

# 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

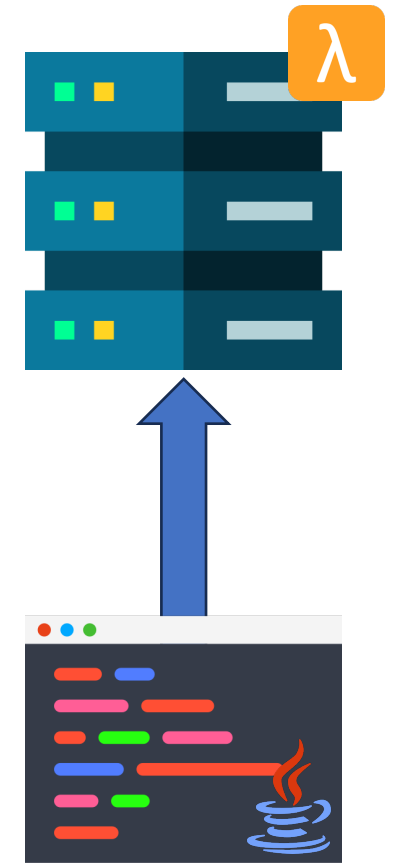
## **AWS Lambda Basics**

- 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.

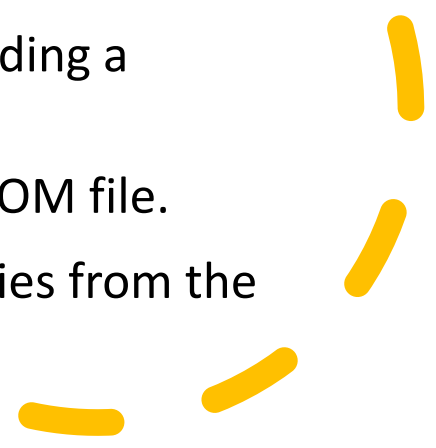
The **AWS Lambda runtime library** includes the necessary classes and interfaces for handling Lambda events and interacting with AWS services.



## 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).



```
<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/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.

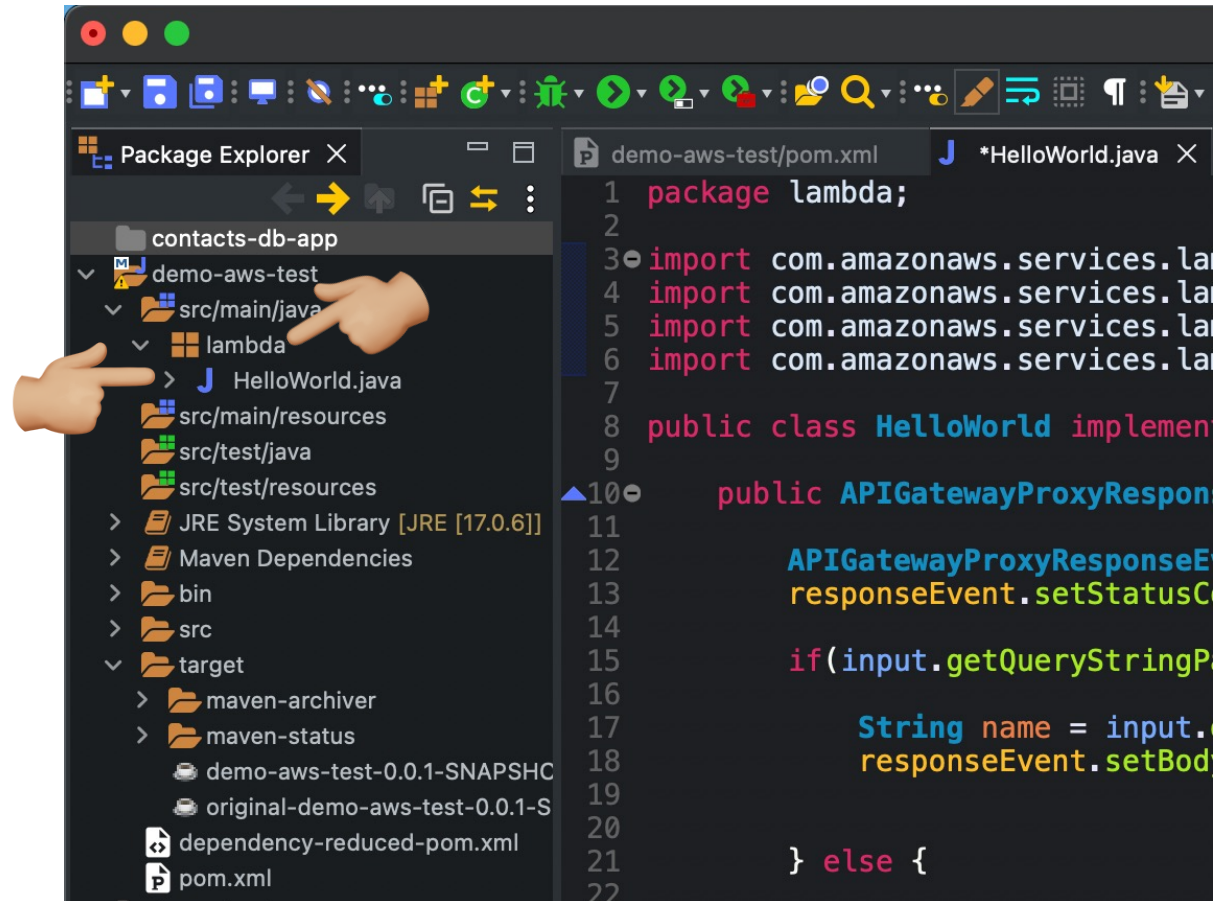
 **FILE**  
[aws-lambda-starter-project](#) 

Simple Maven project to get you started with a hello world cloud function.

 **FILE**  
[Example pom.xml file](#) An example pom.xml file containing all the required dependencies for creating and deploying AWS cloud functions. Just copy and paste the **dependencies** and **build configuration** to get a new cloud function project to get started.

