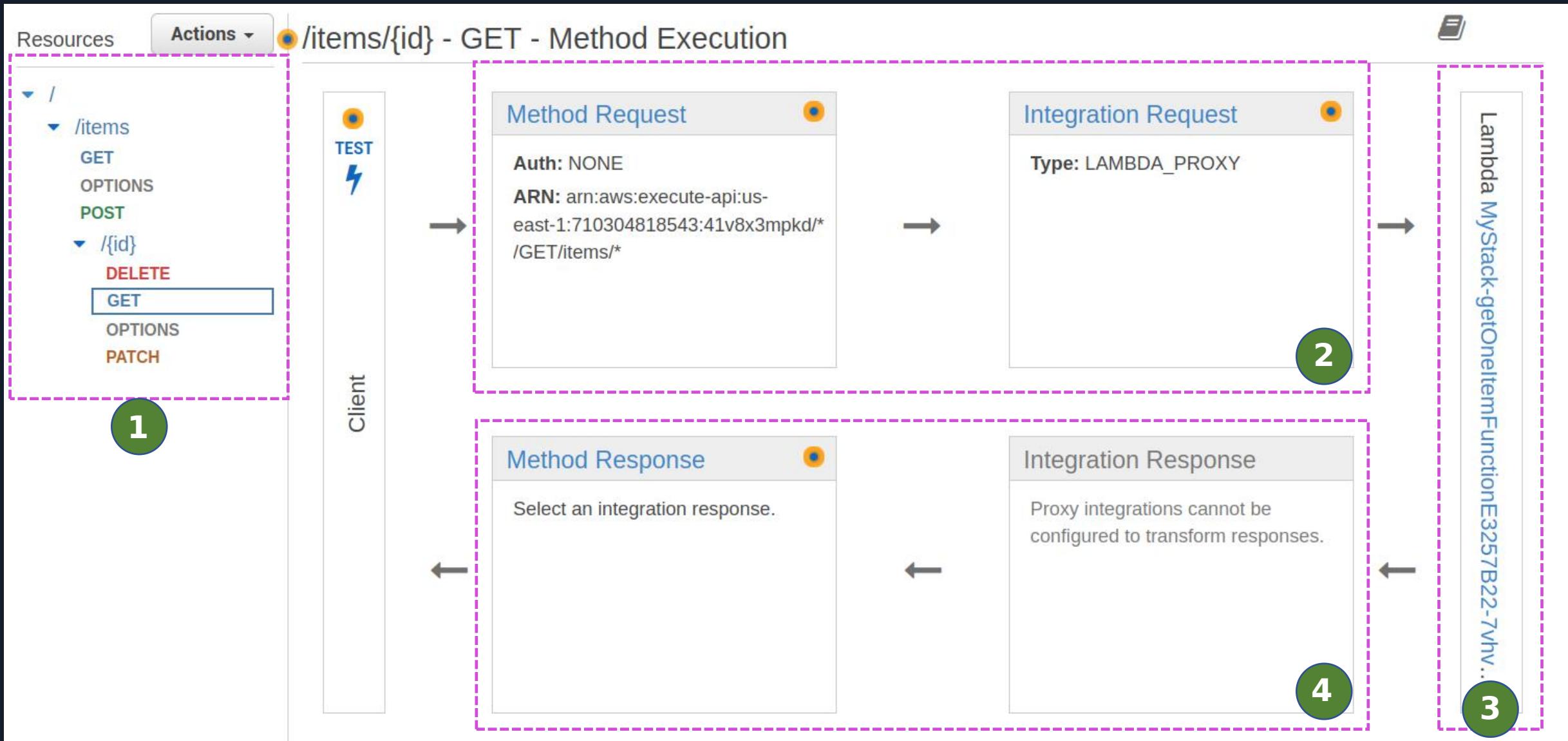
# AWS Resources (API Gateway) create (POST)



# 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					
<a href="#">MyStack-deleteItemFunction2918B1B0-ZEjurzRhyTEP</a>	Lambda deleteItem	Zip	Python 3.10	16 hours ago					
<a href="#">MyStack-getOneItemFunctionE3257B22-7vhvfNEQGTxk</a>	Lambda getOneItem	Zip	Python 3.10	20 hours ago					
<a href="#">MyStack-updateItemFunction59415205-Gqx4MKt4wOxX</a>	Lambda updateItem	Zip	Python 3.10	4 days ago					
<a href="#">MyStack-createItemFunction8D47E48A-6av4PrReEqoU</a>	Lambda createItem	Zip	Python 3.10	4 days ago					
<a href="#">MyStack-getAllItemsFunction0B7A913E-eAYHHix0qcMN</a>	Lambda getAllItems	Zip	Python 3.10	4 days ago					

Figure: 1 - Lambda functions created cdk(Java)

# AWS Resources create (DynamoDB)

The screenshot shows the AWS DynamoDB 'Tables' page with one table listed:

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

Filtering and search bars are at the top, along with 'Actions', 'Delete', and 'Create table' buttons.

