### **AWS CDK Project Report: Deploying AWS SQS and Lambda**

## **1. Introduction**

This project involves using **AWS CDK** to deploy an **AWS SQS queue**, an **AWS Lambda function**, and setting up event-driven execution by triggering the Lambda function from SQS. We also monitor logs to verify execution.

## **2. Prerequisites**

Before starting, ensure the following tools are installed on local machine:

* **Python 3.12 or later**
* **Node.js and npm**
* **AWS CDK**

### **Verify Installation**

Run the following commands to check if CDK is installed:

npm install -g aws-cdk

cdk --version

## **3. Setting Up the CDK Project**

Create a new AWS CDK Python project:

cdk init app --language=python

## 

## **4. Writing the Infrastructure Code**

Modify lib/sqs\_lambda\_stack.py to define the infrastructure:

from aws\_cdk import (

core as cdk,

aws\_lambda as lambda\_,

aws\_sqs as sqs,

aws\_lambda\_event\_sources as lambda\_event\_sources

)

from aws\_cdk.core import Duration

class SqsLambdaStack(cdk.Stack):

def \_\_init\_\_(self, scope: cdk.Construct, id: str, \*\*kwargs) -> None:

super().\_\_init\_\_(scope, id, \*\*kwargs)

# Creating SQS Queue

queue = sqs.Queue(

self, "Sqs-Lambda-Queue",

visibility\_timeout=Duration.seconds(300),

)

# Creating Lambda Function

sqs\_lambda = lambda\_.Function(self, "SQS-Lambda",

runtime=lambda\_.Runtime.PYTHON\_3\_12,

handler="lambda\_handler.handler",

code=lambda\_.Code.from\_asset('lambda')

)

# Creating Event Source

sqs\_event\_source = lambda\_event\_sources.SqsEventSource(queue)

# Add SQS Event Source to Lambda

sqs\_lambda.add\_event\_source(sqs\_event\_source)

## **5. Writing the Lambda Function Code**

Create a new folder **lambda/** and add a lambda\_handler.py file:

def handler(event, context):

print(event)

return {

'statusCode': 200,

'body': 'Success!'

}

## **6. Deploying the Infrastructure**

Run the following AWS CDK commands:

cdk bootstrap

cdk synth

cdk diff

cdk deploy

## **7. Monitoring and Logs**

### **Check Lambda Logs in AWS Console**

1. Open **AWS Lambda Console**.
2. Select the Lambda function **SQS-Lambda**.
3. Navigate to **Monitor > Logs** to check event triggers.

## **8. Destroying the Stack**

To clean up all deployed resources:

cdk destroy

## **9. Conclusion**

This project demonstrated how to:  
 ✅ Deploy an **SQS-triggered Lambda function** using **AWS CDK** ✅ Manage the infrastructure with **CDK commands** ✅ Monitor and verify execution through **CloudWatch logs** ✅ Destroy the stack for cleanup

This setup ensures **event-driven processing**, enabling scalable and efficient handling of messages in AWS. 🚀

***BONUS***

### **AWS CDK Setup and Commands Guide**

This guide covers the essential steps to configure AWS for CDK usage, deploy resources, and manage infrastructure efficiently.

## **1. Setting Up AWS for CDK**

1. **Create an IAM User**
   * Select **Local Code** while creating the IAM user.
   * Assign **AdministratorAccess** during creation or edit the user later to grant permissions.
2. **Generate AWS Access Keys**
   * Retrieve the **Access Key ID** and **Secret Access Key** from the IAM user settings.
3. **Configure AWS Locally**

Run the following command to set up AWS credentials:  
 aws configure

* + Enter the **Access Key ID**, **Secret Access Key**, **Region**, and **Output Format** as prompted.

📌 **Note:** This step ensures that CDK commands can interact with AWS services.

## **2. AWS CDK Commands**

### **2.1 Bootstrapping AWS Environment**

cdk bootstrap

*Initializes the required AWS resources for deploying CDK applications.*

### **2.2 Synthesizing the CloudFormation Template**

cdk synth

*Generates a CloudFormation template (template.json in the cdk.out folder) based on the CDK stack.*

### **2.3 Checking Differences Before Deployment**

cdk diff

*Compares the deployed resources in AWS with local CDK changes, helping in collaborative environments.*

### **2.4 Deploying the CDK Stack**

cdk deploy

*Pushes the CloudFormation template (template.json) to AWS and deploys the infrastructure.*

### **2.5 Destroying Resources**

#### **Dry Run Before Destroying**

cdk destroy --dry-run

*Simulates resource deletion to preview the impact before actual destruction.*

#### **Destroy All Stacks**

cdk destroy

*Deletes all stacks in the current environment.*

#### **Destroy a Specific Stack**

cdk destroy <mystack>

*Removes only the specified stack without affecting others.*

## **3. Important Best Practices**

✅ **Use CDK consistently** – Do not modify resources manually in the AWS console.  
 ✅ **Avoid direct Lambda code changes** – Update Lambda functions through CDK, not the AWS console.  
 ✅ **Manually delete log groups** – After destroying stacks, remove log groups manually in **CloudWatch Logs** to free up resources.

By following this structured approach, you ensure seamless deployment, efficient resource management, and avoid inconsistencies in AWS infrastructure. 🚀

**NOTE: Code is also added in the same git repository**.

* CDK Deploy will first deploy a stack in CloudFormation. This stack will show all the events describing the progress of each resource creation.