# Implementing the lambda function

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



# Code up your function

```
1 package lambda;
2
3 import com.amazonaws.services.lambda.runtime.Context;
4 import com.amazonaws.services.lambda.runtime.RequestHandler;
5 import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
6 import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
7
8 public class HelloWorld implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>{
9
10     public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent input, Context context) {
11
12         APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();
13         responseEvent.setStatusCode(200);
14
15         if(input.getQueryStringParameters().containsKey("name")) {
16
17             String name = input.getQueryStringParameters().get("name");
18             responseEvent.setBody(name + " is testing AWS lambda functions");
19
20         } else {
21
22             responseEvent.setBody("name parameter is required!! ");
23
24         }
25
26         return responseEvent;
27     }
28 }
29
30
31 }
```

Input object contains all data related to the incoming request

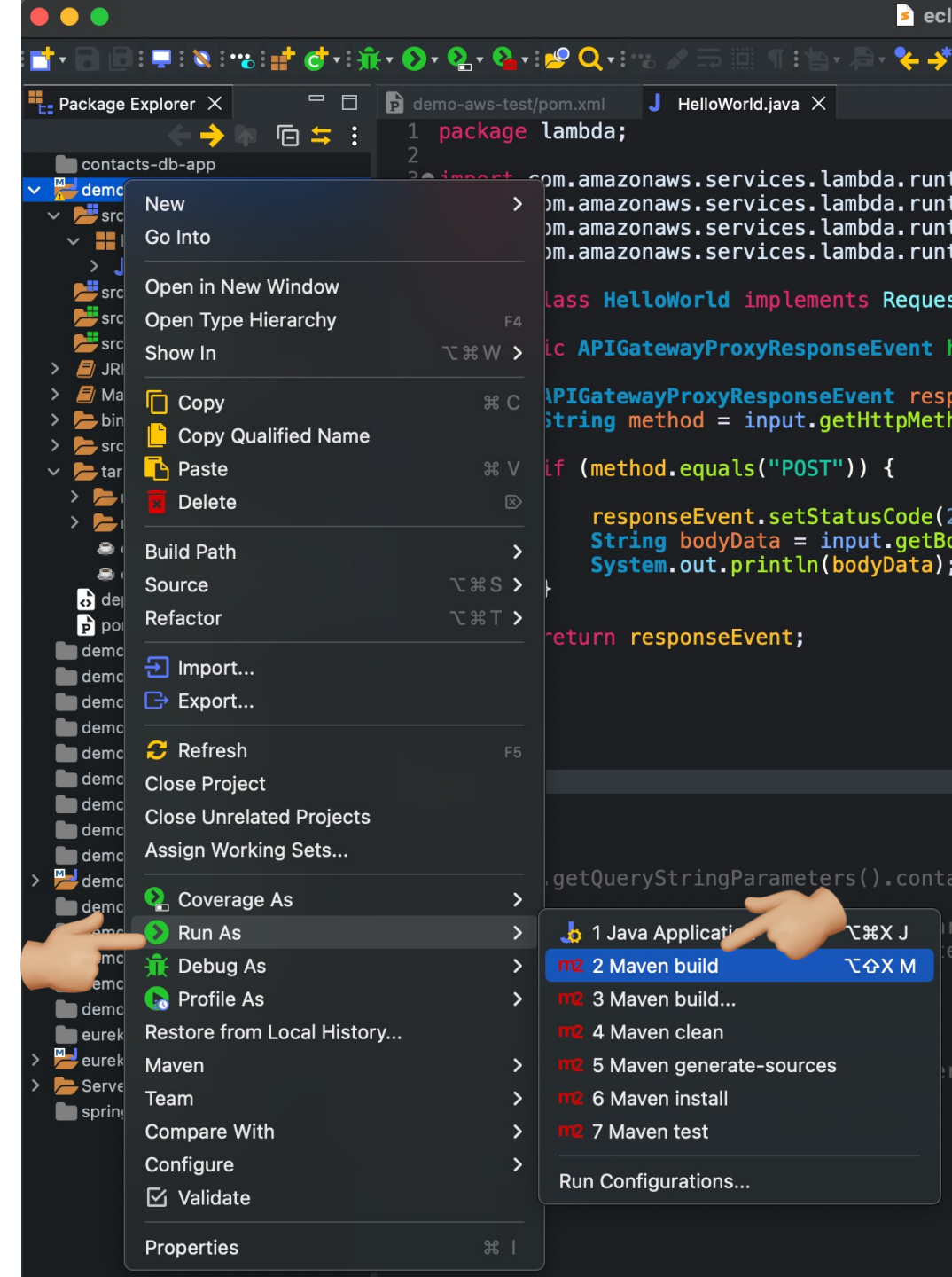
# Code up your function – continued

```
demo-aws-test/pom.xml  HelloWorld.java X
1 package lambda;
2
3 import com.amazonaws.services.lambda.runtime.Context;
4 import com.amazonaws.services.lambda.runtime.RequestHandler;
5 import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
6 import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
7
8 public class HelloWorld implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>{
9
10     public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent input, Context context) {
11
12         APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();
13         String method = input.getHttpMethod();
14
15         if (method.equals("POST")) {
16
17             responseEvent.setStatusCode(200);
18             String bodyData = input.getBody();
19             System.out.println(bodyData);
20         }
21
22         return responseEvent;
23     }
24 }
25
26
```

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

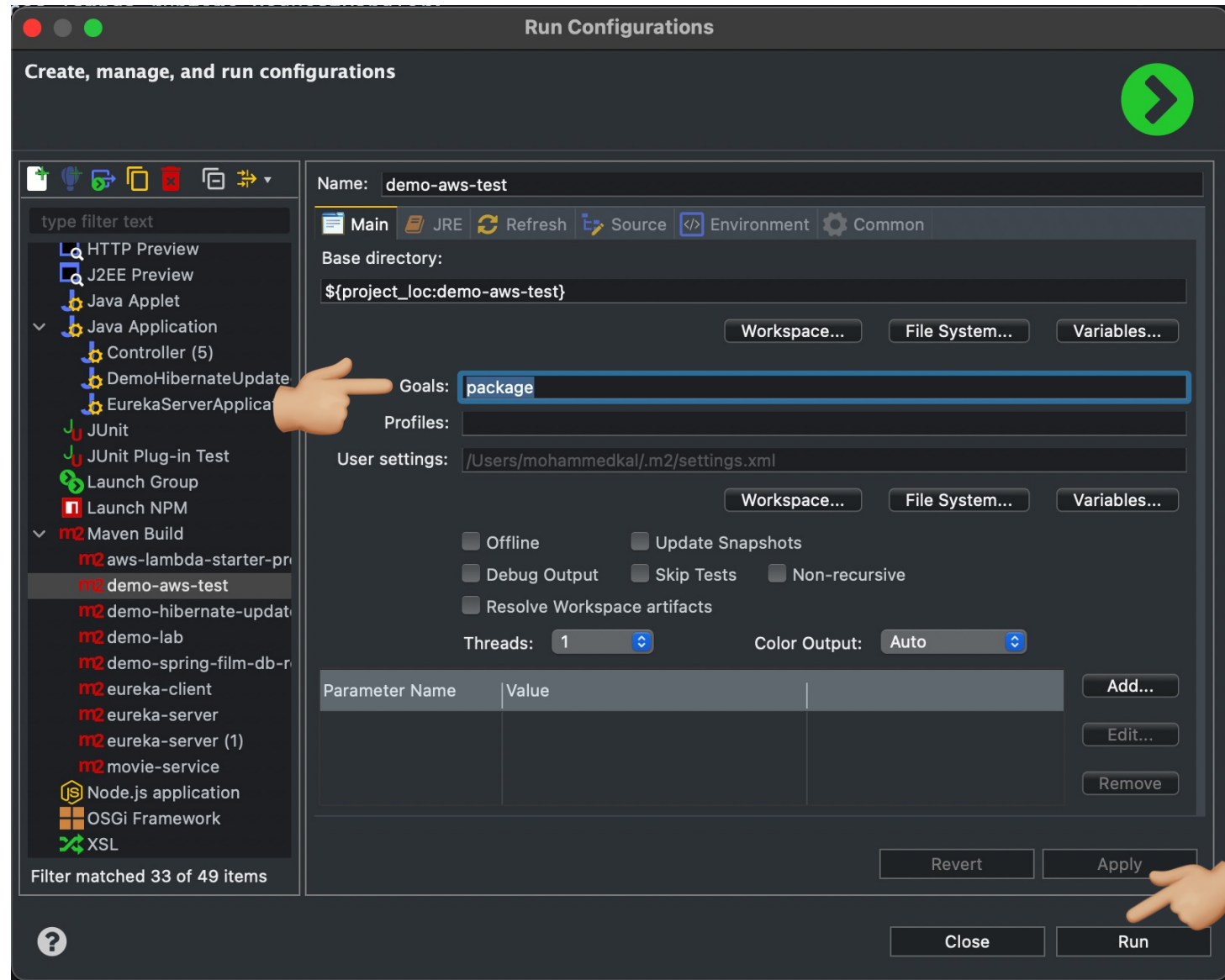




# Run Configurations

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

