AWS Use cases

Tuesday, November 12, 2024

4:22 PM

This example illustrates a Lambda function triggered by an S3 event to perform a task on an EC2 instance.

Use Case

* Event Trigger: File upload to an S3 bucket.
* Action: Start an EC2 instance, process the file, and stop the EC2 instance.

Prerequisites

1. AWS Account: Ensure you have an AWS account set up.
2. AWS CLI: Install and configure the AWS CLI.
3. AWS SDK for Java: Include the AWS SDK for Java in your project.
4. Java IDE: Any Java IDE, such as IntelliJ IDEA or Eclipse.
5. Lambda Execution Role: Create an IAM role with necessary permissions for Lambda to interact with EC2 and S3.

Step 1: Setting Up Your Java Project

Maven POM file

Add the AWS SDK dependencies to your pom.xml file:

xml

1<project xmlns="http://maven.apache.org/POM/4.0.0"   
2 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
3 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <http://maven.apache.org/maven-v4_0_0.xsd>">  
4 <modelVersion>4.0.0</modelVersion>  
5 <groupId>com.example</groupId>  
6 <artifactId>aws-lambda-ec2-s3</artifactId>  
7 <version>1.0-SNAPSHOT</version>  
8 <dependencies>  
9 <dependency>  
10 <groupId>software.amazon.awssdk</groupId>  
11 <artifactId>s3</artifactId>  
12 <version>2.16.49</version>  
13 </dependency>  
14 <dependency>  
15 <groupId>software.amazon.awssdk</groupId>  
16 <artifactId>ec2</artifactId>  
17 <version>2.16.49</version>  
18 </dependency>  
19 <dependency>  
20 <groupId>com.amazonaws</groupId>  
21 <artifactId>aws-lambda-java-core</artifactId>  
22 <version>1.2.1</version>  
23 </dependency>  
24 <dependency>  
25 <groupId>com.amazonaws</groupId>  
26 <artifactId>aws-lambda-java-events</artifactId>  
27 <version>3.9.0</version>  
28 </dependency>  
29 </dependencies>  
30</project>

Step 2: Creating the Lambda Function

1. Create Lambda Handler Class: Create a class named S3ToEC2Handler.java in the src/main/java/com/example directory.
2. Lambda Function Code: Here is the Java code for handling the S3 event to start an EC2 instance:

java

1package com.example;  
2  
3import com.amazonaws.services.lambda.runtime.Context;  
4import com.amazonaws.services.lambda.runtime.RequestHandler;  
5import com.amazonaws.services.lambda.runtime.events.S3Event;  
6import com.amazonaws.services.s3.event.S3EventNotification.S3EventNotificationRecord;  
7import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;  
8import software.amazon.awssdk.regions.Region;  
9import software.amazon.awssdk.services.ec2.Ec2Client;  
10import software.amazon.awssdk.services.ec2.model.DescribeInstancesRequest;  
11import software.amazon.awssdk.services.ec2.model.DescribeInstancesResponse;  
12import software.amazon.awssdk.services.ec2.model.InstanceStateName;  
13import software.amazon.awssdk.services.ec2.model.StartInstancesRequest;  
14import software.amazon.awssdk.services.ec2.model.StopInstancesRequest;  
15import software.amazon.awssdk.services.s3.S3Client;  
16import software.amazon.awssdk.services.s3.model.GetObjectRequest;  
17import software.amazon.awssdk.services.s3.model.GetObjectResponse;  
18  
19import java.util.List;  
20  
21public class S3ToEC2Handler implements RequestHandler<S3Event, String> {  
22  
23 private final S3Client s3Client;  
24 private final Ec2Client ec2Client;  
25  
26 public S3ToEC2Handler() {  
27 Region region = Region.US\_EAST\_1;  
28 s3Client = S3Client.builder().region(region)  
29 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
30 .build();  
31  
32 ec2Client = Ec2Client.builder().region(region)  
33 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
34 .build();  
35 }  
36  
37 @Override  
38 public String handleRequest(S3Event s3Event, Context context) {  
39 String instanceId = "i-your-instance-id"; // Replace with your EC2 instance ID  
40  
41 try {  
42 for (S3EventNotificationRecord record : s3Event.getRecords()) {  
43 String bucketName = record.getS3().getBucket().getName();  
44 String objectKey = record.getS3().getObject().getKey();  
45  
46 // Process the file in the S3 bucket  
47 processS3File(bucketName, objectKey);  
48 }  
49  
50 // Start the EC2 instance  
51 startEC2Instance(instanceId);  
52  
53 // Simulate processing time  
54 Thread.sleep(60000); // Sleep for 1 minute  
55  
56 // Stop the EC2 instance  
57 stopEC2Instance(instanceId);  
58  
59 } catch (Exception e) {  
60 context.getLogger().log("Error: " + e.getMessage());  
61 }  
62  
63 return "Processing completed.";  
64 }  
65  
66 private void processS3File(String bucketName, String objectKey) {  
67 GetObjectRequest getObjectRequest = GetObjectRequest.builder()  
68 .bucket(bucketName)  
69 .key(objectKey)  
70 .build();  
71  
72 try {  
73 GetObjectResponse response = s3Client.getObject(getObjectRequest);  
74 System.out.println("Object content type: " + response.contentType());  
75 // Add your file processing logic here  
76 } catch (Exception e) {  
77 System.err.println("Error retrieving object from S3: " + e.getMessage());  
78 }  
79 }  
80  
81 private void startEC2Instance(String instanceId) {  
82 StartInstancesRequest startRequest = StartInstancesRequest.builder()  
83 .instanceIds(instanceId)  
84 .build();  
85  
86 ec2Client.startInstances(startRequest);  
87 System.out.println("EC2 instance started: " + instanceId);  
88  
89 // Wait for the instance to be running  
90 DescribeInstancesRequest describeRequest = DescribeInstancesRequest.builder()  
91 .instanceIds(instanceId)  
92 .build();  
93 DescribeInstancesResponse describeResponse;  
94 boolean isRunning = false;  
95 while (!isRunning) {  
96 describeResponse = ec2Client.describeInstances(describeRequest);  
97 String instanceState = describeResponse.reservations().get(0)  
98 .instances().get(0).state().nameAsString();  
99 if (InstanceStateName.RUNNING.toString().equals(instanceState)) {  
100 isRunning = true;  
101 } else {  
102 try {  
103 Thread.sleep(10000); // Wait for 10 seconds before checking again  
104 } catch (InterruptedException e) {  
105 Thread.currentThread().interrupt();  
106 throw new RuntimeException(e);  
107 }  
108 }  
109 }  
110 }  
111  
112 private void stopEC2Instance(String instanceId) {  
113 StopInstancesRequest stopRequest = StopInstancesRequest.builder()  
114 .instanceIds(instanceId)  
115 .build();  
116  
117 ec2Client.stopInstances(stopRequest);  
118 System.out.println("EC2 instance stopped: " + instanceId);  
119 }  
120}

Step 3: Deploy the Lambda Function

1. Create a Deployment Package:
   * Package the code into a ZIP file:

sh

1mvn clean package

1. Upload the Deployment Package:
   * Open the AWS Lambda console.
   * Create a new Lambda function.
   * Upload the ZIP file generated in the target directory.
2. Set Up the Trigger:
   * Add an S3 trigger to the Lambda function.
   * Select the bucket and event type (e.g., Object Created).
3. Environment Variables:
   * Configure environment variables for AWS credentials, if not using a role with necessary permissions.

