AWS Lambda Functions with Java

Dr Mohammed Kaleem

Overview

Implementing AWS Lambda functions in Java:

- Java lambda functions workflow
- Maven dependency management
- Packaging and deploying Java lambda functions to AWS

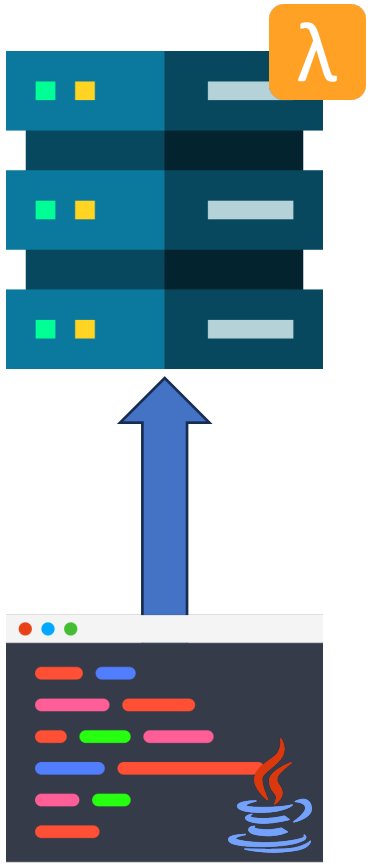
Recap

AWS Lambda Basics

- Executes code in response to events.
  - Several “triggers available” (REST API, Alexa commands, File Uploads etc).
- Key selling point: No need to provision or manage servers.
- Excellent PaaS solution for implementing REST APIs (micro services).

Lambda supports Java

- You can code your lambda functions in Java and deploy them to AWS lambda.
- Java is not supported as an in-browser language like python.
- You can use eclipse (or any IDE) to implement your function, package it as a JAR file and deploy it to lambda.



AWS Lambda Required Java Libraries

When implementing an AWS Lambda function in Java, you **need to include the AWS Lambda runtime library and any additional libraries or dependencies** required for your specific function.

The **AWS Lambda runtime library includes the necessary classes and interfaces for handling Lambda events** and interacting with AWS services.

Dependency Management with Maven

To manage the required dependencies, we will use Maven.

- Maven simplifies the build process by providing a consistent and standardized approach.
- Simply add required dependencies to the POM file.
- Maven will retrieve all required dependencies from the central maven repository.

Why Maven?

- Simplifies project configuration and management.
- Manages project dependencies automatically.
- Provides a standard project structure.
- Promotes best practices for project organization.
- There are alternatives to Maven, but all dependency management tools (regardless of language) have similar aims and objectives.

Key Maven Terminology

- **Project**: The software being built.
- **POM** (Project Object Model): An XML file describing the project configuration.
- **Artifact**: A packaged and versioned project output (e.g., JAR, WAR).

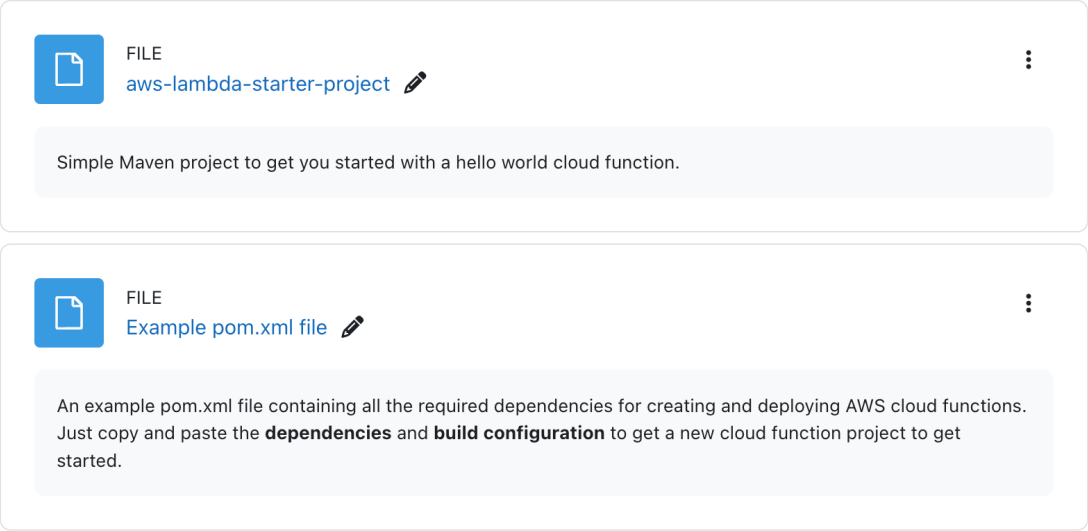
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>uk.ac.mmu</groupId>
   <artifactId>demo-aws-test</artifactId>
   <version>0.0.1-SNAPSHOT</version>
      <dependencies>
      <!-- AWS Lambda -->
          <dependency>
             <groupId>com.amazonaws</groupId>
             <artifactId>aws-lambda-java-core</artifactId>
             <version>1.2.0</version>
          </dependency>
          <dependency>
              <groupId>com.amazonaws</groupId>
              <artifactId>aws-lambda-java-events</artifactId>
              <version>2.2.7</version>
          </dependency>
          <dependency>
              <groupId>com.amazonaws</groupId>
              <artifactId>aws-lambda-java-log4j</artifactId>
              <version>1.0.0</version>
          </dependency>
          <dependency>
              <groupId>com.amazonaws</groupId>
              <artifactId>aws-lambda-java-log4j2</artifactId>
              <version>1.0.0</version>
          </dependency>
       </dependencies>
</project>
```

AWS Lambda Templates on Moodle

There is a template Maven project AWS lambda project that can be imported into eclipse as a starting point.

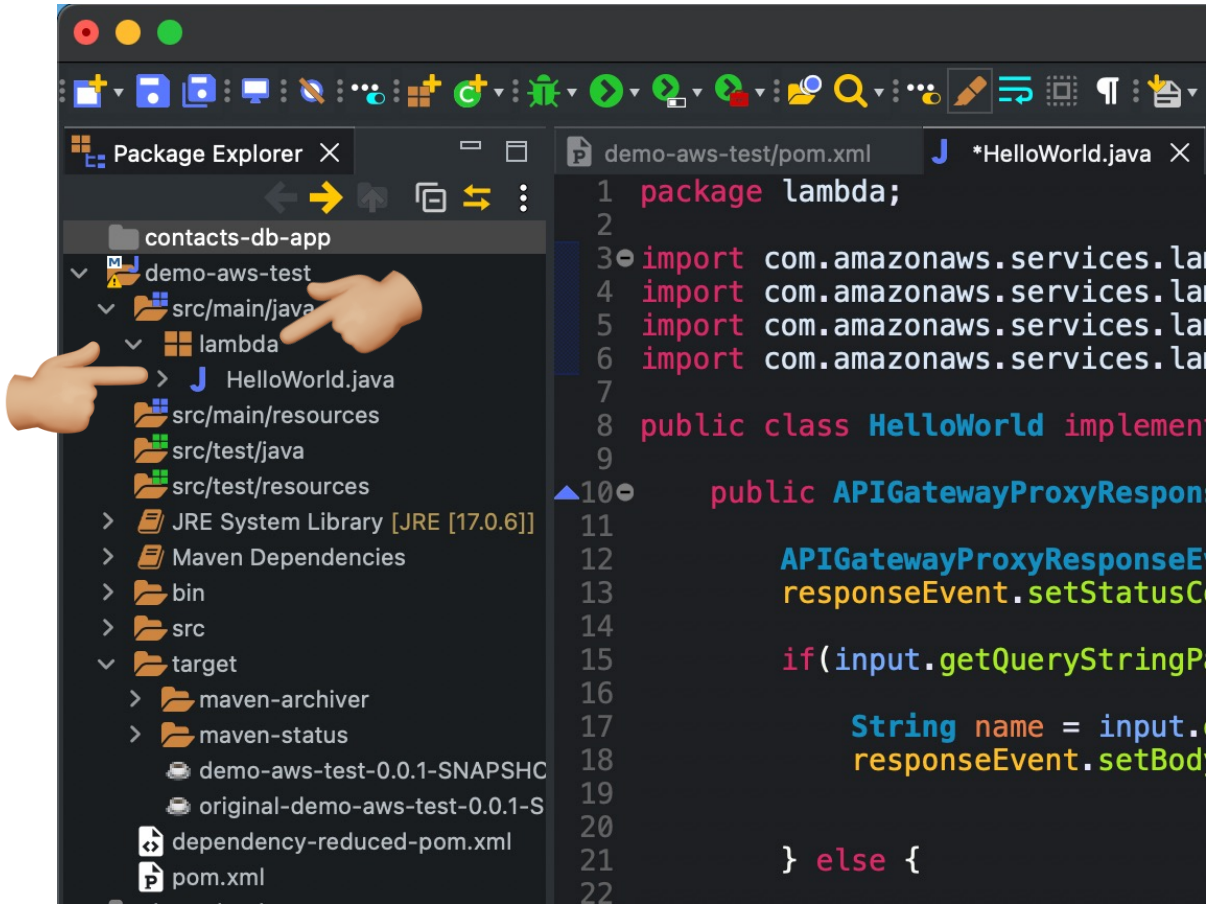
OR

You can create a new Maven project in eclipse and import the necessary dependencies using the POM.xml file. Example POM.xml also on Moodle.



Implementing the lambda function

1. Setup the project
2. Create a new package
3. Create new Java Class



Code up your function