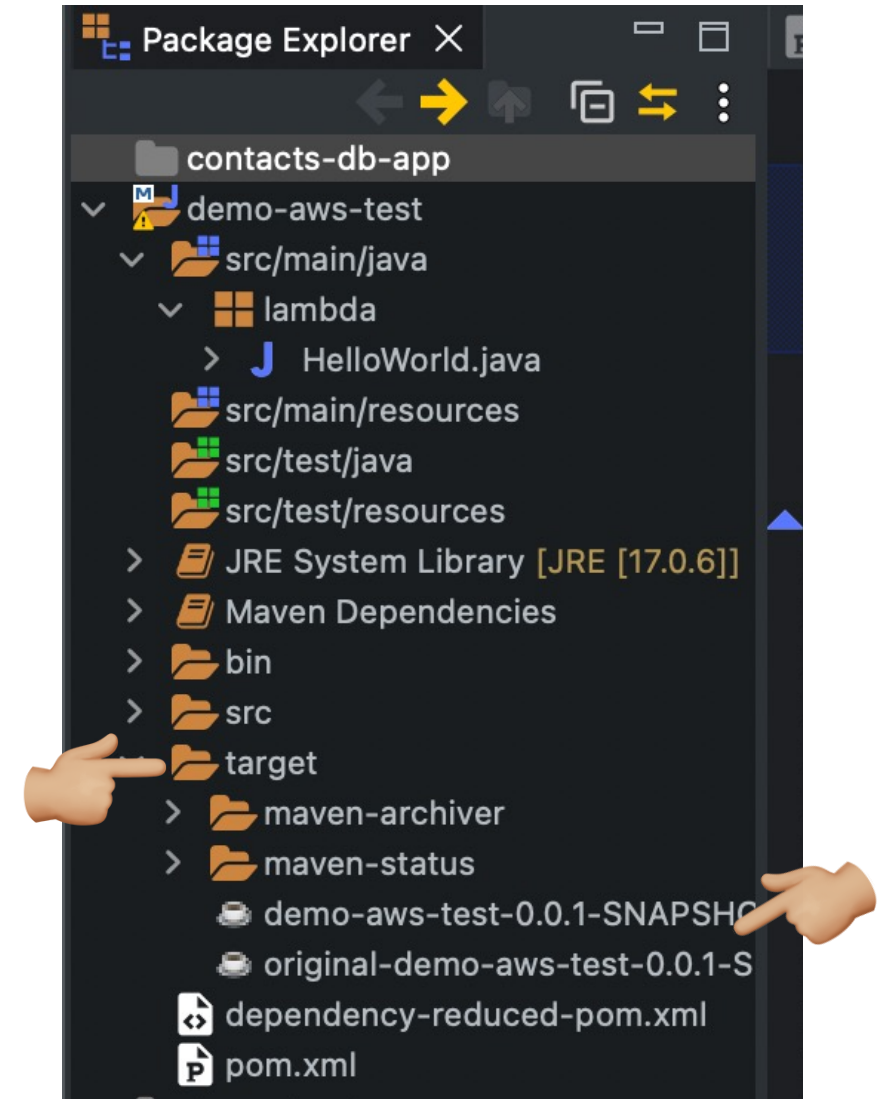
1. In the Goals option enter “package”
2. Click “Run”
3. You will see the build progress and output in the console window.



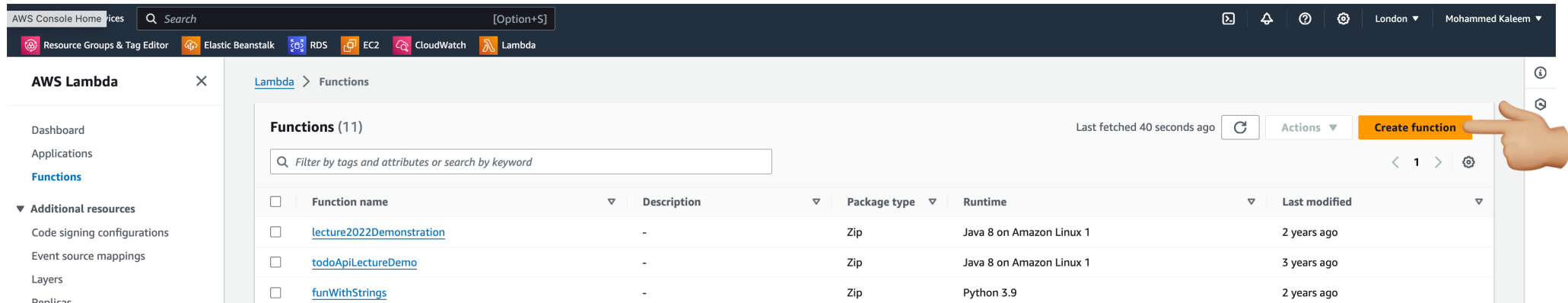
# Generated JAR File

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



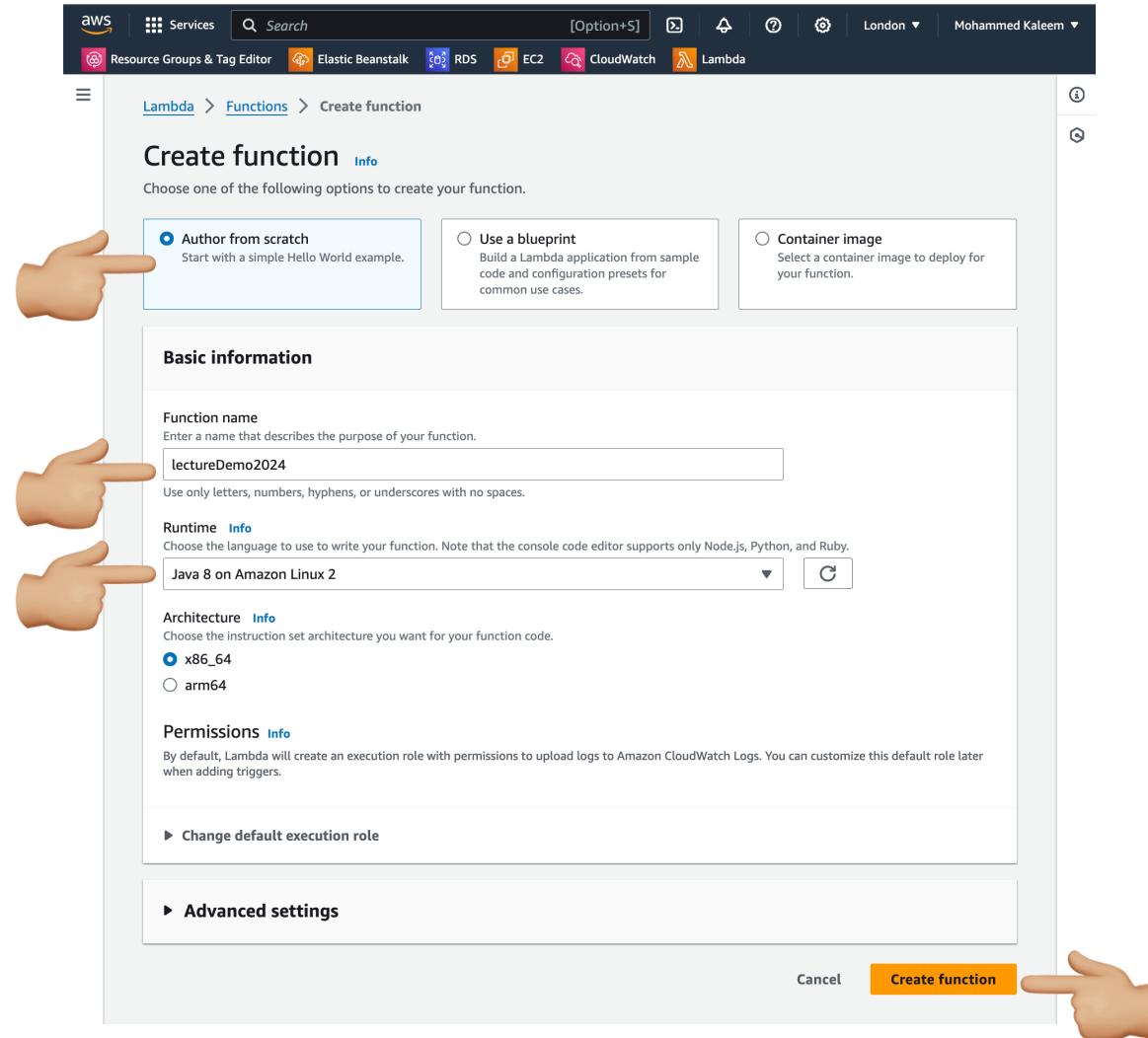
The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, a search bar, and a list of services: Resource Groups & Tag Editor, Elastic Beanstalk, RDS, EC2, CloudWatch, and Lambda. The left sidebar shows the 'AWS Lambda' section with a 'Functions' link. The main content area displays 'Functions (11)' with a search bar and a table of functions. A hand icon points to the 'Create function' button in the top right corner of the main content area.

**Functions (11)** Last fetched 40 seconds ago [Refresh](#) [Actions](#) [Create function](#)

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	<a href="#">lecture2022Demonstration</a>	-	Zip	Java 8 on Amazon Linux 1	2 years ago
<input type="checkbox"/>	<a href="#">todoApiLectureDemo</a>	-	Zip	Java 8 on Amazon Linux 1	3 years ago
<input type="checkbox"/>	<a href="#">funWithStrings</a>	-	Zip	Python 3.9	2 years ago



# Configure Lambda Function



The screenshot shows the AWS Lambda 'Create function' page. At the top, the navigation bar includes the AWS logo, 'Services', a search bar, and user information. Below the navigation bar, the breadcrumb trail reads 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'Info' link. A subtext says 'Choose one of the following options to create your function.' There are three radio button options: 'Author from scratch' (selected, with a hand icon pointing to it), 'Use a blueprint', and 'Container image'. The 'Basic information' section contains three fields: 'Function name' (with value 'lectureDemo2024' and a hand icon pointing to it), 'Runtime' (with value 'Java 8 on Amazon Linux 2' and a hand icon pointing to it), and 'Architecture' (with value 'x86\_64'). Below these is the 'Permissions' section with a 'Change default execution role' link. At the bottom right, there are 'Cancel' and 'Create function' buttons, with a hand icon pointing to the 'Create function' button.

