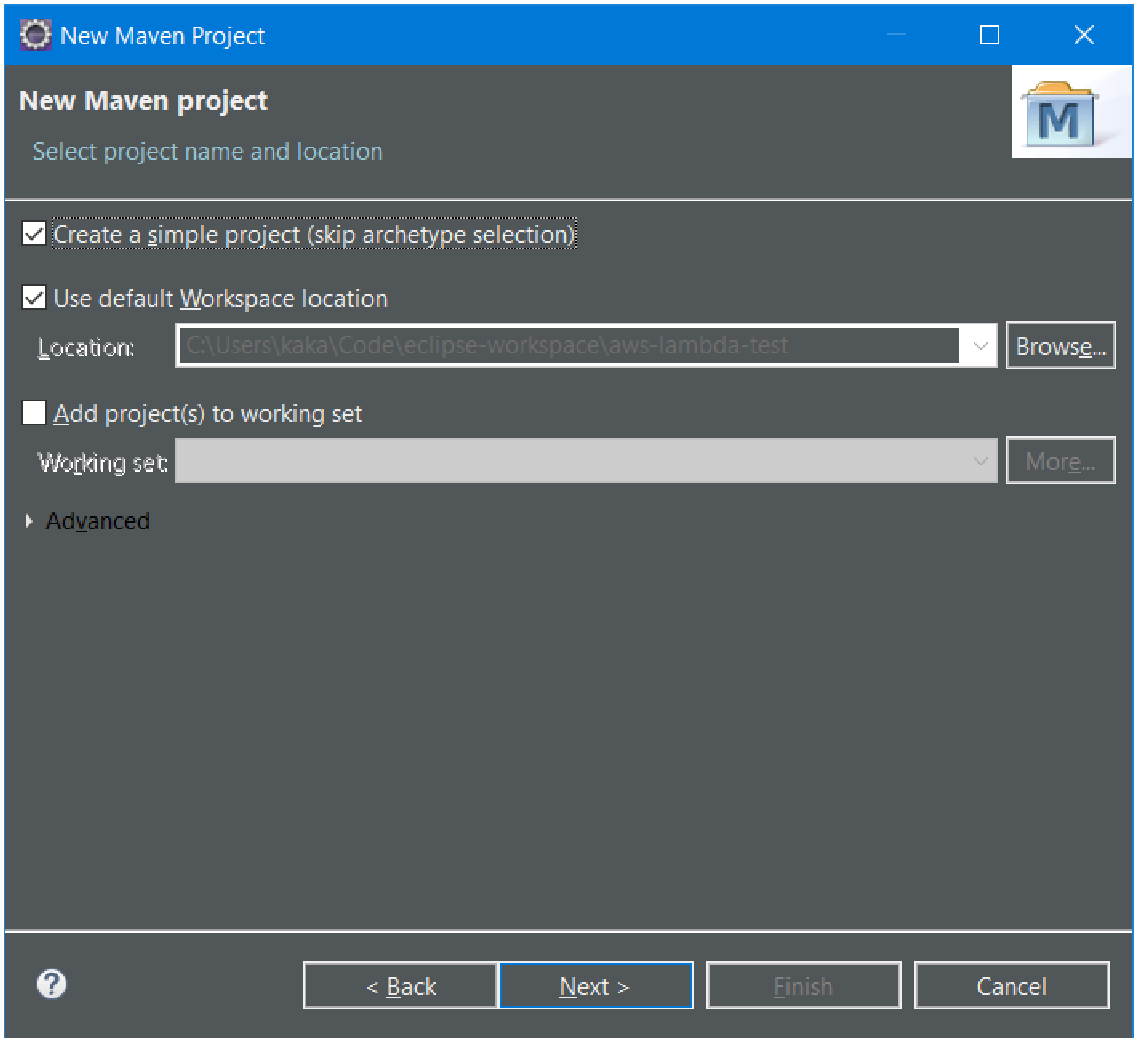
Implementing Cloud Functions in Java

Now that you have had a taste of developing serverless cloud functions, it is time to step it up a notch. In this lab you will learn how to implement more complex cloud functions in eclipse using java and upload the code as a compiled JAR file to AWS Lambda.

Task 1

Getting started:

1. Create a new maven project in eclipse, remember to check "Skip Archetype Selection" as we will be configuring the project from scratch.