Summary

This tutorial guides you through creating a Java-based Lambda function that uses AWS SDK to interact with EC2 and S3 services. The Lambda function is triggered by an S3 event, starts an EC2 instance, processes the uploaded file, and then stops the instance.

 java code in aws which inserts document in s3 then when insert happens need to start an ec2 which reads the doc then uses textract to analyze and prints it then send the doc to queue from queue consumer will pick it and insert in an rds table

Steps to Achieve the Task:

1. Insert Document into S3
2. Start an EC2 Instance Upon S3 Insertion
3. EC2 Reads the Document and Uses Textract to Analyze It
4. Send the Document to an SQS Queue
5. SQS Consumer Picks the Document from the Queue and Inserts It into an RDS Table

Prerequisites:

1. AWS Account: Ensure you have an AWS account set up.
2. AWS SDK for Java: Include the AWS SDK for Java in your project.
3. IAM Roles and Policies: Create IAM roles with the necessary permissions for Lambda, EC2, S3, SQS, and RDS services.
4. S3 Bucket: Create an S3 bucket where documents will be uploaded.
5. SQS Queue: Create an SQS queue.
6. RDS Database: Set up an RDS instance with the necessary database and table structure.

Step 1: Insert Document into S3

Create a class named S3Uploader.java to upload a document to S3:

java

1package com.example;  
2  
3import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;  
4import software.amazon.awssdk.regions.Region;  
5import software.amazon.awssdk.services.s3.S3Client;  
6import software.amazon.awssdk.services.s3.model.PutObjectRequest;  
7import software.amazon.awssdk.services.s3.model.S3Exception;  
8import java.nio.file.Paths;  
9  
10public class S3Uploader {  
11  
12 public static void main(String[] args) {  
13 S3Client s3 = S3Client.builder()  
14 .region(Region.US\_EAST\_1)  
15 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
16 .build();  
17  
18 String bucket = "your-bucket-name";  
19 String key = "your-document-name.pdf";  
20 String filePath = "/path-to-your-document/your-document-name.pdf";  
21  
22 try {  
23 s3.putObject(PutObjectRequest.builder().bucket(bucket).key(key).build(),  
24 Paths.get(filePath));  
25 System.out.println("Document uploaded to S3 bucket.");  
26 } catch (S3Exception e) {  
27 System.err.println(e.awsErrorDetails().errorMessage());  
28 }  
29 }  
30}

Step 2: Lambda to Trigger EC2 Instance on S3 Upload

Create a Lambda function that triggers an EC2 instance when a document is uploaded to S3. Use the AWS Lambda console to create the function and add the following Java code:

java

1package com.example;  
2  
3import com.amazonaws.services.lambda.runtime.Context;  
4import com.amazonaws.services.lambda.runtime.RequestHandler;  
5import com.amazonaws.services.lambda.runtime.events.S3Event;  
6import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;  
7import software.amazon.awssdk.regions.Region;  
8import software.amazon.awssdk.services.ec2.Ec2Client;  
9import software.amazon.awssdk.services.ec2.model.StartInstancesRequest;  
10  
11public class S3EventHandler implements RequestHandler<S3Event, String> {  
12  
13 private final Ec2Client ec2Client;  
14  
15 public S3EventHandler() {  
16 ec2Client = Ec2Client.builder()  
17 .region(Region.US\_EAST\_1)  
18 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
19 .build();  
20 }  
21  
22 @Override  
23 public String handleRequest(S3Event s3Event, Context context) {  
24 String instanceId = "i-your-instance-id"; // Your EC2 instance ID  
25  
26 try {  
27 StartInstancesRequest startRequest = StartInstancesRequest.builder()  
28 .instanceIds(instanceId)  
29 .build();  
30 ec2Client.startInstances(startRequest);  
31 System.out.println("EC2 instance started.");  
32  
33 } catch (Exception e) {  
34 System.err.println(e.getMessage());  
35 }  
36  
37 return "EC2 instance started.";  
38 }  
39}

Step 3: EC2 Instance Code to Analyze Document with Textract and Send to SQS

Create a Java application that will run on the EC2 instance. This application will use Textract to analyze the document and send the result to an SQS queue:

java

1package com.example;  
2  
3import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;  
4import software.amazon.awssdk.regions.Region;  
5import software.amazon.awssdk.services.ec2.Ec2Client;  
6import software.amazon.awssdk.services.ec2.model.StopInstancesRequest;  
7import software.amazon.awssdk.services.s3.S3Client;  
8import software.amazon.awssdk.services.s3.model.GetObjectRequest;  
9import software.amazon.awssdk.services.s3.model.GetObjectResponse;  
10import software.amazon.awssdk.services.sqs.SqsClient;  
11import software.amazon.awssdk.services.sqs.model.SendMessageRequest;  
12import software.amazon.awssdk.services.textract.TextractClient;  
13import software.amazon.awssdk.services.textract.model.\*;  
14  
15import java.util.List;  
16  
17public class EC2TextractProcessor {  
18  
19 private final S3Client s3Client;  
20 private final TextractClient textractClient;  
21 private final SqsClient sqsClient;  
22 private final Ec2Client ec2Client;  
23 private final String bucketName = "your-bucket-name";  
24 private final String objectKey = "your-document-name.pdf";  
25 private final String queueUrl = "<https://sqs.us-east-1.amazonaws.com/your-account-id/your-queue-name>";  
26  
27 public EC2TextractProcessor() {  
28 s3Client = S3Client.builder()  
29 .region(Region.US\_EAST\_1)  
30 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
31 .build();  
32  
33 textractClient = TextractClient.builder()  
34 .region(Region.US\_EAST\_1)  
35 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
36 .build();  
37  
38 sqsClient = SqsClient.builder()  
39 .region(Region.US\_EAST\_1)  
40 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
41 .build();  
42  
43 ec2Client = Ec2Client.builder()  
44 .region(Region.US\_EAST\_1)  
45 .credentialsProvider(EnvironmentVariableCredentialsProvider.create())  
46 .build();  
47 }  
48  
49 public static void main(String[] args) {  
50 EC2TextractProcessor processor = new EC2TextractProcessor();  
51 processor.analyzeDocument();  
52 processor.stopEC2Instance("i-your-instance-id");  
53 }  
54  
55 private void analyzeDocument() {  
56 GetObjectRequest getObjectRequest = GetObjectRequest.builder()  
57 .bucket(bucketName)  
58 .key(objectKey)  
59 .build();  
60  
61 try {  
62 GetObjectResponse response = s3Client.getObject(getObjectRequest);  
63 System.out.println("Object content type: " + response.contentType());  
64  
65 Document document = Document.builder()  
66 .s3Object(S3Object.builder().bucket(bucketName).name(objectKey).build())  
67 .build();  
68  
69 AnalyzeDocumentRequest analyzeRequest = AnalyzeDocumentRequest.builder()  
70 .document(document)  
71 .featureTypes(FeatureType.FORMS, FeatureType.TABLES)  
72 .build();  
73  
74 AnalyzeDocumentResponse analyzeResponse = textractClient.analyzeDocument(analyzeRequest);  
75  
76 for (Block block : analyzeResponse.blocks()) {  
77 if (block.blockType().equals(BlockType.LINE)) {  
78 String text = block.text();  
79 sendMessageToSQS(text);  
80 }  
81 }  
82  
83 } catch (Exception e) {  
84 System.err.println("Error processing document: " + e.getMessage());  
85 }  
86 }  
87  
88 private void sendMessageToSQS(String message) {  
89 SendMessageRequest sendMsgRequest = SendMessageRequest.builder()  
90 .queueUrl(queueUrl)  
91 .messageBody(message)  
92 .build();  
93  
94 sqsClient.sendMessage(sendMsgRequest);  
95 System.out.println("Message sent to SQS: " + message);  
96 }  
97  
98 private void stopEC2Instance(String instanceId) {  
99 StopInstancesRequest stopRequest = StopInstancesRequest.builder()  
100 .instanceIds(instanceId)  
101 .build();  
102  
103 ec2Client.stopInstances(stopRequest);  
104 System.out.println("EC2 instance stopped: " + instanceId);  
105 }  
106}