aws Services Search [Option+S] London Mohammed Kaleem

Resource Groups & Tag Editor Elastic Beanstalk RDS EC2 CloudWatch Lambda

Lambda > Functions > Create function

## Create function [Info](#)

Choose one of the following options to create your function.

- ☒ **Author from scratch**  
Start with a simple Hello World example.
- ☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.
- ☐ **Container image**  
Select a container image to deploy for your function.

### Basic information

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.  
[Change default execution role](#)

[Advanced settings](#)

Cancel **Create function**

# Upload your code (JAR file)

The screenshot displays the AWS Lambda console interface for a function named 'lectureDemo2024'. At the top, a green notification bar states: 'Successfully created the function lectureDemo2024. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The breadcrumb navigation shows 'Lambda > Functions > lectureDemo2024'. The function name 'lectureDemo2024' is prominently displayed, along with buttons for 'Throttle', 'Copy ARN', and 'Actions'. Below this, the 'Function overview' section includes tabs for 'Diagram' and 'Template', a visual representation of the function with its layers, and buttons for '+ Add trigger' and '+ Add destination'. To the right, a metadata panel lists 'Description' (empty), 'Last modified' (43 seconds ago), 'Function ARN' (arn:aws:lambda:eu-west-2:271900167803:function:lectureDemo2024), and 'Function URL' (empty). The 'Code source' tab is active, showing a message: 'The code editor does not support the Java 8 on Amazon Linux 2 runtime.' An 'Upload from' dropdown menu is open, with two hand icons pointing to the '.zip or .jar file' and 'Amazon S3 location' options.