```java
package lambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;

public class HelloWorld implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>{

    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent input, Context context) {

        APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();
        responseEvent.setStatusCode(200);

        if(input.getQueryStringParameters().containsKey("name")) {

            String name = input.getQueryStringParameters().get("name");
            responseEvent.setBody(name +  " is testing AWS lambda functions");


        } else {

            responseEvent.setBody("name parameter is required!! ");

        }


        return responseEvent;
    }

}
```

Input object contains all data related to the incoming request

Code up your function – continued

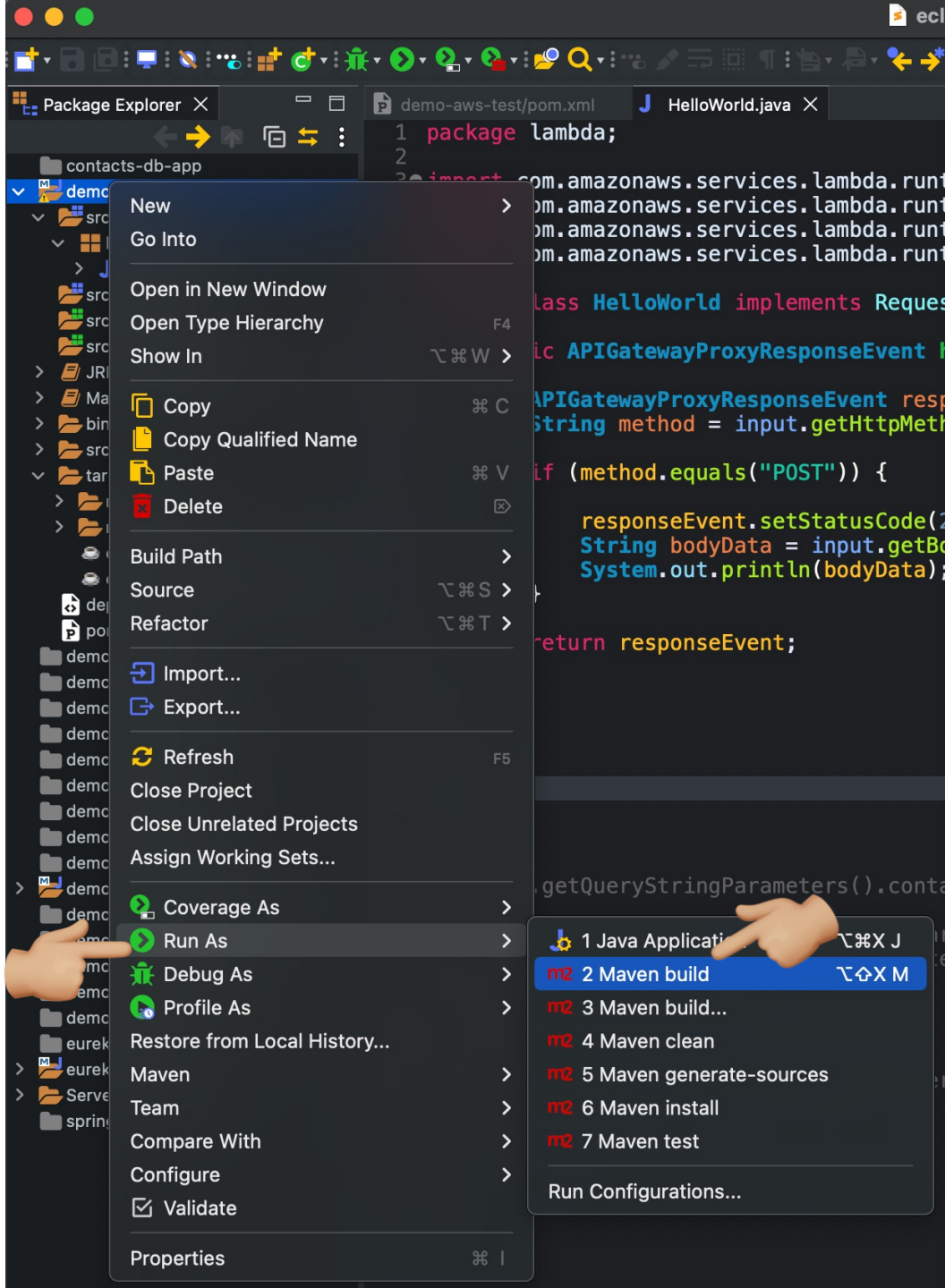
demo-aws-test/pom.xml | HelloWorld.java

```java
package lambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;

public class HelloWorld implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>{

    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent input, Context context) {

        APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();
        String method = input.getHttpMethod();

        if (method.equals("POST")) {

            responseEvent.setStatusCode(200);
            String bodyData = input.getBody();
            System.out.println(bodyData);
        }