Step 4: SQS Consumer to Insert Data into RDS

Create a Lambda function that will consume messages from the SQS queue and insert them into an RDS table:

java

1package com.example;  
2  
3import com.amazonaws.services.lambda.runtime.Context;  
4import com.amazonaws.services.lambda.runtime.RequestHandler;  
5import com.amazonaws.services.lambda.runtime.events.SQSEvent;  
6import java.sql.Connection;  
7import java.sql.DriverManager;  
8import java.sql.PreparedStatement;  
9  
10public class SQSConsumerHandler implements RequestHandler<SQSEvent, String> {  
11  
12 @Override  
13 public String handleRequest(SQSEvent event, Context context) {  
14 String jdbcUrl = "jdbc:mysql://your-rds-endpoint:3306/your-database";  
15 String username = "your-username";  
16 String password = "your-password";  
17  
18 try (Connection connection = DriverManager.getConnection(jdbcUrl, username, password)) {  
19 for (SQSEvent.SQSMessage msg : event.getRecords()) {  
20 String message = msg.getBody();  
21 insertMessageIntoRDS(connection, message);  
22 }  
23 } catch (Exception e) {  
24 context.getLogger().log("Error: " + e.getMessage());  
25 }  
26  
27 return "Messages processed.";  
28 }  
29  
30 private void insertMessageIntoRDS(Connection connection, String message) throws Exception {  
31 String insertQuery = "INSERT INTO your\_table (column\_name) VALUES (?)";  
32 try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {  
33 preparedStatement.setString(1, message);  
34 preparedStatement.executeUpdate();  
35 }  
36 }  
37}

Summary

1. S3Uploader.java: Uploads a document to S3.
2. S3EventHandler.java: Lambda function triggered by S3 to start an EC2 instance.
3. EC2TextractProcessor.java: Runs on the EC2 instance, processes the document with Textract, and sends the result to an SQS queue.
4. SQSConsumerHandler.java: Lambda function that consumes messages from the SQS queue and inserts them into an RDS database.

This setup ensures that a document uploaded to S3 triggers a series of actions that lead to its analysis, processing, and storage in an RDS database. You may need to fine-tune the IAM roles, security groups, and other configurations to match your specific AWS environment.

AG51657\_evhealth: hi

AG51657\_evhealth: aws performance test, load test,stress test options

GitHub Copilot: AWS offers several services and tools for performance testing, load testing, and stress testing your applications. Here are some options:

### 1. \*\*AWS CloudWatch\*\*

AWS CloudWatch provides monitoring and observability of AWS resources and applications. It can be used to track performance metrics and set alarms based on thresholds.

### 2. \*\*AWS X-Ray\*\*

AWS X-Ray helps with analyzing and debugging distributed applications. It provides insights into the performance of your application and helps identify bottlenecks.

### 3. \*\*AWS CodeBuild\*\*

AWS CodeBuild can be used to run performance tests as part of your CI/CD pipeline. You can integrate it with testing frameworks to automate performance testing.

### 4. \*\*AWS Fargate\*\*

AWS Fargate allows you to run containers without managing servers. You can use it to run load testing tools like Apache JMeter or Locust in a containerized environment.

### 5. \*\*AWS Lambda\*\*

AWS Lambda can be used to create serverless functions that generate load on your application. This can be useful for simulating traffic and testing scalability.

### 6. \*\*Third-Party Tools\*\*

Several third-party tools can be integrated with AWS for performance testing:

- \*\*Apache JMeter\*\*: A popular open-source tool for load testing. You can run JMeter tests on AWS EC2 instances or containers.

- \*\*Locust\*\*: An open-source load testing tool that allows you to define user behavior with Python code. It can be run on AWS EC2 instances or containers.

- \*\*BlazeMeter\*\*: A cloud-based load testing service that integrates with JMeter and other testing tools. It can be used to run large-scale load tests on AWS infrastructure.

- \*\*Gatling\*\*: An open-source load testing tool that can be run on AWS EC2 instances or containers.

### Example: Running JMeter on AWS EC2

1. \*\*Launch an EC2 Instance\*\*: Start an EC2 instance with the desired specifications.

2. \*\*Install JMeter\*\*: SSH into the instance and install JMeter.

3. \*\*Run JMeter Tests\*\*: Upload your JMeter test plan and run the tests.

```bash

# Example commands to install JMeter on an EC2 instance

sudo yum update -y

sudo yum install -y java-1.8.0-openjdk

wget [https://downloads.apache.org//jmeter/binaries/apache-jmeter-5.4.1.tgz](https://downloads.apache.org/jmeter/binaries/apache-jmeter-5.4.1.tgz)

tar -xvzf apache-jmeter-5.4.1.tgz

cd apache-jmeter-5.4.1/bin

./jmeter -n -t /path/to/your/testplan.jmx -l /path/to/results.jtl

```

### Example: Running Locust on AWS Fargate

1. \*\*Create a Docker Image\*\*: Create a Docker image with Locust installed and your test scripts.

2. \*\*Push to ECR\*\*: Push the Docker image to Amazon Elastic Container Registry (ECR).

3. \*\*Run on Fargate\*\*: Create a Fargate task to run the Locust container.

```python

# Example Locust test script (locustfile.py)

from locust import HttpUser, TaskSet, task

class UserBehavior(TaskSet):

@task(1)

def index(self):

self.client.get("/")

class WebsiteUser(HttpUser):

tasks = [UserBehavior]

min\_wait = 5000

max\_wait = 9000

```

These are some of the options available for performance testing, load testing, and stress testing on AWS. You can choose the one that best fits your requirements and integrate it into your testing strategy.

AG51657\_evhealth: which aws services with java is most sought after

GitHub Copilot: Several AWS services are highly sought after for Java developers due to their robust integration with Java applications. Here are some of the most popular AWS services used with Java:

### 1. \*\*Amazon EC2 (Elastic Compute Cloud)\*\*

Amazon EC2 provides scalable computing capacity in the cloud. Java developers can deploy and manage Java applications on EC2 instances.

### 2. \*\*Amazon RDS (Relational Database Service)\*\*

Amazon RDS makes it easy to set up, operate, and scale a relational database in the cloud. It supports several database engines, including MySQL, PostgreSQL, Oracle, and SQL Server, which can be accessed from Java applications using JDBC.

### 3. \*\*Amazon S3 (Simple Storage Service)\*\*

Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. Java developers can use the AWS SDK for Java to interact with S3 for storing and retrieving data.

### 4. \*\*AWS Lambda\*\*

AWS Lambda lets you run code without provisioning or managing servers. Java is one of the supported languages for AWS Lambda, allowing developers to run Java code in response to events.

### 5. \*\*Amazon DynamoDB\*\*

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. Java developers can use the AWS SDK for Java to interact with DynamoDB.

### 6. \*\*Amazon SNS (Simple Notification Service)\*\*

Amazon SNS is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. Java applications can use SNS to send notifications.

### 7. \*\*Amazon SQS (Simple Queue Service)\*\*

Amazon SQS is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. Java developers can use SQS to manage message queues.

### 8. \*\*AWS Elastic Beanstalk\*\*

AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java. It automatically handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring.

### 9. \*\*Amazon Kinesis\*\*

Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data. Java developers can use the AWS SDK for Java to interact with Kinesis streams.

### 10. \*\*Amazon CloudWatch\*\*

Amazon CloudWatch is a monitoring and observability service. Java applications can use CloudWatch to collect and track metrics, collect and monitor log files, and set alarms.