aws Services Search [Option+S] London Mohammed Kaleem

Resource Groups & Tag Editor Elastic Beanstalk RDS EC2 CloudWatch Lambda

Successfully created the function **lectureDemo2024**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > lectureDemo2024

**lectureDemo2024** Throttle Copy ARN Actions

Function overview Info

Diagram Template

lectureDemo2024

Layers (0)

+ Add trigger + Add destination

Export to Application Composer Download

Description -

Last modified 43 seconds ago

Function ARN  
arn:aws:lambda:eu-west-2:271900167803:function:lectureDemo2024

Function URL Info

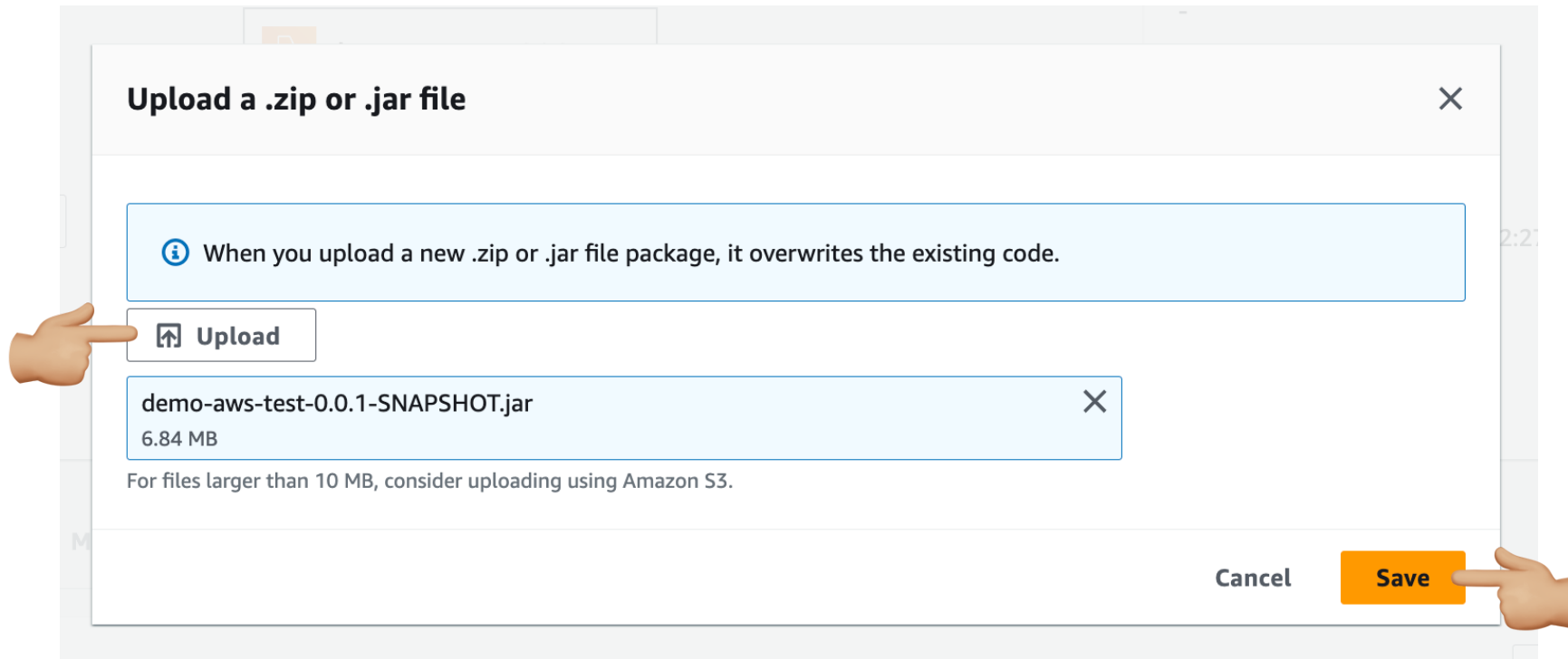
Code Test Monitor Configuration Aliases Versions

Code source Info

The code editor does not support the Java 8 on Amazon Linux 2 runtime.

Upload from ▲  
.zip or .jar file  
Amazon S3 location

# Upload your code (JAR file)



# Runtime settings

---

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

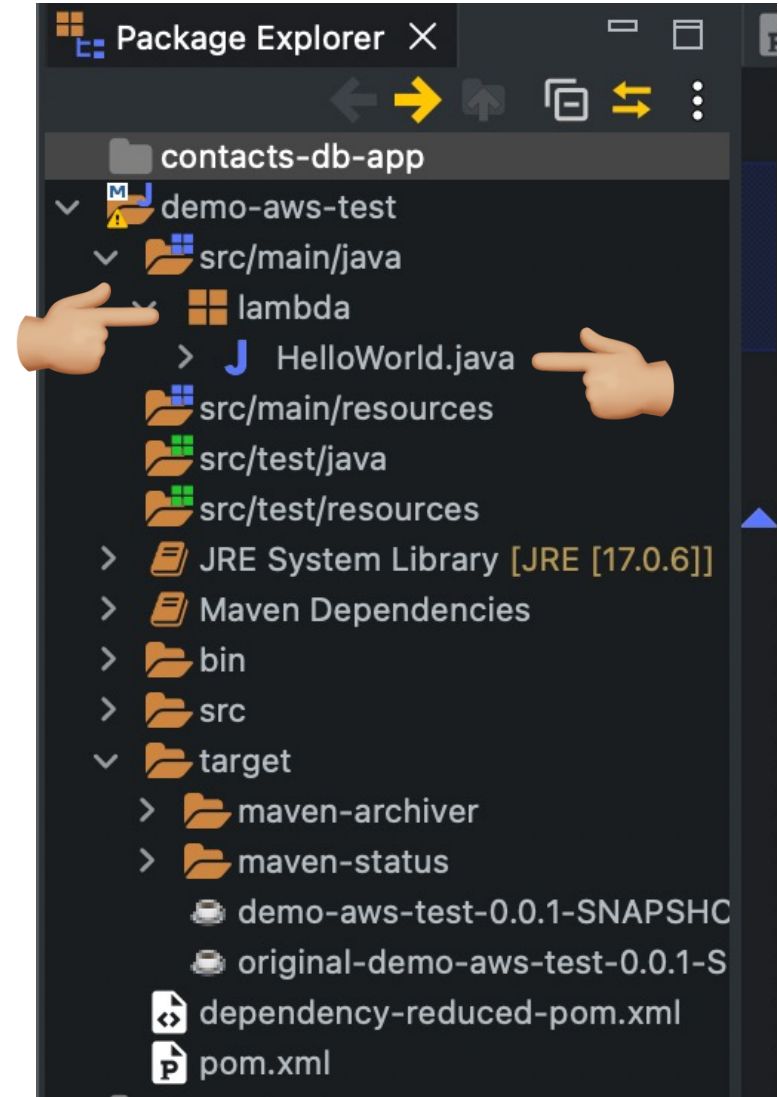
You need to tell the Lambda function where in your code to find the handler method (the method that executes when the lambda function is invoked).