        return responseEvent;
    }

}
```

You can also access the body data from the input object as well as get the HTTP method type.

Package and Deploy

1. Right click the project
2. Select “Run As” -> Maven Build

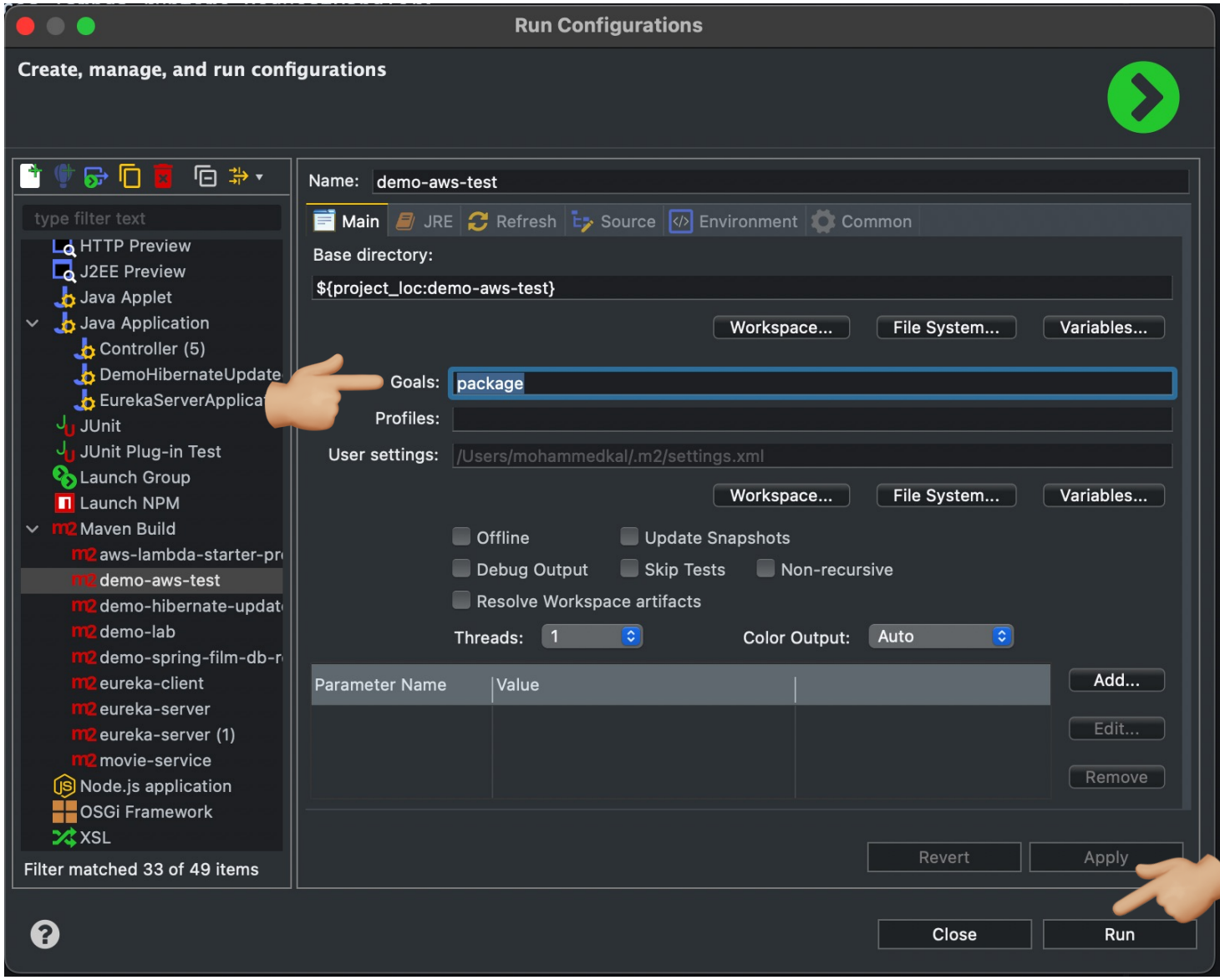


Run Configurations

The first time you run a project a maven build you have to setup the run configurations.

1. In the Goals option enter “package”
2. Click “Run”

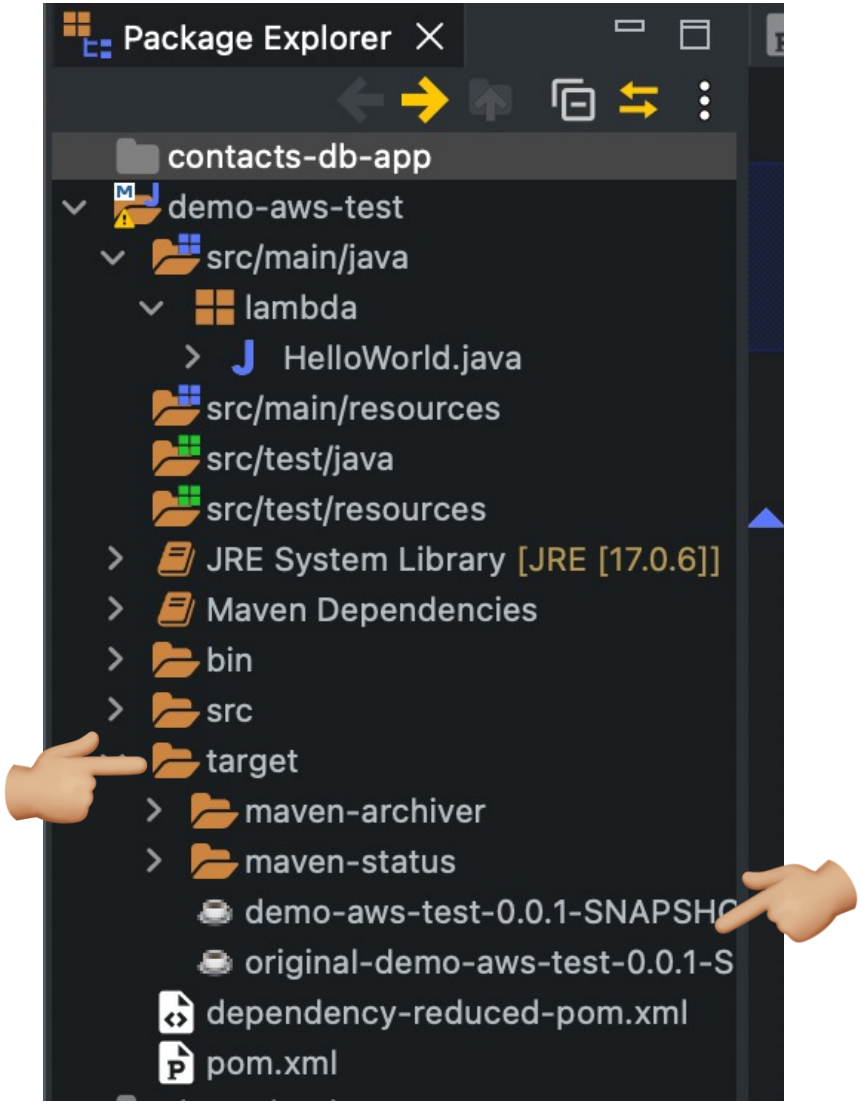
3. You will see the build progress and output in the console window.