### Example: Using AWS SDK for Java with S3

Here's an example of how to use the AWS SDK for Java to interact with Amazon S3:

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;

public class S3Example {

public static void main(String[] args)

{

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String bucketName = "your-bucket-name";

String keyName = "your-key-name";

String filePath = "path/to/your/file";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

s3Client.putObject(new PutObjectRequest(bucketName, keyName, new File(filePath)));

System.out.println("File uploaded successfully to S3 bucket " + bucketName);

}

}

```

### Example: Using AWS Lambda with Java

Here's an example of a simple AWS Lambda function written in Java:

```java

import com.amazonaws.services.lambda.runtime.Context;

import com.amazonaws.services.lambda.runtime.RequestHandler;

public class HelloWorldLambda implements RequestHandler<Object, String> {

@Override

public String handleRequest(Object input, Context context) {

return "Hello, World!";

}

}

```

These examples demonstrate how Java developers can leverage AWS services to build and deploy applications. The AWS SDK for Java provides comprehensive support for interacting with various AWS services, making it a powerful tool for Java developers.

can you give a java code in aws which inserts document in s3 then when insert happens need to start an ec2 which reads the doc then uses textract to analyze and prints it then send the doc to queue from queue consumer will pick it and insert in an rds table

GitHub Copilot: Sure! Below is a high-level example of how you can achieve this using AWS services with Java. This example includes the following steps:

1. \*\*Insert a document into S3.\*\*

2. \*\*Trigger an EC2 instance to read the document from S3.\*\*

3. \*\*Use Amazon Textract to analyze the document.\*\*

4. \*\*Send the analyzed document to an SQS queue.\*\*

5. \*\*Consume the message from the SQS queue and insert it into an RDS table.\*\*

### Step 1: Insert a Document into S3

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;

public class S3Uploader {

public static void main(String

[]

args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String bucketName = "your-bucket-name";

String keyName = "your-key-name";

String filePath = "path/to/your/file";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

s3Client.putObject(new PutObjectRequest(bucketName, keyName, new File(filePath)));

System.out.println("File uploaded successfully to S3 bucket " + bucketName);

}

}

```

### Step 2: Trigger an EC2 Instance

You can use AWS Lambda to trigger an EC2 instance when a new object is uploaded to S3. Here is an example Lambda function:

```java

import com.amazonaws.services.lambda.runtime.Context;

import com.amazonaws.services.lambda.runtime.events.S3Event;

import com.amazonaws.services.ec2.AmazonEC2;

import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;

import com.amazonaws.services.ec2.model.StartInstancesRequest;

public class S3EventHandler {

public void handleRequest(S3Event s3event, Context context) {

String instanceId = "your-ec2-instance-id";

AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StartInstancesRequest startRequest = new StartInstancesRequest().withInstanceIds(instanceId);

ec2.startInstances(startRequest);

System.out.println("EC2 instance started: " + instanceId);

}

}

```

### Step 3: Use Amazon Textract to Analyze the Document

On the EC2 instance, you can use the following code to read the document from S3 and analyze it using Amazon Textract:

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.textract.AmazonTextract;

import com.amazonaws.services.textract.AmazonTextractClientBuilder;

import com.amazonaws.services.textract.model.Document;

import com.amazonaws.services.textract.model.S3Object;

import com.amazonaws.services.textract.model.AnalyzeDocumentRequest;

import com.amazonaws.services.textract.model.AnalyzeDocumentResult;

public class TextractAnalyzer {

public static void main(String[] args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String bucketName = "your-bucket-name";

String keyName = "your-key-name";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonTextract textractClient = AmazonTextractClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

Document document = new Document().withS3Object(new S3Object().withBucket(bucketName).withName(keyName));

AnalyzeDocumentRequest request = new AnalyzeDocumentRequest().withDocument(document);

AnalyzeDocumentResult result = textractClient.analyzeDocument(request);

System.out.println("Textract analysis result: " + result);

}

}

```

### Step 4: Send the Analyzed Document to an SQS Queue

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.SendMessageRequest;

public class SQSProducer {

public static void main(String[] args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String queueUrl = "your-sqs-queue-url";

String messageBody = "your-message-body";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

SendMessageRequest sendMsgRequest = new SendMessageRequest()

.withQueueUrl(queueUrl)

.withMessageBody(messageBody);

sqsClient.sendMessage(sendMsgRequest);

System.out.println("Message sent to SQS queue: " + queueUrl);

}

}

```

### Step 5: Consume the Message from the SQS Queue and Insert into RDS

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.ReceiveMessageRequest;

import com.amazonaws.services.sqs.model.Message;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class SQSConsumer {

public static void main(String[] args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String queueUrl = "your-sqs-queue-url";

String jdbcUrl = "your-rds-jdbc-url";

String dbUser = "your-db-username";

String dbPassword = "your-db-password";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

ReceiveMessageRequest receiveMsgRequest = new ReceiveMessageRequest(queueUrl);

for (Message message : sqsClient.receiveMessage(receiveMsgRequest).getMessages()) {

String messageBody = message.getBody();

System.out.println("Received message: " + messageBody);

try (Connection conn = DriverManager.getConnection(jdbcUrl, dbUser, dbPassword)) {

String sql = "INSERT INTO your\_table (column1, column2) VALUES (?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, messageBody); // Adjust as needed

pstmt.setString(2, "additional-data"); // Adjust as needed

pstmt.executeUpdate();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

```

### Summary

1. \*\*S3Uploader\*\*: Uploads a document to S3.

2. \*\*S3EventHandler\*\*: AWS Lambda function that triggers an EC2 instance when a new object is uploaded to S3.

3. \*\*TextractAnalyzer\*\*: EC2 instance reads the document from S3 and analyzes it using Amazon Textract.

4. \*\*SQSProducer\*\*: Sends the analyzed document to an SQS queue.

5. \*\*SQSConsumer\*\*: Consumes the message from the SQS queue and inserts it into an RDS table.

This is a high-level example, and you may need to adjust the code to fit your specific requirements and handle error cases appropriately.

1. \*\*Uploading a document to S3.\*\*

2. \*\*Triggering an EC2 instance via AWS Lambda when a new object is uploaded to S3.\*\*

3. \*\*Reading the document from S3 on the EC2 instance, analyzing it using Amazon Textract, and sending the results to an SQS queue.\*\*

4. \*\*Consuming the message from the SQS queue and inserting it into an RDS table.\*\*

### Step 1: Upload a Document to S3

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;

public class S3Uploader {

public static void main(String

[]

args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String bucketName = "your-bucket-name";

String keyName = "your-key-name";

String filePath = "path/to/your/file";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

s3Client.putObject(new PutObjectRequest(bucketName, keyName, new File(filePath)));

System.out.println("File uploaded successfully to S3 bucket " + bucketName);

}

}

```

### Step 2: Trigger an EC2 Instance via AWS Lambda

Create an AWS Lambda function that starts an EC2 instance when a new object is uploaded to S3. This function can be written in Java or any other supported language. Here is an example in Java:

```java

import com.amazonaws.services.lambda.runtime.Context;

import com.amazonaws.services.lambda.runtime.events.S3Event;

import com.amazonaws.services.ec2.AmazonEC2;

import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;

import com.amazonaws.services.ec2.model.StartInstancesRequest;

public class S3EventHandler {

public void handleRequest(S3Event s3event, Context context) {

String instanceId = "your-ec2-instance-id";

AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StartInstancesRequest startRequest = new StartInstancesRequest().withInstanceIds(instanceId);

ec2.startInstances(startRequest);

System.out.println("EC2 instance started: " + instanceId);

}

}

```

### Step 3: Read the Document from S3, Analyze with Textract, and Send to SQS

On the EC2 instance, you can use the following code to read the document from S3, analyze it using Amazon Textract, and send the results to an SQS queue:

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.textract.AmazonTextract;

import com.amazonaws.services.textract.AmazonTextractClientBuilder;

import com.amazonaws.services.textract.model.Document;

import com.amazonaws.services.textract.model.S3Object;

import com.amazonaws.services.textract.model.AnalyzeDocumentRequest;

import com.amazonaws.services.textract.model.AnalyzeDocumentResult;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.SendMessageRequest;

public class TextractAnalyzer {

public static void main(String[] args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String bucketName = "your-bucket-name";

String keyName = "your-key-name";

String queueUrl = "your-sqs-queue-url";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

AmazonTextract textractClient = AmazonTextractClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

Document document = new Document().withS3Object(new S3Object().withBucket(bucketName).withName(keyName));

AnalyzeDocumentRequest request = new AnalyzeDocumentRequest().withDocument(document);

AnalyzeDocumentResult result = textractClient.analyzeDocument(request);

System.out.println("Textract analysis result: " + result);

AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

SendMessageRequest sendMsgRequest = new SendMessageRequest()

.withQueueUrl(queueUrl)

.withMessageBody(result.toString());

sqsClient.sendMessage(sendMsgRequest);

System.out.println("Message sent to SQS queue: " + queueUrl);

}

}

```

### Step 4: Consume the Message from SQS and Insert into RDS

Create a consumer that reads messages from the SQS queue and inserts them into an RDS table:

```java

import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.ReceiveMessageRequest;

import com.amazonaws.services.sqs.model.Message;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class SQSConsumer {

public static void main(String[] args) {

String accessKey = "your-access-key";

String secretKey = "your-secret-key";

String queueUrl = "your-sqs-queue-url";

String jdbcUrl = "your-rds-jdbc-url";

String dbUser = "your-db-username";

String dbPassword = "your-db-password";

BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new AWSStaticCredentialsProvider(awsCreds))

.build();

ReceiveMessageRequest receiveMsgRequest = new ReceiveMessageRequest(queueUrl);

for (Message message : sqsClient.receiveMessage(receiveMsgRequest).getMessages()) {

String messageBody = message.getBody();

System.out.println("Received message: " + messageBody);

try (Connection conn = DriverManager.getConnection(jdbcUrl, dbUser, dbPassword)) {

String sql = "INSERT INTO your\_table (column1, column2) VALUES (?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, messageBody); // Adjust as needed