# Runtime Settings

PackageName.ClassName::MethodName


lambda.HelloWorld::handleRequest




# Runtime settings

[Code](#) | [Test](#) | [Monitor](#) | [Configuration](#) | [Aliases](#) | [Versions](#)

**Code source** [Info](#) Upload from ▼

 The deployment package of your Lambda function "lectureDemo2024" is too large to enable inline code editing. However, you can still invoke your function.

**Code properties** [Info](#)

Package size 6.5 MB	SHA256 hash  HfChOFFwFokgoBy1kC7nrle1PmlZ5uoLZWYxRqeYS Hc=	Last modified March 10, 2024 at 12:40 PM GMT
------------------------	---	---

**Runtime settings** [Info](#) Edit Edit runtime management configuration

Runtime Java 8 on Amazon Linux 2  ▶ Runtime management configuration	Handler <a href="#">Info</a> example.Hello::handleRequest	Architecture <a href="#">Info</a> x86_64
---	--	---

# Runtime settings

The screenshot displays the AWS Lambda console interface for editing the runtime settings of a function named 'lectureDemo2024'. The top navigation bar shows the AWS logo, 'Services', a search bar, and a list of services including Resource Groups & Tag Editor, Elastic Beanstalk, RDS, EC2, CloudWatch, and Lambda. The breadcrumb trail indicates the path: Lambda > Functions > lectureDemo2024 > Edit runtime settings.

The main content area is titled 'Edit runtime settings' and contains the following sections:

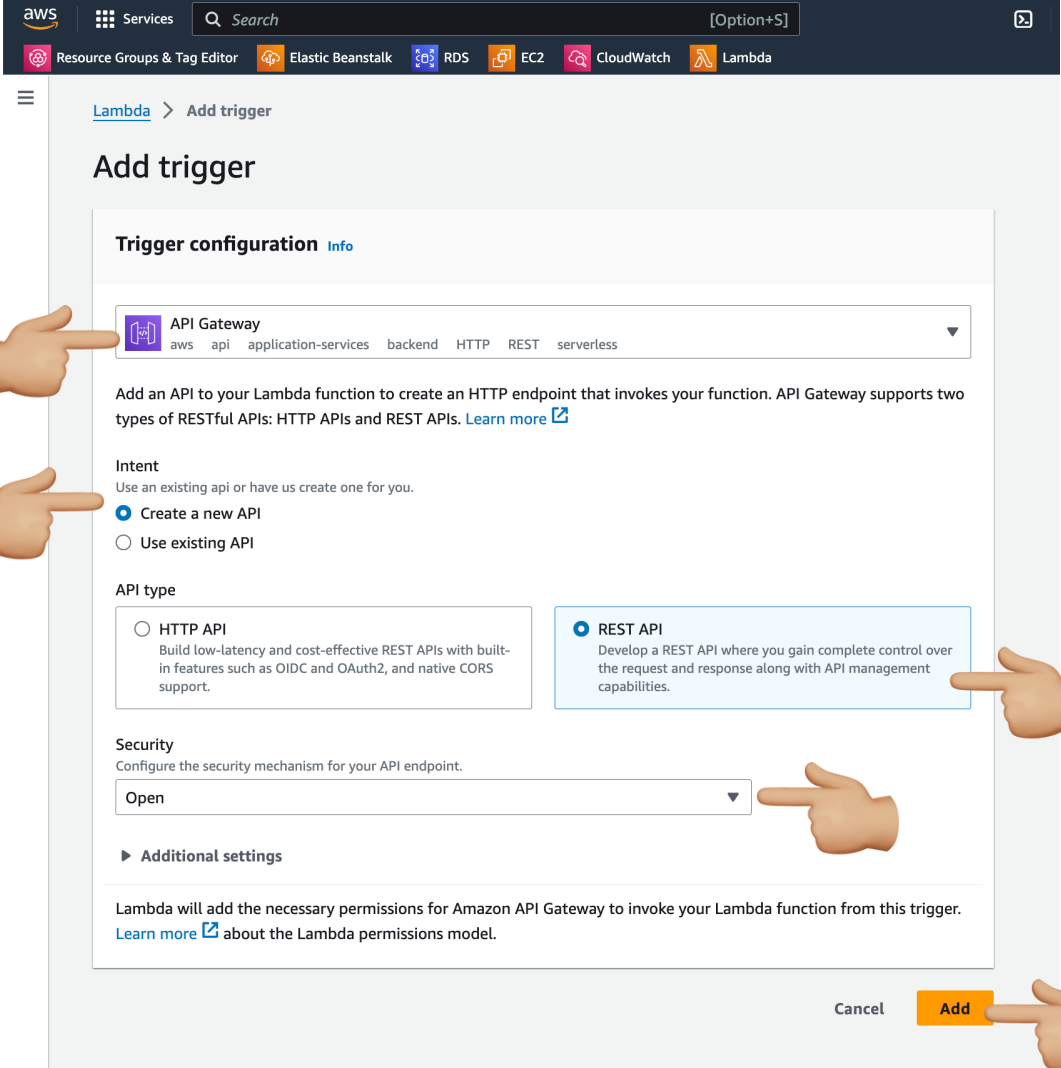
- Runtime settings** (with an 'Info' link):
  - Runtime**: A dropdown menu currently set to 'Java 8 on Amazon Linux 2'. A note states: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.' A refresh button is located to the right of the dropdown.
  - New runtime available**: A light blue notification box with a close button (X) stating: 'A new runtime is available for your function's language: Java 21'.
- Handler** (with an 'Info' link): A text input field containing 'lambda.HelloWorld::handleRequest'. A hand icon points to this field.
- Architecture** (with an 'Info' link): Radio buttons for 'x86\_64' (selected) and 'arm64'. A note states: 'Choose the instruction set architecture you want for your function code.'
- Information**: A light blue box at the bottom stating: 'You can change either the function's runtime or the instruction set architecture in one update. To update both, you must repeat the update process.'

A 'Save' button is located at the bottom right of the console, with a hand icon pointing to it.

# 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.



The screenshot shows the AWS Lambda 'Add trigger' configuration page. The page title is 'Add trigger' under the 'Lambda' service. The 'Trigger configuration' section shows 'API Gateway' selected as the trigger type. Below this, the 'Intent' section has 'Create a new API' selected. The 'API type' section has 'REST API' selected. The 'Security' section has 'Open' selected. The 'Additional settings' section is collapsed. At the bottom, there are 'Cancel' and 'Add' buttons. Hand annotations point to the 'API Gateway' dropdown, the 'Create a new API' radio button, the 'REST API' radio button, the 'Open' dropdown, and the 'Add' button.