Figure: 1 - table order

The screenshot shows the 'Explore Items' page for the 'orders' table:

Autopreview is enabled. A success message states: "Completed. Read capacity units consumed: 0.5".

The 'Items returned (1)' section shows the following item:

orderId (String)	orderDate (String)	desc	Email	Name
<a href="#">db9f81d5-fcfe-4bc4-a201-9051175dc256</a>	2023-09-12	Xiom... read...	weder.sous...	Weder So...

Figure: 2 - item table order

# AWS Resources create (IAM Role)

Roles (37) <small>Info</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.			
<input type="checkbox"/>	Role name	Trusted entities	Last activity
<input type="checkbox"/>	<a href="#">AWSServiceRoleForSupport</a>	AWS Service: support (Service-Linker)	-
<input type="checkbox"/>	<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS Service: trustedadvisor (Service)	-
<input type="checkbox"/>	<a href="#">cdk-hnb659fds-cfn-exec-role-710304818543-us-east-1</a>	AWS Service: cloudformation	3 days ago
<input type="checkbox"/>	<a href="#">cdk-hnb659fds-deploy-role-710304818543-us-east-1</a>	Account: 710304818543	3 days ago
<input type="checkbox"/>	<a href="#">cdk-hnb659fds-file-publishing-role-710304818543-us-east-1</a>	Account: 710304818543	3 days ago
<input type="checkbox"/>	<a href="#">cdk-hnb659fds-image-publishing-role-710304818543-us-east-1</a>	Account: 710304818543	-

Figure: 1 - Roles created aws-cdk



## Next Steps



# The AWS Step Functions Workshop



# The AWS Step Functions Workshop

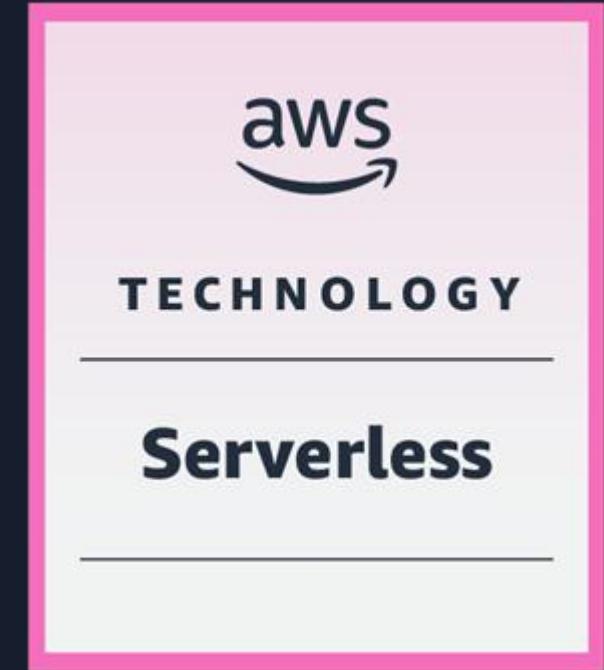


© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

<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)

<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](https://hands-on.cloud)

The screenshot shows the homepage of hands-on.cloud. At the top left, there are social media icons for Facebook, YouTube, and Twitter. In the top center is the hands-on.cloud logo, which consists of the text "hands-on.cloud" in blue with a small white cloud icon above the "o". To the right of the logo is a search bar with the placeholder text "serverless". 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 "Logout" button. The main content area features a large, semi-transparent background image of a person's hands typing on a keyboard. Overlaid on this image is the text "Hands-On.Cloud" in a large, bold, dark blue serif font. Below it is the subtitle "Tutorials, How-Tos for Cloud Engineers" in a smaller, lighter blue sans-serif font. A large, bold, dark blue word "serverless" is centered in the lower half of the main area. At the bottom left, there is a white button-like element with the word "SERVERLESS" in capital letters. At the bottom right, there is a teal-colored button with the URL "https://hands-on.cloud/" in white text.

serverless

Python Boto3 Tutorials Courses (In progress) AWS Certifications How-To Articles Write For Us Logout

# Hands-On.Cloud

Tutorials, How-Tos for Cloud Engineers

## serverless

SERVERLESS

https://hands-on.cloud/

# 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/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/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 Java - GFT

# Q & A



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