pstmt.setString(2, "additional-data"); // Adjust as needed

pstmt.executeUpdate();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

```

### Summary

1. \*\*S3Uploader\*\*: Uploads a document to S3.

2. \*\*S3EventHandler\*\*: AWS Lambda function that triggers an EC2 instance when a new object is uploaded to S3.

3. \*\*TextractAnalyzer\*\*: EC2 instance reads the document from S3, analyzes it using Amazon Textract, and sends the results to an SQS queue.

4. \*\*SQSConsumer\*\*: Consumes the message from the SQS queue and inserts it into an RDS table.

This integrated example demonstrates how to use AWS services with Java to create a workflow that uploads a document to S3, triggers an EC2 instance to analyze the document with Textract, sends the results to an SQS queue, and finally inserts the data into an RDS table. Adjust the code as needed to fit your specific requirements and handle error cases appropriately.

Below is a comprehensive Java program that demonstrates the integration of various AWS services, including S3, EC2, SNS, SQS, Elastic Beanstalk, DynamoDB, and CloudWatch.

Prerequisites

AWS Account: Ensure you have an AWS account with necessary permissions.

AWS CLI: Install and configure the AWS CLI.

AWS SDK for Java: Include the AWS SDK for Java in your project.

Java IDE: Any Java IDE, such as IntelliJ IDEA or Eclipse.

IAM Roles and Policies: Create IAM roles with necessary permissions for using these services.

Maven POM File

Add the following dependencies to your pom.xml file:

xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <http://maven.apache.org/maven-v4_0_0.xsd>">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>aws-multi-service-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- AWS SDK Dependencies -->

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>s3</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>ec2</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>sns</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>sqs</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>beanstalk</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>dynamodb</artifactId>

<version>2.16.49</version>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>cloudwatch</artifactId>

<version>2.16.49</version>

</dependency>

</dependencies>

</project>

Java Program

Here is a Java program that demonstrates how to interact with these AWS services:

java

package com.example;

import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.s3.S3Client;

import software.amazon.awssdk.services.s3.model.PutObjectRequest;

import software.amazon.awssdk.services.ec2.Ec2Client;

import software.amazon.awssdk.services.ec2.model.RunInstancesRequest;

import software.amazon.awssdk.services.sns.SnsClient;

import software.amazon.awssdk.services.sns.model.PublishRequest;

import software.amazon.awssdk.services.sqs.SqsClient;

import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import software.amazon.awssdk.services.elasticbeanstalk.ElasticBeanstalkClient;

import software.amazon.awssdk.services.elasticbeanstalk.model.\*;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;

import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;

import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataRequest;

import software.amazon.awssdk.services.cloudwatch.model.MetricDatum;

import java.util.HashMap;

import java.util.Map;

public class AWSIntegrationDemo {

public static void main(String[] args) {

Region region = Region.US\_EAST\_1;

// Initialize clients

S3Client s3 = S3Client.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

Ec2Client ec2 = Ec2Client.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

SnsClient sns = SnsClient.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

SqsClient sqs = SqsClient.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

ElasticBeanstalkClient ebClient = ElasticBeanstalkClient.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

DynamoDbClient dynamoDb = DynamoDbClient.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

CloudWatchClient cloudWatch = CloudWatchClient.builder().region(region).credentialsProvider(EnvironmentVariableCredentialsProvider.create()).build();

// 1. S3 - Upload a file to S3

String bucketName = "your-bucket-name";

String objectKey = "sample.txt";

s3.putObject(PutObjectRequest.builder().bucket(bucketName).key(objectKey).build(), Paths.get("path/to/yourfile.txt"));

// 2. EC2 - Start an EC2 instance

RunInstancesRequest runInstancesRequest = RunInstancesRequest.builder()

.imageId("ami-xxxxxxxx")

.instanceType("t2.micro")

.maxCount(1)

.minCount(1)

.build();

ec2.runInstances(runInstancesRequest);

// 3. SNS - Publish a message to an SNS topic

String topicArn = "arn:aws:sns:us-east-1:123456789012:your-topic";

sns.publish(PublishRequest.builder()

.topicArn(topicArn)

.message("Sample message")

.build());

// 4. SQS - Send a message to an SQS queue

String queueUrl = "<https://sqs.us-east-1.amazonaws.com/123456789012/your-queue>";

sqs.sendMessage(SendMessageRequest.builder()

.queueUrl(queueUrl)

.messageBody("Sample message")

.build());

// 5. Elastic Beanstalk - Create an application

String applicationName = "SampleApp";

CreateApplicationRequest createApplicationRequest = CreateApplicationRequest.builder()

.applicationName(applicationName)

.build();

ebClient.createApplication(createApplicationRequest);

// 6. DynamoDB - Insert an item into a DynamoDB table

String tableName = "SampleTable";

Map<String, AttributeValue> item = new HashMap<>();

item.put("Id", AttributeValue.builder().s("001").build());

item.put("Name", AttributeValue.builder().s("Sample Name").build());

dynamoDb.putItem(PutItemRequest.builder()

.tableName(tableName)

.item(item)

.build());

// 7. CloudWatch - Publish a custom metric

String namespace = "SampleNamespace";

cloudWatch.putMetricData(PutMetricDataRequest.builder()

.namespace(namespace)

.metricData(MetricDatum.builder()

.metricName("SampleMetric")

.value(1.0)

.build())

.build());

}

}

Explanation

S3: Uploads a file to the specified S3 bucket.

EC2: Starts a new EC2 instance using a specified AMI.

SNS: Publishes a message to an SNS topic.

SQS: Sends a message to an SQS queue.

Elastic Beanstalk: Creates a new application in Elastic Beanstalk.

DynamoDB: Inserts an item into a DynamoDB table.

CloudWatch: Publishes a custom metric to CloudWatch.

Additional Configuration

Environment Variables:

Set AWS credentials and region via environment variables or configure them in the AWS CLI.

IAM Roles and Policies:

Ensure the necessary IAM roles and policies are in place to allow access to each of these services.

Running the Program

Compile and package the application:

mvn clean package

Run the application:

java -jar target/aws-multi-service-demo-1.0-SNAPSHOT.jar

Conclusion

This program demonstrates how to interact with various AWS services using the AWS SDK for Java. It provides a solid foundation for building more complex workflows and automations involving multiple AWS services.

Feel free to modify the example to fit your specific requirements. Let me know if you need further assistance!

1.Read a document from S3

2.Using textract analyze the doc and print details

3.Then send it to a queue using SQS

3.Once it is in Queue using SNS send a notification and alert

4. Once It is successful insert the alert in RDS

5. Then read data from queue and push it into another s3 bucket

6.add code to use cloudwatch to send alerts and metrics in between

<dependencies>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-s3</artifactId>

<version>1.11.1000</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-textract</artifactId>

<version>1.11.1000</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-sqs</artifactId>

<version>1.11.1000</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-sns</artifactId>

<version>1.11.1000</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-rds</artifactId>

<version>1.11.1000</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-cloudwatch</artifactId>

<version>1.11.1000</version>

</dependency>

<!-- Other dependencies such as JDBC or any specific database driver can be added here -->

</dependencies>

package com.example.lambda;

import com.amazonaws.services.lambda.runtime.Context;

import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.amazonaws.services.lambda.runtime.events.S3Event;

import com.amazonaws.services.rds.AmazonRDS;

import com.amazonaws.services.rds.AmazonRDSClientBuilder;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.GetObjectRequest;

import com.amazonaws.services.s3.model.S3Object;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.SendMessageRequest;

import com.amazonaws.services.sns.AmazonSNS;

import com.amazonaws.services.sns.AmazonSNSClientBuilder;

import com.amazonaws.services.sns.model.PublishRequest;

import com.amazonaws.services.textract.AmazonTextract;

import com.amazonaws.services.textract.AmazonTextractClientBuilder;

import com.amazonaws.services.textract.model.AnalyzeDocumentRequest;

import com.amazonaws.services.textract.model.AnalyzeDocumentResult;

import com.amazonaws.services.textract.model.Document;

import com.amazonaws.services.cloudwatch.AmazonCloudWatch;

import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;

import com.amazonaws.services.cloudwatch.model.Dimension;

import com.amazonaws.services.cloudwatch.model.MetricDatum;

import com.amazonaws.services.cloudwatch.model.PutMetricDataRequest;

import com.amazonaws.services.cloudwatch.model.StandardUnit;

import java.io.InputStream;

import java.nio.ByteBuffer;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.util.Scanner;

public class DocumentProcessorLambda implements RequestHandler<S3Event, String> {

private static final String DB\_URL = "jdbc:mysql://your-database-endpoint:port/dbname";