aws Services Search [Option+S]

Resource Groups & Tag Editor Elastic Beanstalk RDS EC2 CloudWatch Lambda

Lambda > Add trigger

### Add trigger

**Trigger configuration** Info

**API Gateway**  
aws api application-services backend HTTP REST serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**Intent**  
Use an existing api or have us create one for you.

☒ Create a new API  
☐ Use existing API

**API type**

☐ HTTP API  
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

☒ REST API  
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Security**  
Configure the security mechanism for your API endpoint.

Open

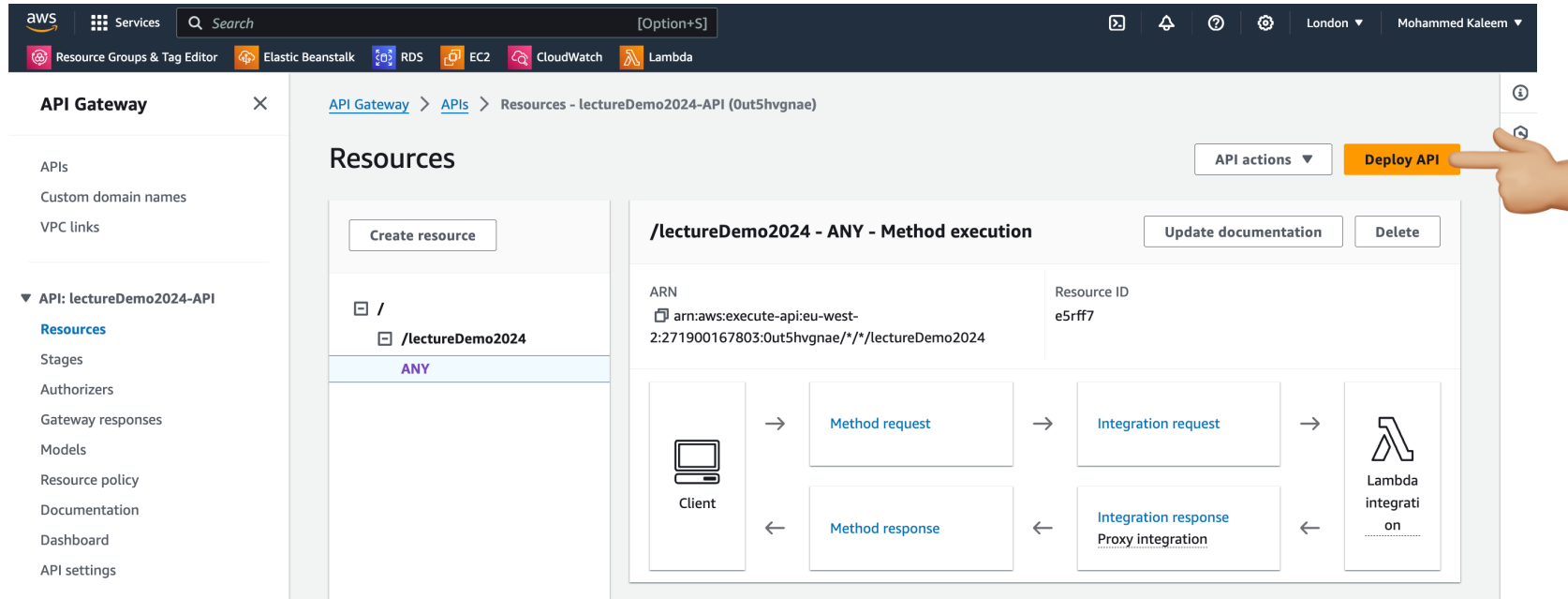
► **Additional settings**

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add



# Deploy API



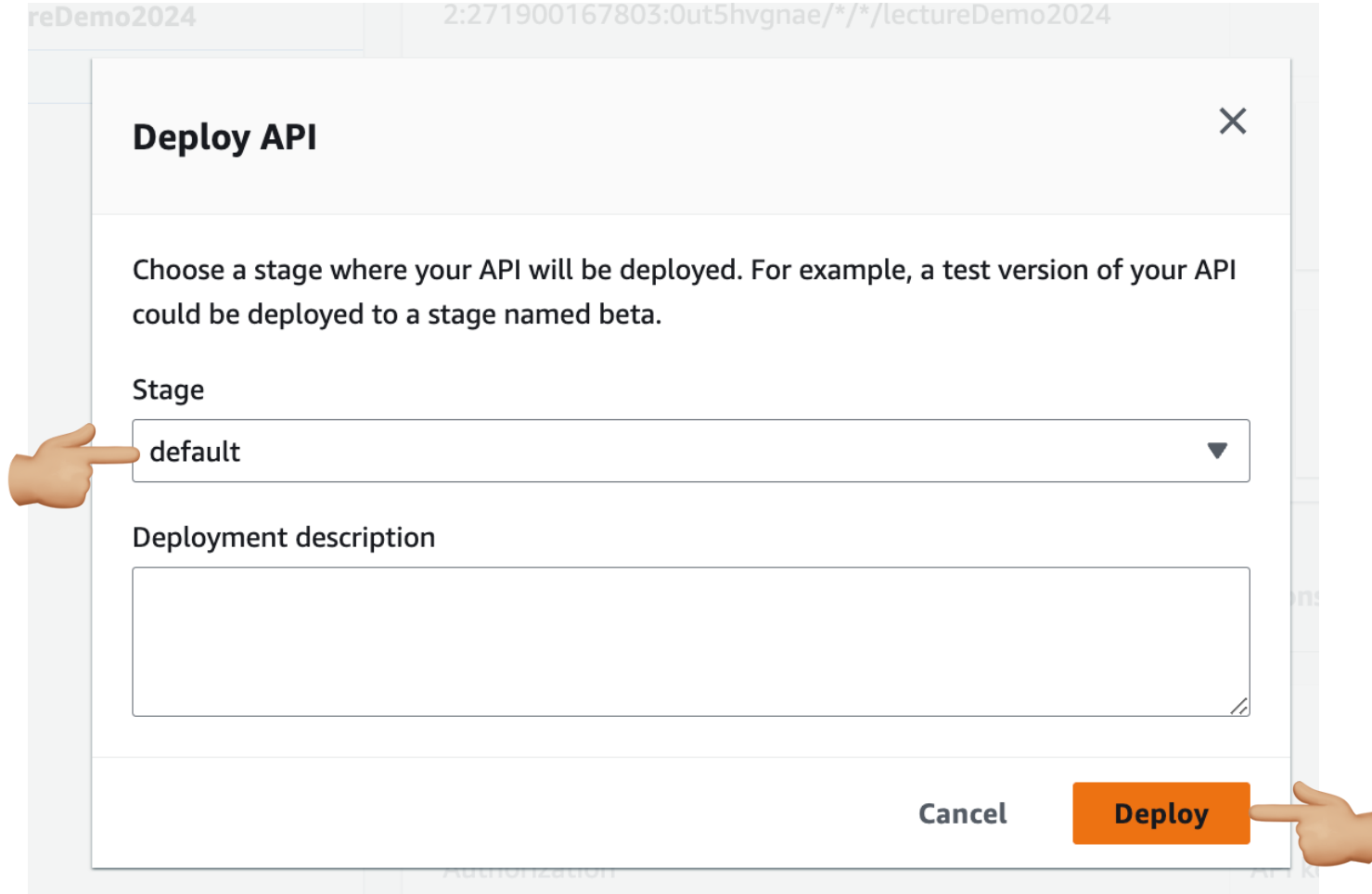
The screenshot displays the AWS API Gateway console interface. The top navigation bar includes the AWS logo, a search bar, and service icons for Resource Groups & Tag Editor, Elastic Beanstalk, RDS, EC2, CloudWatch, and Lambda. The user's location is set to London, and the account name is Mohammed Kaleem.