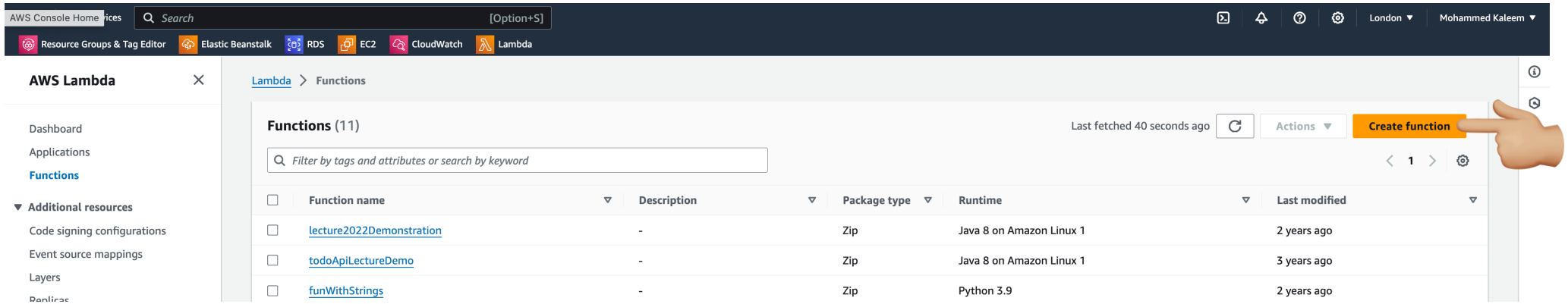
Generated JAR File

Once the build has finished you will find the generated JAR files in the Target folder (you may need to refresh the project the to see the files).

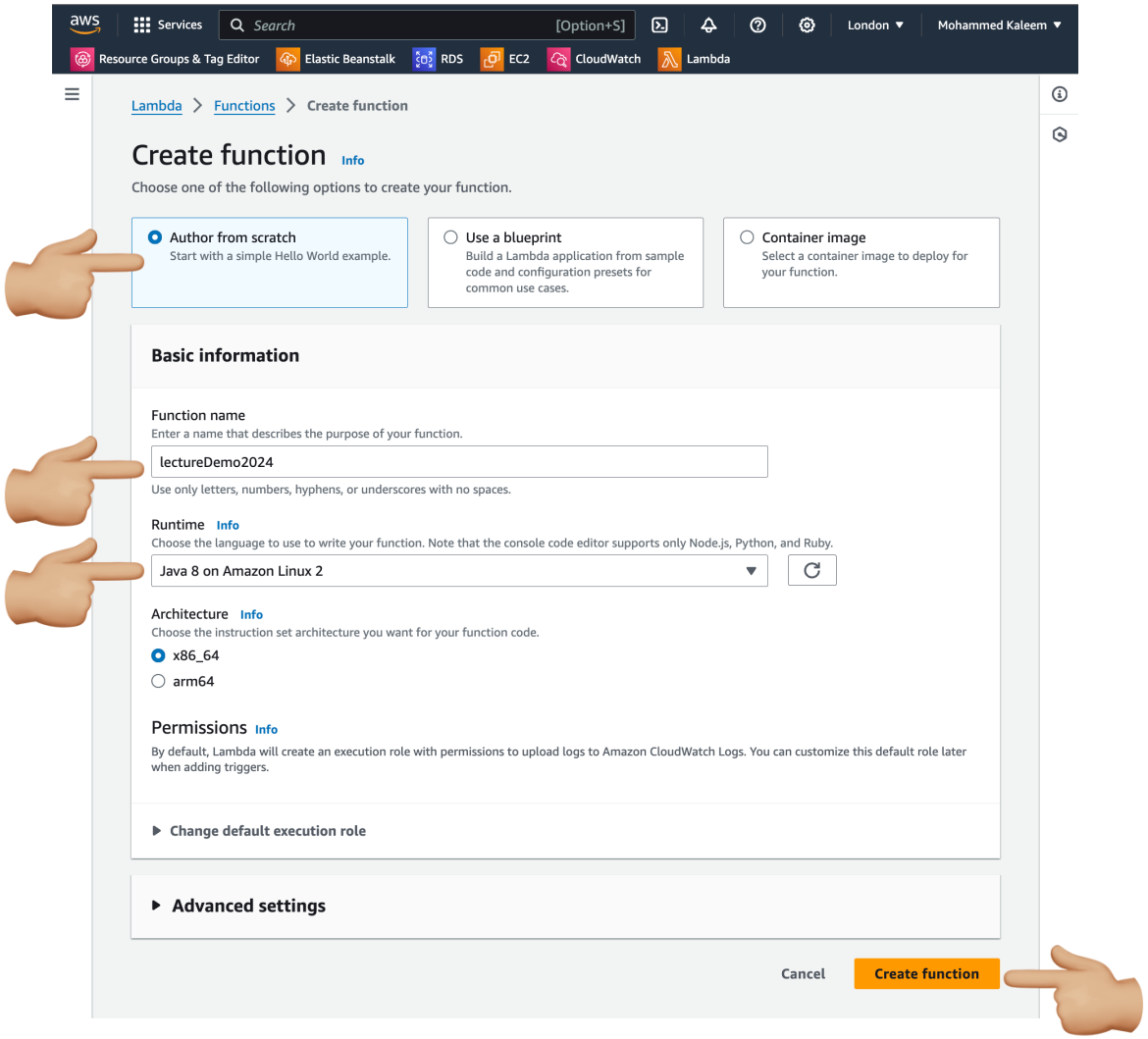
This is the JAR file the we need to deploy the function to AWS Lambda.



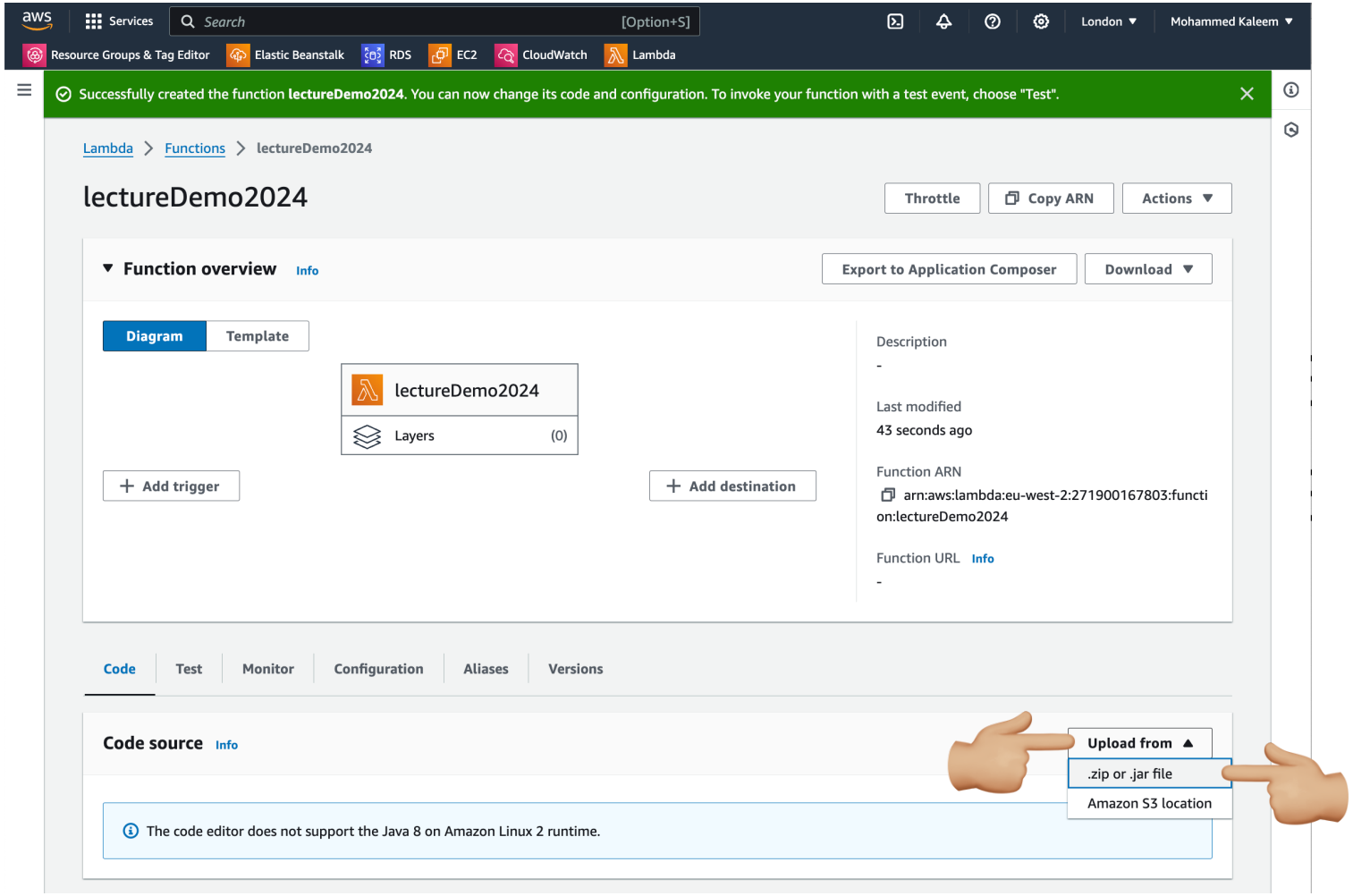
New Lambda Function



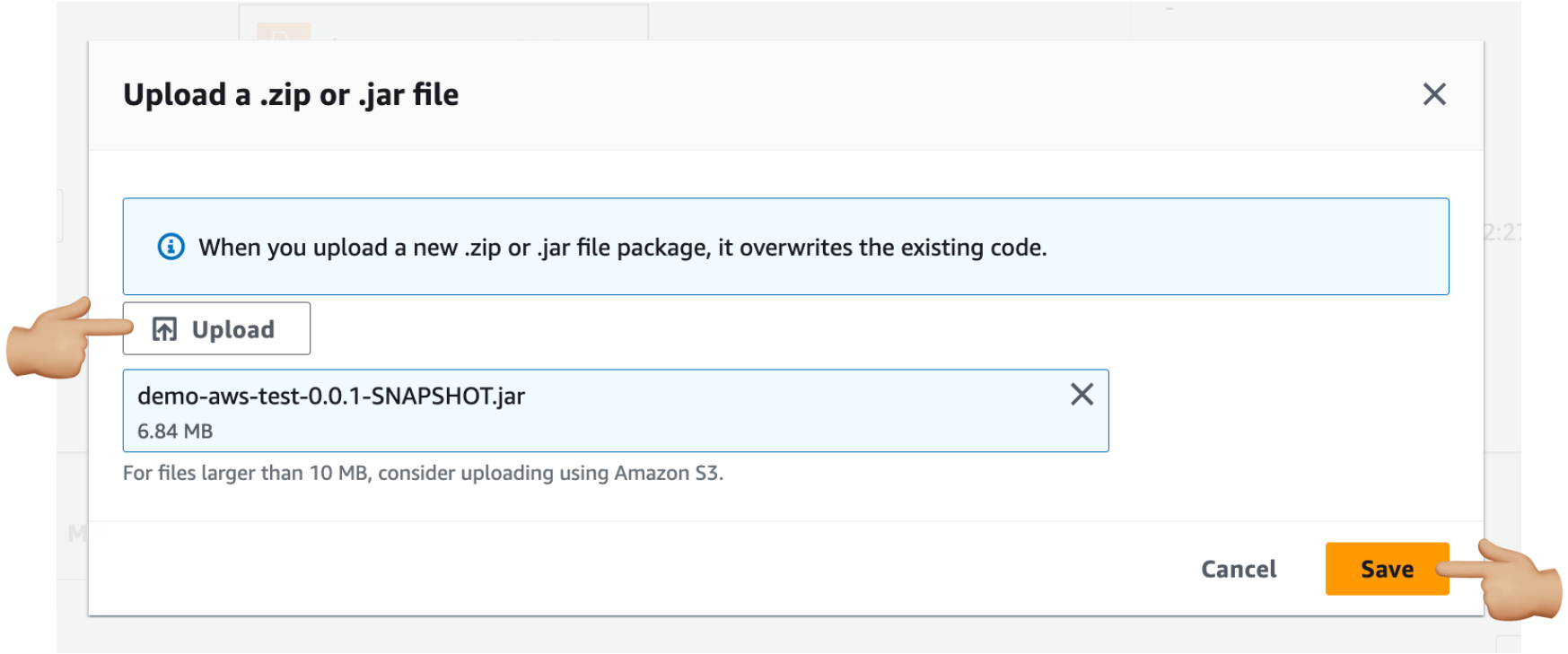
Configure Lambda Function