New Maven Project

New Maven project

Select project name and location

☑ Create a simple project (skip archetype selection)

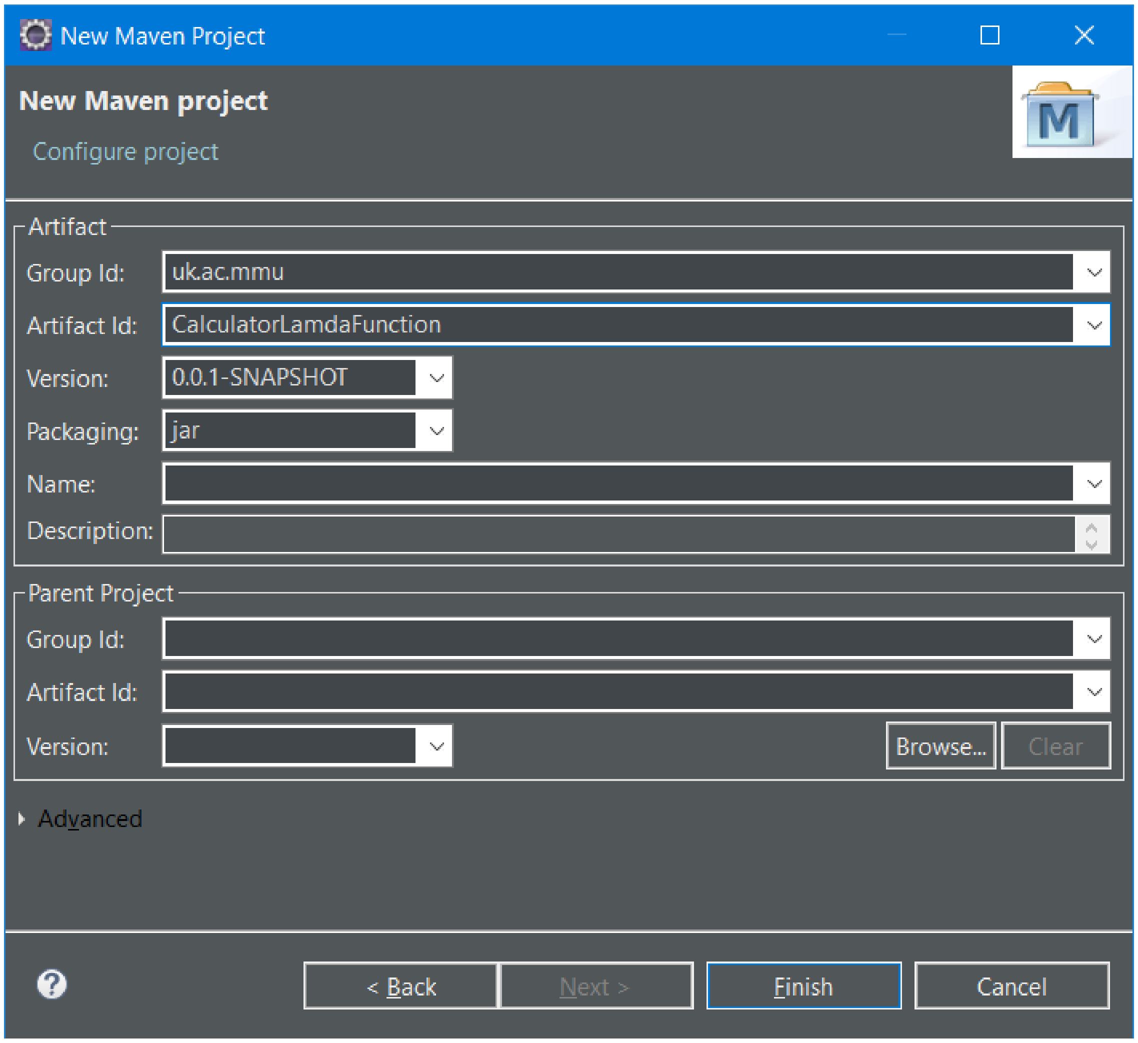
☑ Use default Workspace location

Location: C:\Users\kaka\Code\eclipse-workspace\aws-lambda-test

☐ Add project(s) to working set

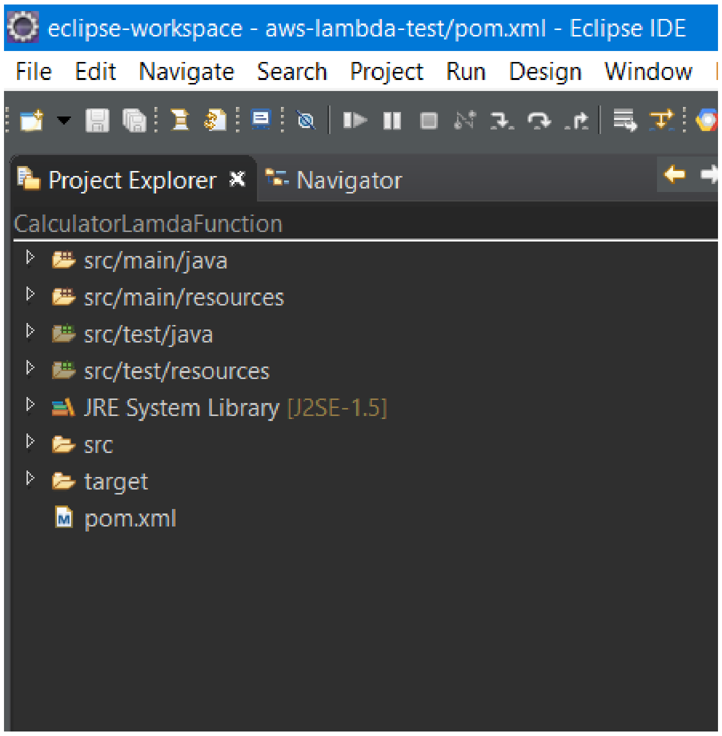
Working set:

2. Give project a “group id” (normally your organisations URL in reverse) and an “Artifact id” (this will be the name of the project), then hit finish.

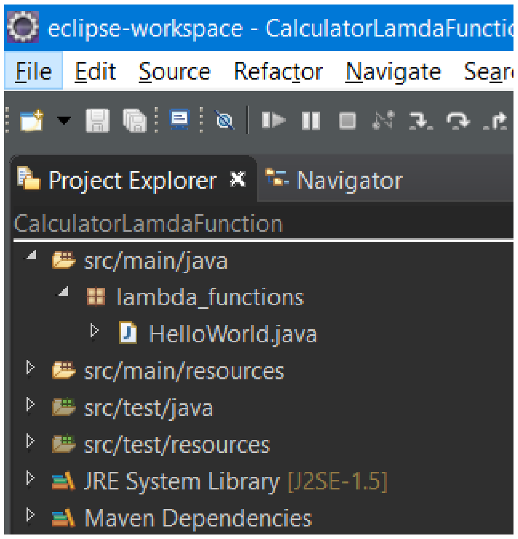


Group Id: uk.ac.mmu
Artifact Id: CalculatorLamdaFunction
Version: 0.0.1-SNAPSHOT
Packaging: jar
Name:
Description:

3. When the project is created, locate the pom.xml file and add the required dependencies for creating AWS lambda supported functions (the required dependencies can be found on Moodle).

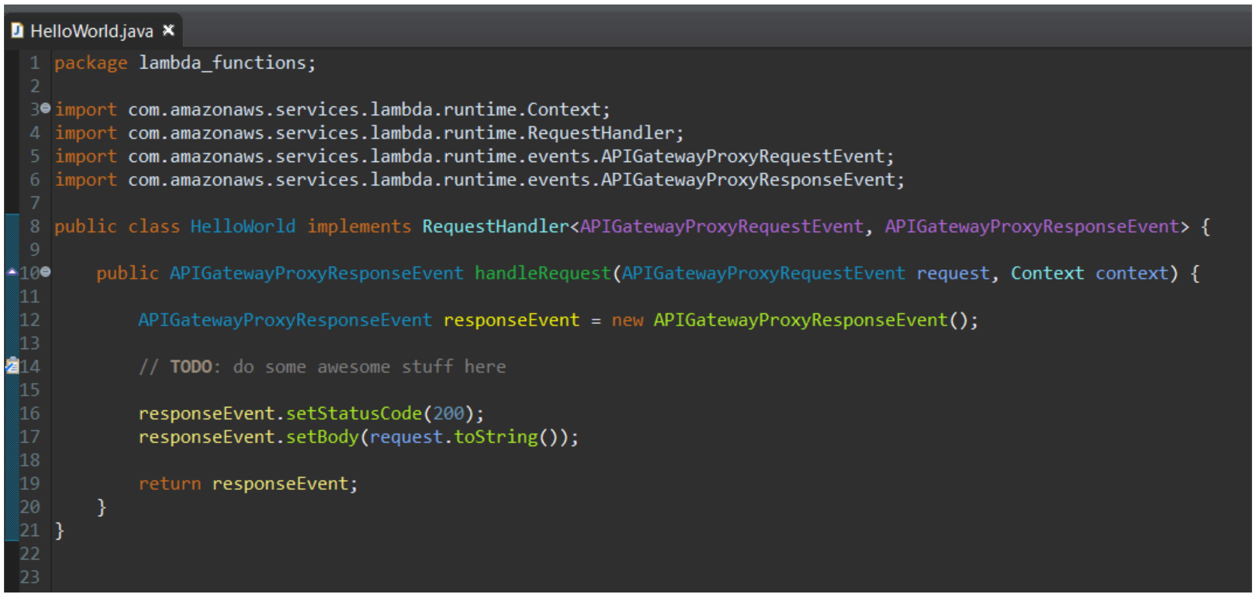

eclipse-workspace - aws-lambda-test/pom.xml - Eclipse IDE
File Edit Navigate Search Project Run Design Window
Project Explorer
Navigator
CalculatorLamdaFunction
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
src
target
pom.xml

4. Create a new package to house your code/java files and create a new java class to implement your cloud function:



5. Implement the RequestHandler interface and override the handleRequest method that is required by all AWS cloud functions.