private static final String DB\_USER = "your-db-username";

private static final String DB\_PASSWORD = "your-db-password";

private static Connection getConnection() throws Exception {

return DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

}

private static void publishMetric(String metricName, double value, String dimensionName, String dimensionValue) {

AmazonCloudWatch cloudWatch = AmazonCloudWatchClientBuilder.defaultClient();

Dimension dimension = new Dimension().withName(dimensionName).withValue(dimensionValue);

MetricDatum datum = new MetricDatum().withMetricName(metricName).withUnit(StandardUnit.None).withValue(value).withDimensions(dimension);

PutMetricDataRequest request = new PutMetricDataRequest().withNamespace("DocumentProcessor").withMetricData(datum);

cloudWatch.putMetricData(request);

}

@Override

public String handleRequest(S3Event s3Event, Context context) {

String bucketName = s3Event.getRecords().get(0).getS3().getBucket().getName();

String documentKey = s3Event.getRecords().get(0).getS3().getObject().getKey();

AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

S3Object s3Object = s3Client.getObject(new GetObjectRequest(bucketName, documentKey));

try {

// Read the document from S3

InputStream objectData = s3Object.getObjectContent();

context.getLogger().log("Document Content from S3:");

Scanner scanner = new Scanner(objectData);

while (scanner.hasNextLine()) {

context.getLogger().log(scanner.nextLine());

}

publishMetric("DocumentReadFromS3", 1, "DocumentKey", documentKey);

// Extract text from the document using Textract

AmazonTextract textractClient = AmazonTextractClientBuilder.defaultClient();

AnalyzeDocumentRequest request = new AnalyzeDocumentRequest()

.withDocument(new Document()

.withBytes(ByteBuffer.wrap(s3Object.getObjectContent().readAllBytes())))

.withFeatureTypes("TABLES", "FORMS");

AnalyzeDocumentResult result = textractClient.analyzeDocument(request);

// Print details

context.getLogger().log("Textract Result: " + result.toString());

publishMetric("DocumentProcessedByTextract", 1, "DocumentKey", documentKey);

// Send the details to SQS

AmazonSQS sqsClient = AmazonSQSClientBuilder.defaultClient();

SendMessageRequest sendMessageRequest = new SendMessageRequest()

.withQueueUrl("your-queue-url")

.withMessageBody(result.toString());

sqsClient.sendMessage(sendMessageRequest);

publishMetric("MessageSentToSQS", 1, "DocumentKey", documentKey);

// Send a notification via SNS

AmazonSNS snsClient = AmazonSNSClientBuilder.defaultClient();

PublishRequest publishRequest = new PublishRequest()

.withTopicArn("your-sns-topic-arn")

.withMessage("Document processed and details sent to SQS. Document Key: " + documentKey);

snsClient.publish(publishRequest);

publishMetric("NotificationSentBySNS", 1, "DocumentKey", documentKey);

// Insert alert in RDS

String alertMessage = "Document processed and details sent to SQS. Document Key: " + documentKey;

try (Connection connection = getConnection()) {

String insertQuery = "INSERT INTO alerts (message) VALUES (?)";

try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {

preparedStatement.setString(1, alertMessage);

preparedStatement.executeUpdate();

}

}

publishMetric("AlertInsertedInRDS", 1, "DocumentKey", documentKey);

// Read data from SQS and push into another S3 bucket

String queueUrl = "your-queue-url";

String message = sqsClient.receiveMessage(queueUrl).getMessages().get(0).getBody();

s3Client.putObject("your-target-bucket-name", "processed/" + documentKey, message);

publishMetric("DataPushedToS3", 1, "DocumentKey", documentKey);

} catch (Exception e) {

context.getLogger().log("Exception: " + e.getMessage());

}

return "Process Completed!";

}

}

This program demonstrates how to use Amazon S3, Amazon DynamoDB, and Amazon SQS services.

import com.amazonaws.AmazonServiceException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;

import com.amazonaws.services.dynamodbv2.document.Item;

import com.amazonaws.services.dynamodbv2.document.Table;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.S3Object;

import com.amazonaws.services.s3.model.PutObjectRequest;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.Message;

import com.amazonaws.services.sqs.model.ReceiveMessageRequest;

import com.amazonaws.services.sqs.model.SendMessageRequest;

import java.io.File;

public class AWSServicesExample {

private static final String BUCKET\_NAME = "your-bucket-name";

private static final String FILE\_PATH = "path/to/your/file.txt";

private static final String DYNAMODB\_TABLE\_NAME = "your-table-name";

private static final String SQS\_QUEUE\_URL = "your-queue-url";

public static void main(String[] args) {

uploadToS3();

insertIntoDynamoDB();

sendMessageToSQS();

receiveMessagesFromSQS();

}

private static void uploadToS3() {

try {

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

s3Client.putObject(new PutObjectRequest(BUCKET\_NAME, new File(FILE\_PATH).getName(), new File(FILE\_PATH)));

System.out.println("File uploaded to S3 successfully.");

} catch (AmazonServiceException e) {

System.err.println(e.getErrorMessage());

}

}

private static void insertIntoDynamoDB() {

AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable(DYNAMODB\_TABLE\_NAME);

try {

Item item = new Item()

.withPrimaryKey("ID", 1)

.withString("Name", "Example Name")

.withNumber("Age", 25);

table.putItem(item);

System.out.println("Item inserted into DynamoDB successfully.");

} catch (Exception e) {

System.err.println("Unable to add item: " + e.getMessage());

}

}

private static void sendMessageToSQS() {

AmazonSQS sqs = AmazonSQSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

SendMessageRequest send\_msg\_request = new SendMessageRequest()

.withQueueUrl(SQS\_QUEUE\_URL)

.withMessageBody("Hello from SQS!");

sqs.sendMessage(send\_msg\_request);

System.out.println("Message sent to SQS successfully.");

}

private static void receiveMessagesFromSQS() {

AmazonSQS sqs = AmazonSQSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

ReceiveMessageRequest receiveRequest = new ReceiveMessageRequest()

.withQueueUrl(SQS\_QUEUE\_URL)

.withMaxNumberOfMessages(10);

for (Message message : sqs.receiveMessage(receiveRequest).getMessages()) {

System.out.println("Message received: " + message.getBody());

}

}

}

import com.amazonaws.AmazonServiceException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;

import com.amazonaws.services.cloudwatch.AmazonCloudWatch;

import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;

import com.amazonaws.services.cloudwatch.model.PutLogEventsRequest;

import com.amazonaws.services.cloudwatch.model.PutLogEventsResult;

import com.amazonaws.services.cloudwatch.model.InputLogEvent;

import com.amazonaws.services.cloudwatch.model.CreateLogStreamRequest;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;

import com.amazonaws.services.dynamodbv2.document.Item;

import com.amazonaws.services.dynamodbv2.document.Table;

import com.amazonaws.services.ec2.model.VolumeType;

import com.amazonaws.services.ecs.AmazonECS;

import com.amazonaws.services.ecs.AmazonECSClientBuilder;

import com.amazonaws.services.ecs.model.ListClustersRequest;

import com.amazonaws.services.ecs.model.ListClustersResult;

import com.amazonaws.services.ebs.AmazonElasticBlockStore;

import com.amazonaws.services.ebs.AmazonElasticBlockStoreClientBuilder;

import com.amazonaws.services.ebs.model.CreateVolumeRequest;

import com.amazonaws.services.glue.AWSGlue;

import com.amazonaws.services.glue.AWSGlueClientBuilder;

import com.amazonaws.services.glue.model.StartJobRunRequest;

import com.amazonaws.services.glue.model.StartJobRunResult;

import com.amazonaws.services.kendra.AmazonKendra;

import com.amazonaws.services.kendra.AmazonKendraClientBuilder;

import com.amazonaws.services.kendra.model.QueryRequest;

import com.amazonaws.services.kendra.model.QueryResult;

import com.amazonaws.services.kinesis.AmazonKinesis;

import com.amazonaws.services.kinesis.AmazonKinesisClientBuilder;

import com.amazonaws.services.kinesis.model.PutRecordsRequest;

import com.amazonaws.services.kinesis.model.PutRecordsRequestEntry;

import com.amazonaws.services.kinesis.model.PutRecordsResult;

import com.amazonaws.services.logs.AWSLogs;

import com.amazonaws.services.logs.AWSLogsClientBuilder;

import com.amazonaws.services.logs.model.CreateLogGroupRequest;

import com.amazonaws.services.logs.model.CreateLogStreamRequest;

import com.amazonaws.services.logs.model.InputLogEvent;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.PutObjectRequest;

import com.amazonaws.services.s3.model.S3Object;

import com.amazonaws.services.sns.AmazonSNS;

import com.amazonaws.services.sns.AmazonSNSClientBuilder;

import com.amazonaws.services.sns.model.PublishRequest;

import com.amazonaws.services.sns.model.PublishResult;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClientBuilder;

import com.amazonaws.services.sqs.model.Message;

import com.amazonaws.services.sqs.model.ReceiveMessageRequest;

import com.amazonaws.services.sqs.model.SendMessageRequest;

import com.amazonaws.services.textract.AmazonTextract;

import com.amazonaws.services.textract.AmazonTextractClientBuilder;

import com.amazonaws.services.textract.model.Document;

import com.amazonaws.services.textract.model.S3Object;

import com.amazonaws.services.textract.model.DetectDocumentTextRequest;

import com.amazonaws.services.textract.model.DetectDocumentTextResult;

import com.amazonaws.services.cloudfront.AmazonCloudFront;

import com.amazonaws.services.cloudfront.AmazonCloudFrontClientBuilder;

import com.amazonaws.services.cloudfront.model.CreateDistributionRequest;

import com.amazonaws.services.cloudfront.model.CreateDistributionResult;

import com.amazonaws.services.elasticbeanstalk.AWSElasticBeanstalk;