Upload your code (JAR file)



Upload your code (JAR file)



Runtime settings

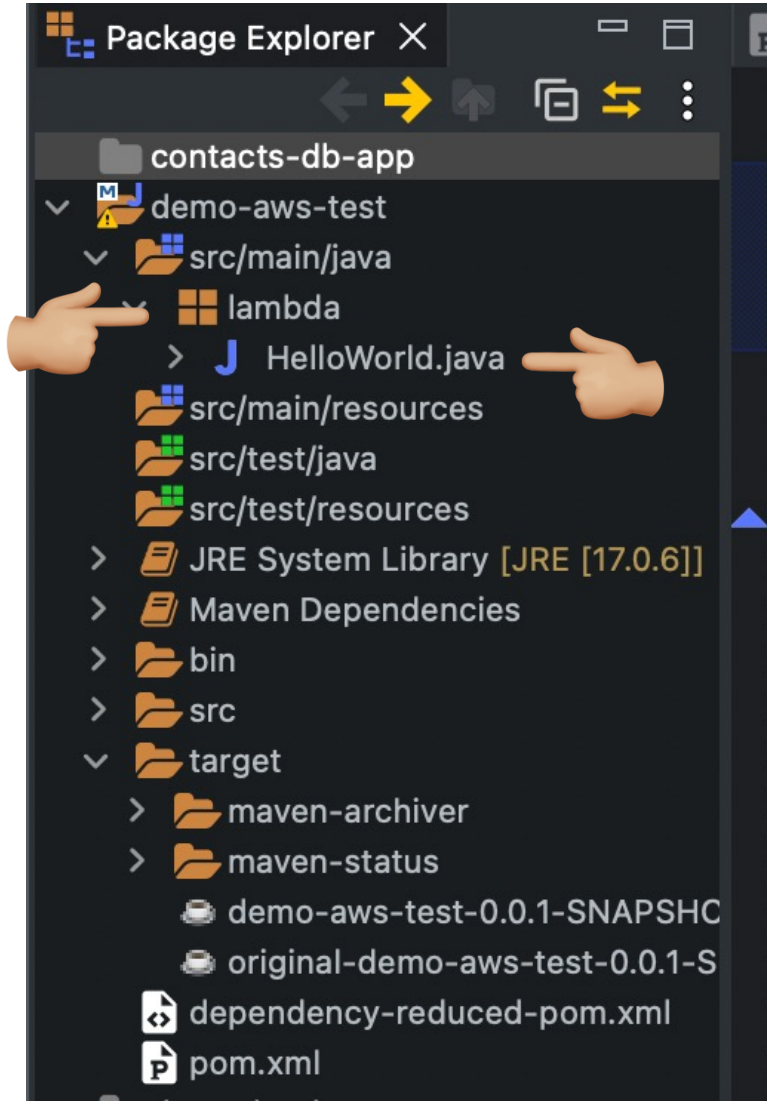
Now the the JAR file is uploaded the last step is to configure the run time settings.

You need to tell the Lambda function where in your code to find the handler method (the method that executes when the lambda function is invoked).

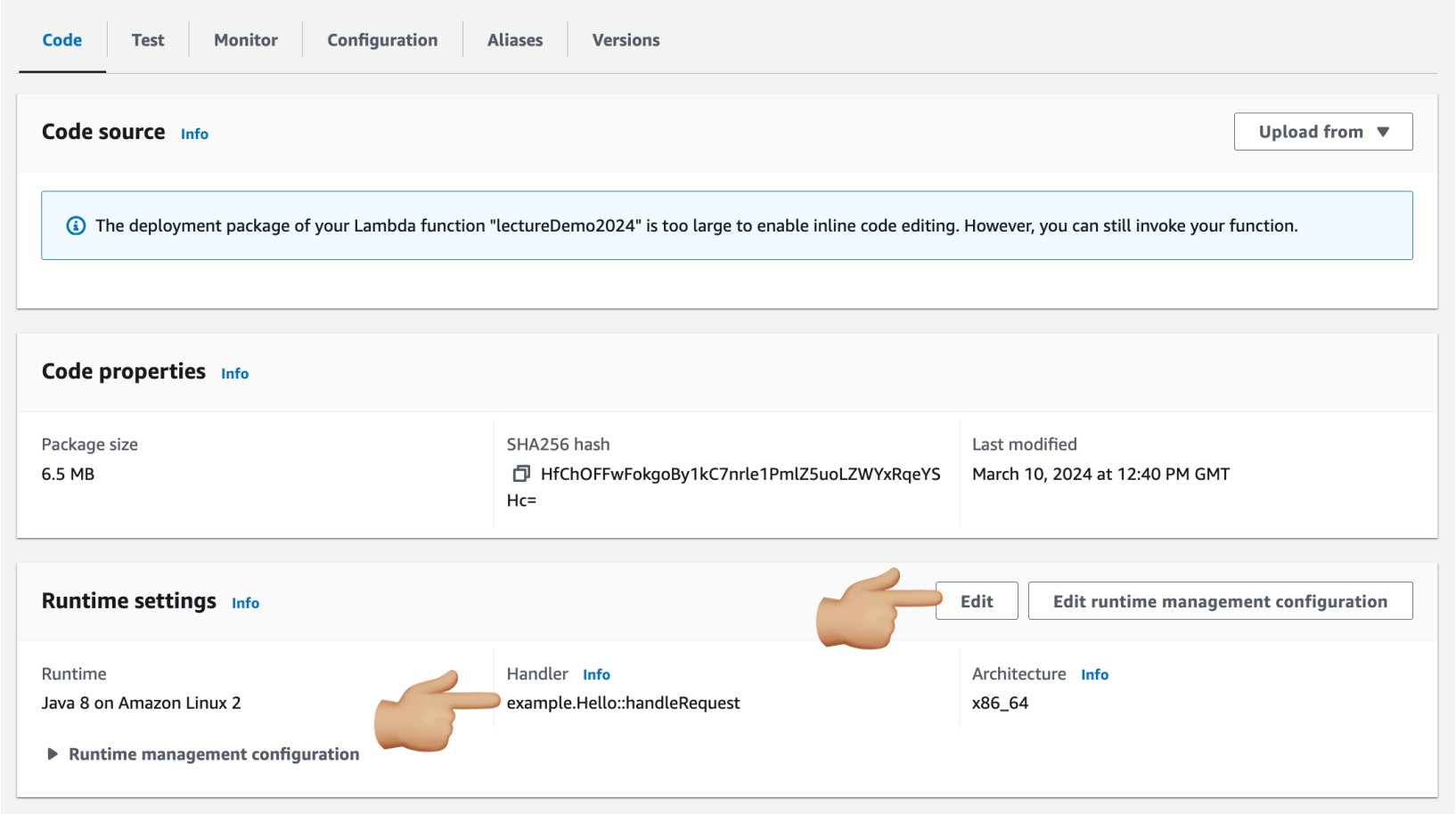
Runtime Settings

PackageName.ClassName::MethodName

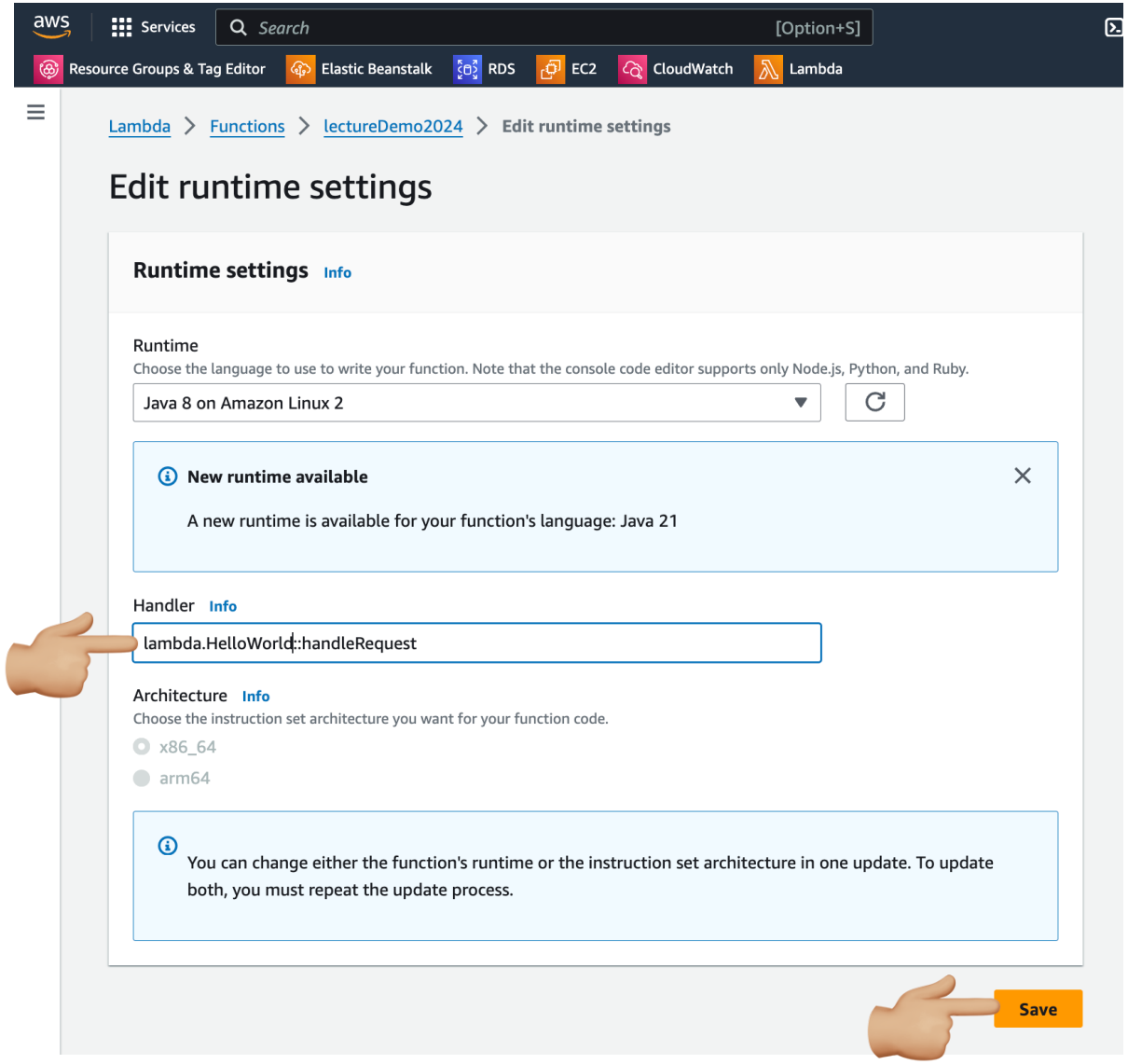
lambda.HelloWorld::handleRequest