The left sidebar shows the 'API Gateway' section with a list of options: APIs, Custom domain names, VPC links, and a dropdown menu for 'API: lectureDemo2024-API'. The dropdown menu is open, showing 'Resources' (highlighted), Stages, Authorizers, Gateway responses, Models, Resource policy, Documentation, Dashboard, and API settings.

The main content area is titled 'Resources' and shows the path 'API Gateway > APIs > Resources - lectureDemo2024-API (0ut5hvgnae)'. A 'Create resource' button is visible. The resource list shows a path '/' with a sub-resource '/lectureDemo2024' under the 'ANY' method. The 'API actions' dropdown menu is open, and the 'Deploy API' button is highlighted with a hand cursor.

The resource details for '/lectureDemo2024 - ANY - Method execution' are shown, including the ARN 'arn:aws:execute-api:eu-west-2:271900167803:0ut5hvgnae/\*/\*/lectureDemo2024' and the Resource ID 'e5rff7'. A diagram illustrates the request flow: Client → Method request → Integration request → Lambda integration → Integration response (Proxy integration) → Method response → Client.

# Pick a Deployment Stage



**Deploy API** ✕

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

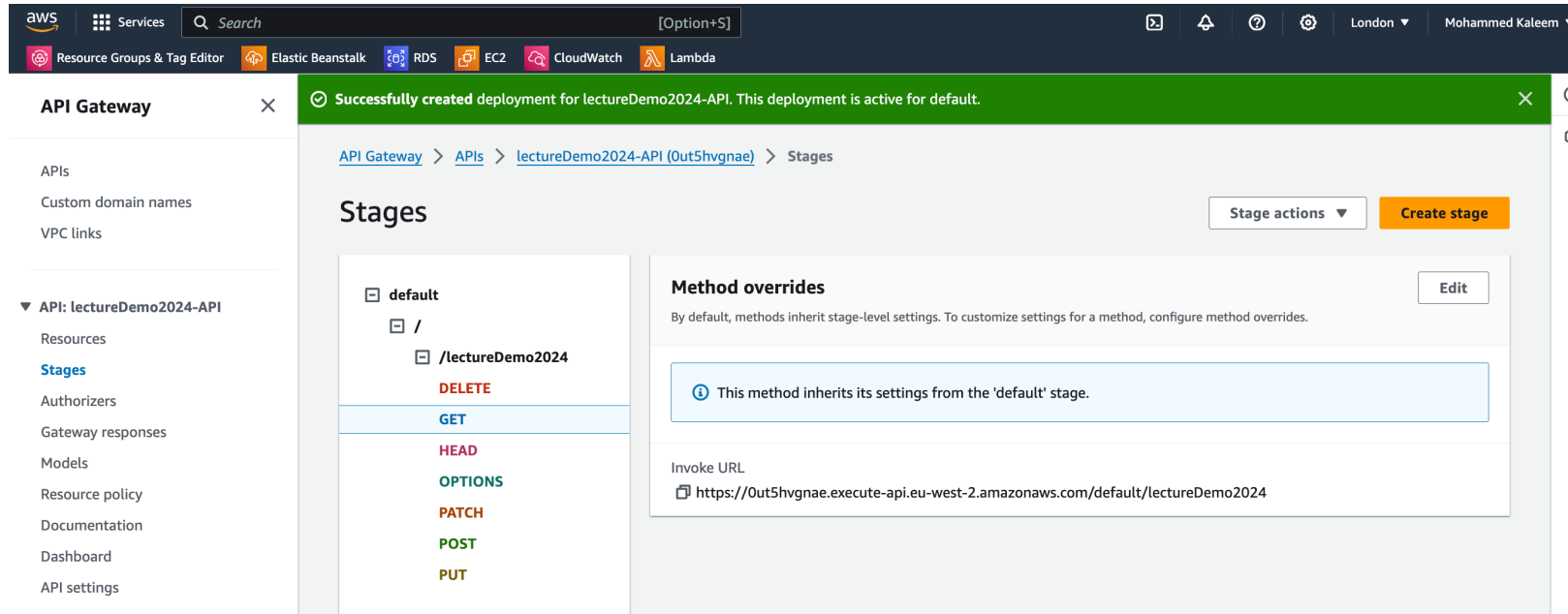
Stage

default ▼

Deployment description

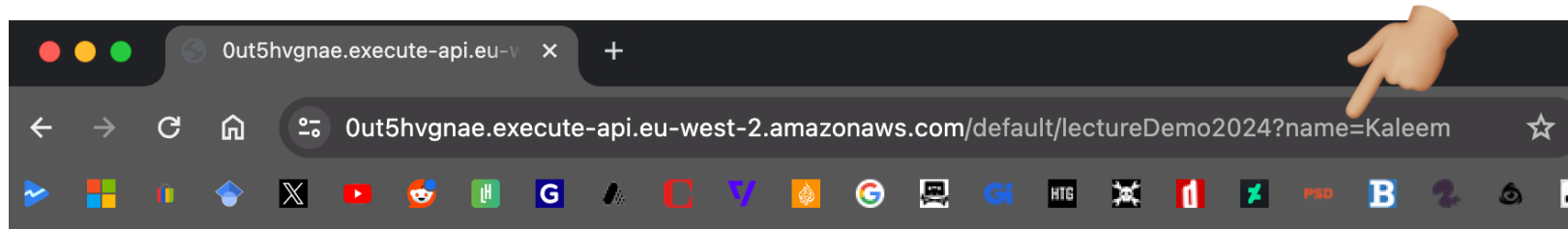
Cancel **Deploy**

# 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.