import com.amazonaws.services.elasticbeanstalk.AWSElasticBeanstalkClientBuilder;

import com.amazonaws.services.elasticbeanstalk.model.CreateApplicationRequest;

import com.amazonaws.services.elasticbeanstalk.model.CreateApplicationResult;

import software.amazon.awssdk.services.iot.model.PublishRequest;

import software.amazon.awssdk.services.iot.AwsIot;

import software.amazon.awssdk.services.iot.AwsIotClient;

import java.io.File;

import java.nio.ByteBuffer;

import java.time.Instant;

import java.util.ArrayList;

import java.util.List;

public class ExtendedAWSServicesExample {

private static final String BUCKET\_NAME = "your-bucket-name";

private static final String FILE\_PATH = "path/to/your/file.txt";

private static final String DYNAMODB\_TABLE\_NAME = "your-table-name";

private static final String SQS\_QUEUE\_URL = "your-queue-url";

private static final String LOG\_GROUP\_NAME = "your-log-group-name";

private static final String LOG\_STREAM\_NAME = "your-log-stream-name";

private static final String KENDRA\_INDEX\_ID = "your-kendra-index-id";

private static final String ECS\_CLUSTER\_NAME = "your-ecs-cluster-name";

private static final String EBS\_VOLUME\_ID = "your-ebs-volume-id";

private static final String IOT\_ENDPOINT = "your-iot-endpoint";

private static final String KINESIS\_STREAM\_NAME = "your-kinesis-stream-name";

private static final String GLUE\_JOB\_NAME = "your-glue-job-name";

private static final String SNS\_TOPIC\_ARN = "your-sns-topic-arn";

private static final String CLOUD\_FRONT\_DISTRIBUTION\_ID = "your-cloud-front-distribution-id";

private static final String EBS\_APPLICATION\_NAME = "your-ebs-application-name";

private static final String EKS\_CLUSTER\_NAME = "your-eks-cluster-name";

public static void main(String[] args) {

uploadToS3();

insertIntoDynamoDB();

sendMessageToSQS();

receiveMessagesFromSQS();

logToCloudWatch();

extractTextUsingTextract();

listECSClusters();

queryKendra();

createAndAttachEBSVolume();

performEdgeComputingOperation();

sendAlertUsingSNS();

streamDataUsingKinesis();

createCloudFrontDistribution();

runGlueJob();

createEKSCluster();

createElasticBeanstalkApplication();

}

private static void uploadToS3() {

try {

AmazonS3 s3Client = AmazonS3ClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

s3Client.putObject(new PutObjectRequest(BUCKET\_NAME, new File(FILE\_PATH).getName(), new File(FILE\_PATH)));

System.out.println("File uploaded to S3 successfully.");

} catch (AmazonServiceException e) {

System.err.println(e.getErrorMessage());

}

}

private static void insertIntoDynamoDB() {

AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable(DYNAMODB\_TABLE\_NAME);

try {

Item item = new Item()

.withPrimaryKey("ID", 1)

.withString("Name", "Example Name")

.withNumber("Age", 25);

table.putItem(item);

System.out.println("Item inserted into DynamoDB successfully.");

} catch (Exception e) {

System.err.println("Unable to add item: " + e.getMessage());

}

}

private static void sendMessageToSQS() {

AmazonSQS sqs = AmazonSQSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

SendMessageRequest send\_msg\_request = new SendMessageRequest()

.withQueueUrl(SQS\_QUEUE\_URL)

.withMessageBody("Hello from SQS!");

sqs.sendMessage(send\_msg\_request);

System.out.println("Message sent to SQS successfully.");

}

private static void receiveMessagesFromSQS() {

AmazonSQS sqs = AmazonSQSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

ReceiveMessageRequest receiveRequest = new ReceiveMessageRequest()

.withQueueUrl(SQS\_QUEUE\_URL)

.withMaxNumberOfMessages(10);

for (Message message : sqs.receiveMessage(receiveRequest).getMessages()) {

System.out.println("Message received: " + message.getBody());

}

}

private static void logToCloudWatch() {

AmazonCloudWatch cloudWatch = AmazonCloudWatchClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

CreateLogStreamRequest request = new CreateLogStreamRequest()

.withLogGroupName(LOG\_GROUP\_NAME)

.withLogStreamName(LOG\_STREAM\_NAME);

cloudWatch.createLogStream(request);

InputLogEvent logEvent = new InputLogEvent()

.withMessage("This is a sample log message")

.withTimestamp(Instant.now().toEpochMilli());

List<InputLogEvent> logEvents = new ArrayList<>();

logEvents.add(logEvent);

PutLogEventsRequest logEventsRequest = new PutLogEventsRequest()

.withLogGroupName(LOG\_GROUP\_NAME)

.withLogStreamName(LOG\_STREAM\_NAME)

.withLogEvents(logEvents);

PutLogEventsResult result = cloudWatch.putLogEvents(logEventsRequest);

System.out.println("Log event uploaded to CloudWatch successfully.");

}

private static void extractTextUsingTextract() {

AmazonTextract textractClient = AmazonTextractClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

Document document = new Document()

.withS3Object(new S3Object()

.withBucket(BUCKET\_NAME)

.withName("input-document.png"));

DetectDocumentTextRequest request = new DetectDocumentTextRequest()

.withDocument(document);

DetectDocumentTextResult result = textractClient.detectDocumentText(request);

System.out.println("Detected text:");

result.getBlocks().forEach(block -> {

if (block.getBlockType().equals("LINE")) {

System.out.println(block.getText());

}

});

}

private static void listECSClusters() {

AmazonECS ecsClient = AmazonECSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

ListClustersRequest request = new ListClustersRequest();

ListClustersResult result = ecsClient.listClusters(request);

result.getClusterArns().forEach(clusterArn -> {

System.out.println("Cluster: " + clusterArn);

});

}

private static void queryKendra() {

AmazonKendra kendraClient = AmazonKendraClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

QueryRequest request = new QueryRequest()

.withIndexId(KENDRA\_INDEX\_ID)

.withQueryText("What is Amazon Kendra?");

QueryResult result = kendraClient.query(request);

result.getResultItems().forEach(item -> {

System.out.println("Kendra Query Result: " + item.getDocumentTitle().getString());

});

}

private static void createAndAttachEBSVolume() {

AmazonElasticBlockStore ebsClient = AmazonElasticBlockStoreClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

CreateVolumeRequest request = new CreateVolumeRequest()

.withAvailabilityZone("us-west-2b")

.withSize(10)

.withVolumeType(VolumeType.Gp2);

ebsClient.createVolume(request);

System.out.println("EBS Volume created successfully.");

}

private static void performEdgeComputingOperation() {

// Edge computing simulation using AWS IoT

AwsIot iotClient = AwsIotClient.builder()

.endpointOverride(URI.create(IOT\_ENDPOINT))

.credentialsProvider(ProfileCredentialsProvider.create())

.build();

PublishRequest request = PublishRequest.builder()

.topic("iot/edge/computing")

.payload(ByteBuffer.wrap("Edge computing payload".getBytes(StandardCharsets.UTF\_8)))

.build();

iotClient.publish(request);

System.out.println("Edge computing operation performed successfully.");

}

private static void sendAlertUsingSNS() {

AmazonSNS snsClient = AmazonSNSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

PublishRequest publishRequest = new PublishRequest()

.withTopicArn(SNS\_TOPIC\_ARN)

.withMessage("Alert from SNS!");

PublishResult publishResult = snsClient.publish(publishRequest);

System.out.println("Message sent to SNS: " + publishResult.getMessageId());

}

private static void streamDataUsingKinesis() {

AmazonKinesis kinesisClient = AmazonKinesisClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

PutRecordsRequestEntry entry = new PutRecordsRequestEntry()

.withData(ByteBuffer.wrap("Some data to stream".getBytes(StandardCharsets.UTF\_8)))

.withPartitionKey("partitionKey");

List<PutRecordsRequestEntry> records = new ArrayList<>();

records.add(entry);

PutRecordsRequest request = new PutRecordsRequest()

.withStreamName(KINESIS\_STREAM\_NAME)

.withRecords(records);

PutRecordsResult result = kinesisClient.putRecords(request);

System.out.println("Data streamed to Kinesis: " + result.getRecords());

}

private static void createCloudFrontDistribution() {

AmazonCloudFront cloudFrontClient = AmazonCloudFrontClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

// Create a new CloudFront Distribution (using dummy data for illustration)

CreateDistributionRequest request = new CreateDistributionRequest()

.withDistributionConfig(new DistributionConfig()

.withCallerReference("cloudfront-distribution")

.withEnabled(true));

CreateDistributionResult result = cloudFrontClient.createDistribution(request);

System.out.println("CloudFront distribution created: " + result.getDistribution().getId());

}

private static void runGlueJob() {

AWSGlue glueClient = AWSGlueClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

StartJobRunRequest request = new StartJobRunRequest()

.withJobName(GLUE\_JOB\_NAME);

StartJobRunResult result = glueClient.startJobRun(request);

System.out.println("Glue job started: " + result.getJobRunId());

}

private static void createEKSCluster() {

AmazonEKS eksClient = AmazonEKSClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

CreateClusterRequest request = new CreateClusterRequest()

.withName(EKS\_CLUSTER\_NAME)

.withVersion("1.15");

CreateClusterResult result = eksClient.createCluster(request);

System.out.println("EKS cluster created: " + result.getCluster().getName());

}

private static void createElasticBeanstalkApplication() {

AWSElasticBeanstalk elasticBeanstalkClient = AWSElasticBeanstalkClientBuilder.standard()

.withCredentials(new ProfileCredentialsProvider())

.build();

CreateApplicationRequest request = new CreateApplicationRequest()

.withApplicationName(EBS\_APPLICATION\_NAME);

CreateApplicationResult result = elasticBeanstalkClient.createApplication(request);

System.out.println("Elastic Beanstalk application created: " + result.getApplication().getApplicationName());

}

}