Runtime settings



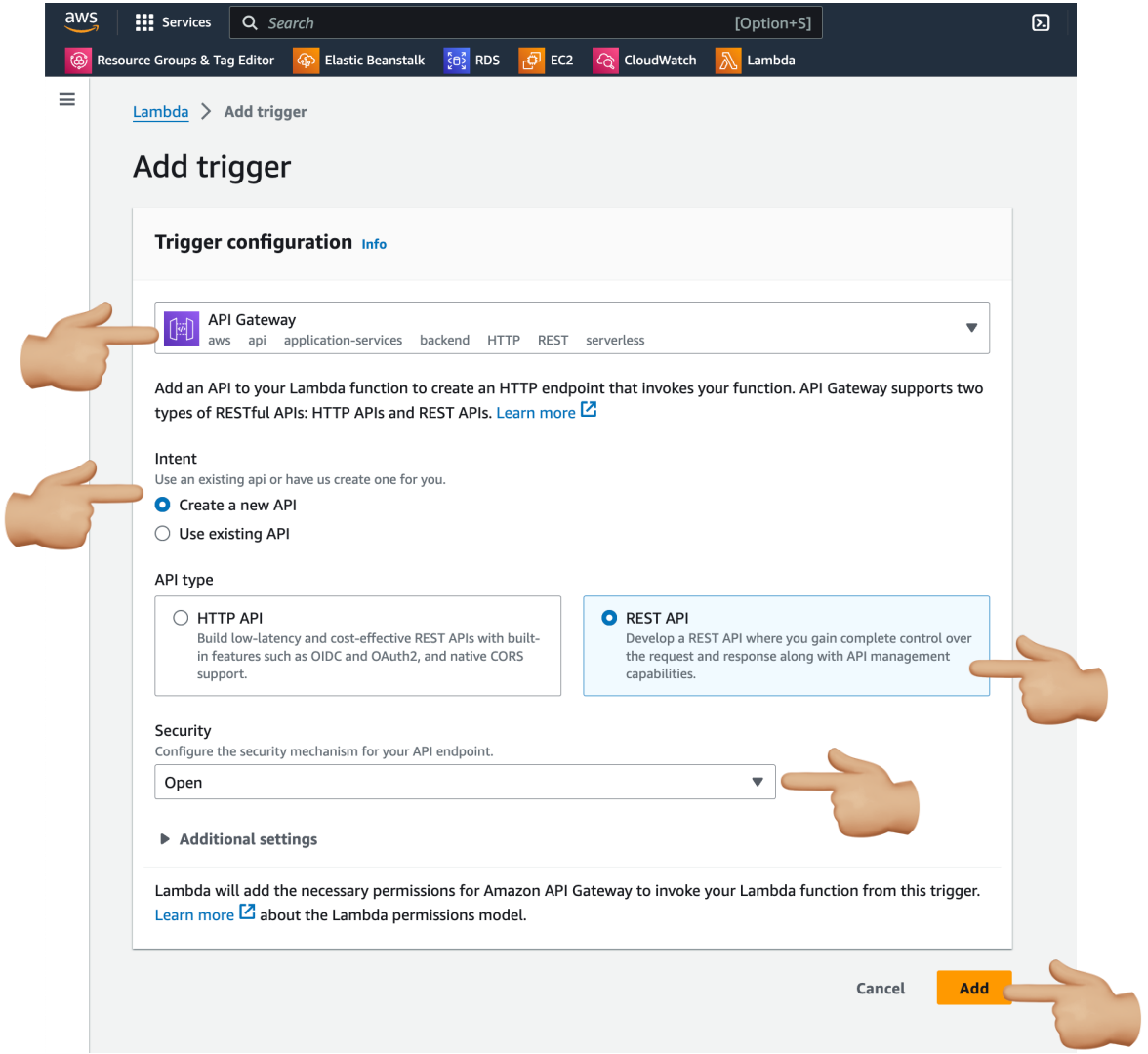
Runtime settings



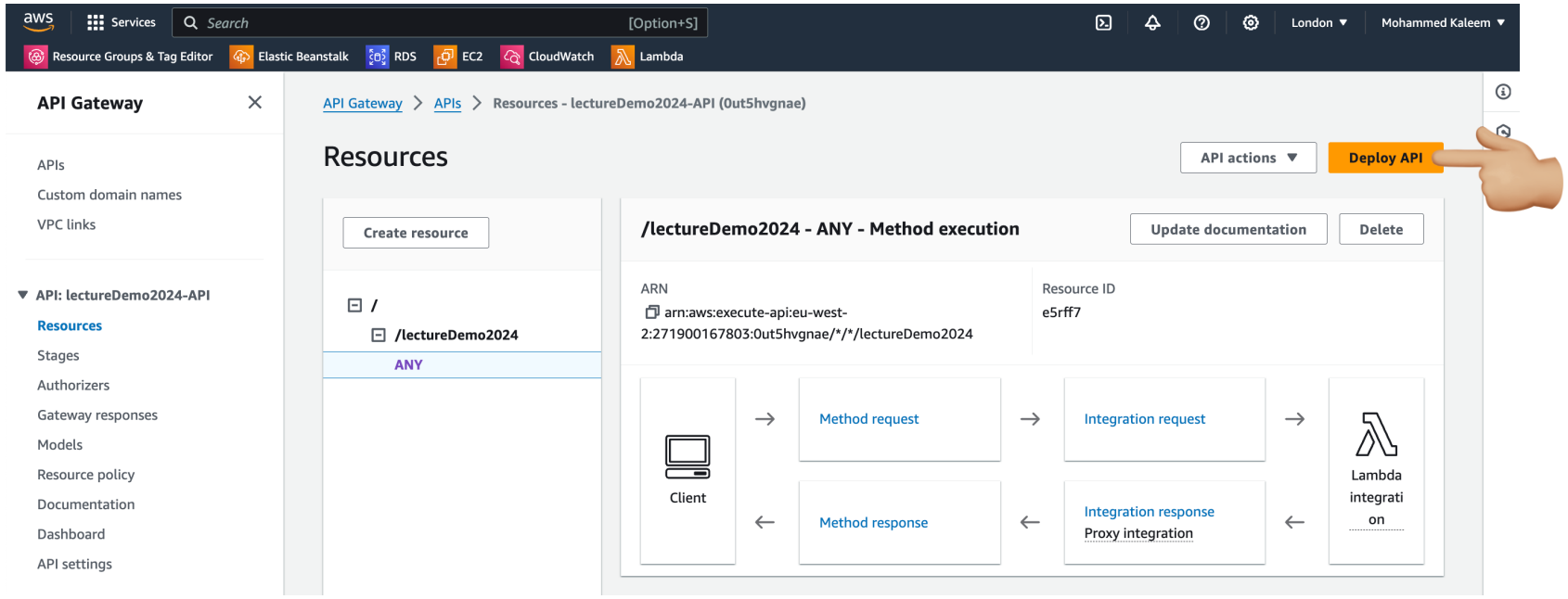
Add a Trigger

Same as last week, except this time we do not have to do any mappings.

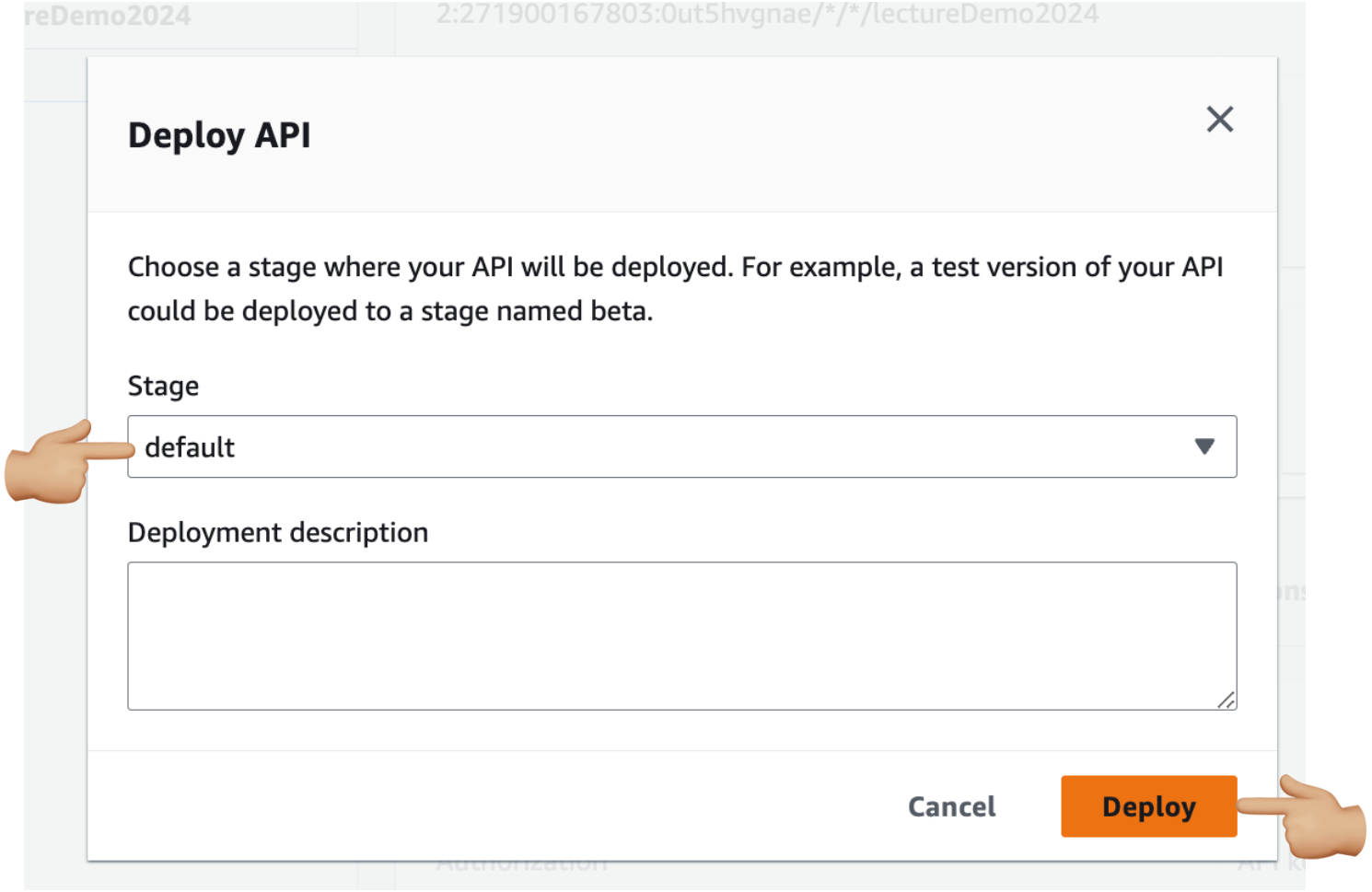
The request and all the associated request data will be passed to the function.



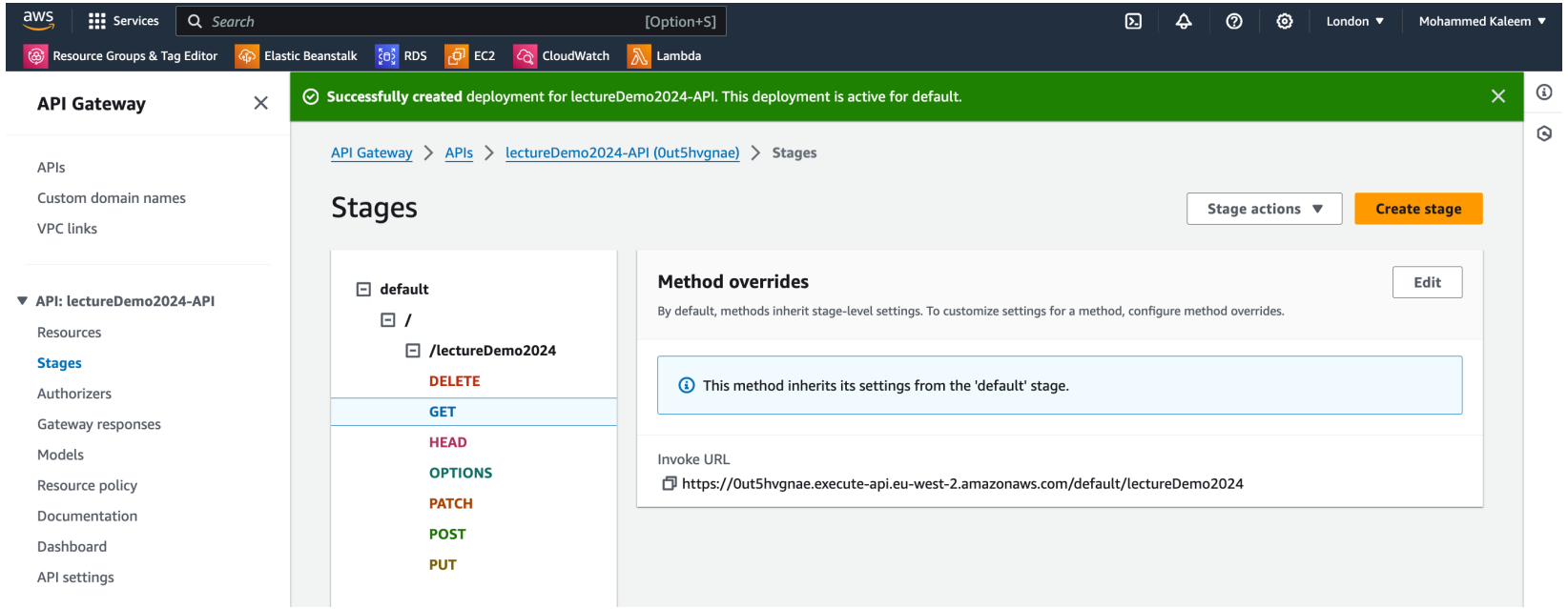
Deploy API


aws
Services
Search
[Option+S]
London
Mohammed Kaleem
Resource Groups & Tag Editor
Elastic Beanstalk
RDS
EC2
CloudWatch
Lambda
API Gateway
APIs
Custom domain names
VPC links
API: lectureDemo2024-API
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
API Gateway > APIs > Resources - lectureDemo2024-API (0ut5hvgnae)
Resources
API actions
Deploy API
Create resource
/
/lectureDemo2024
ANY
/lectureDemo2024 - ANY - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:eu-west-2:271900167803:0ut5hvgnae/*/*/lectureDemo2024
Resource ID
e5rff7
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration

Pick a Deployment Stage

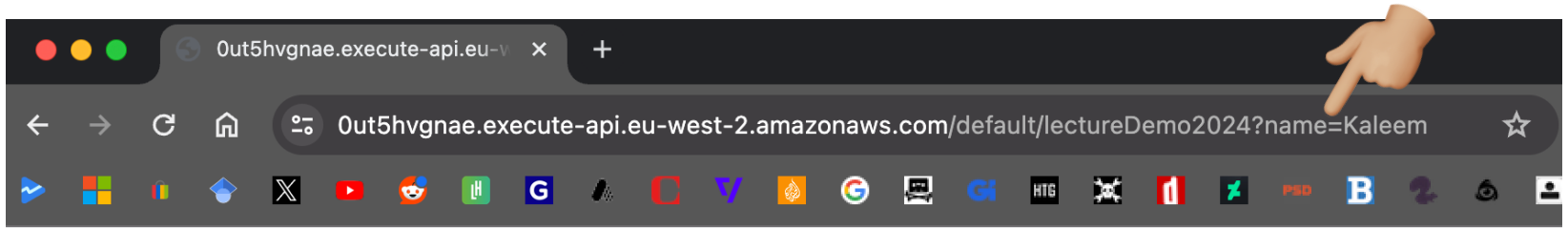


Use the GET URL and Test



Since we are not using any mappings, all the URLs for every method (GET, POST, PUT etc) are the same.

Use the GET URL and Test




```
Kaleem is testing AWS lambda functions
```

Summary

- Use maven to import the AWS specific dependencies.
- Code up your function in Java
- Upload and Deploy
- You can install AWS CLI to speed up the process, but for the purpose of learning the workflow we are doing everything manually.