   The following code example just output the entire request object as a response to any request made.

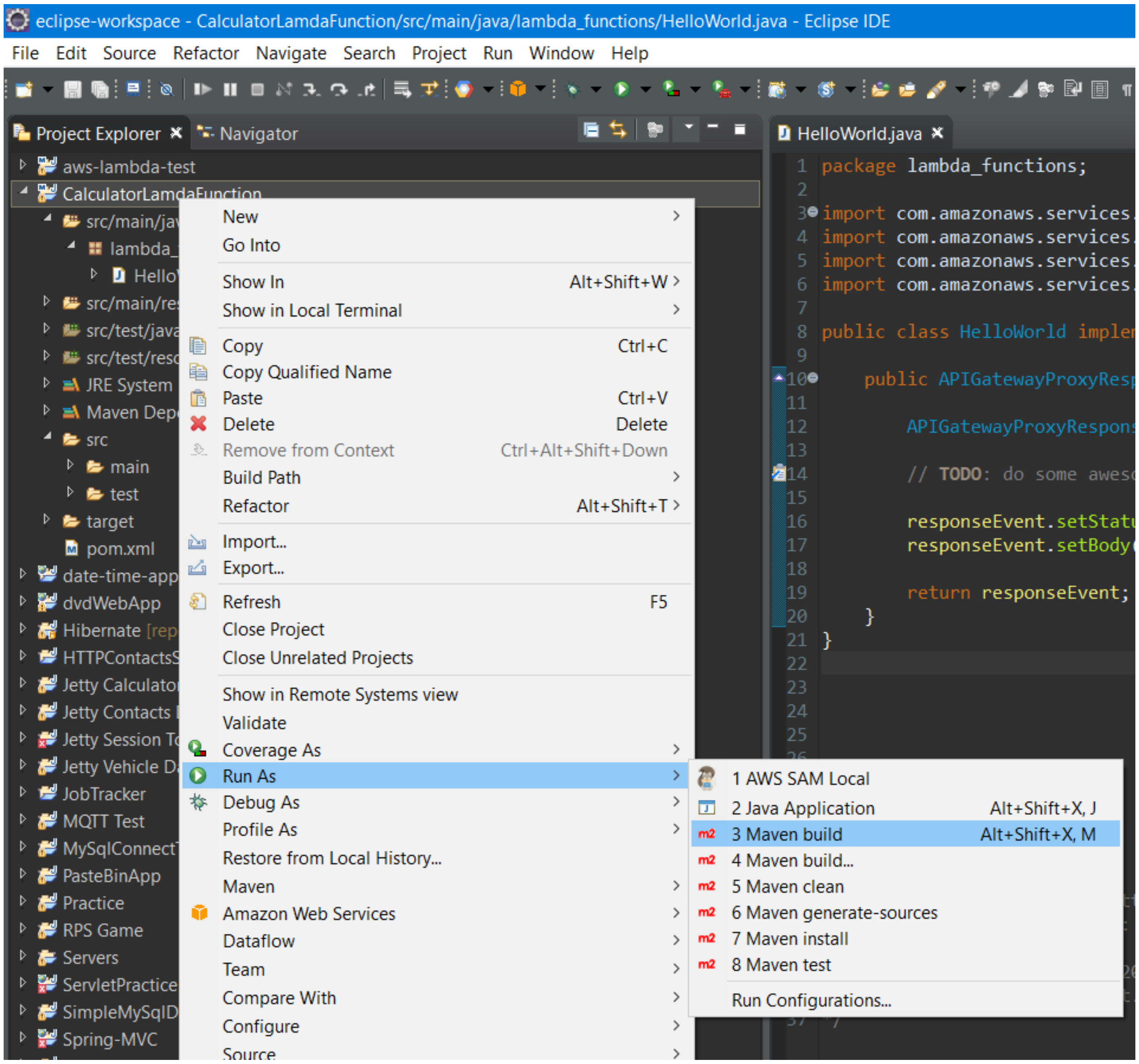


HelloWorld.java

```java
package lambda_functions;

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 request, Context context) {

        APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();

        // TODO: do some awesome stuff here

        responseEvent.setStatusCode(200);
        responseEvent.setBody(request.toString());

        return responseEvent;
    }
}
```

6. Before we proceed let's test this function and check the output in the browser. Therefore, we need to export this code are a single JAR file that we can upload to AWS lambda. To do this all you have to do is run a "Maven build" as depicted in the diagram below *(the first time you run this build you will need to configure it, see step 7)*.

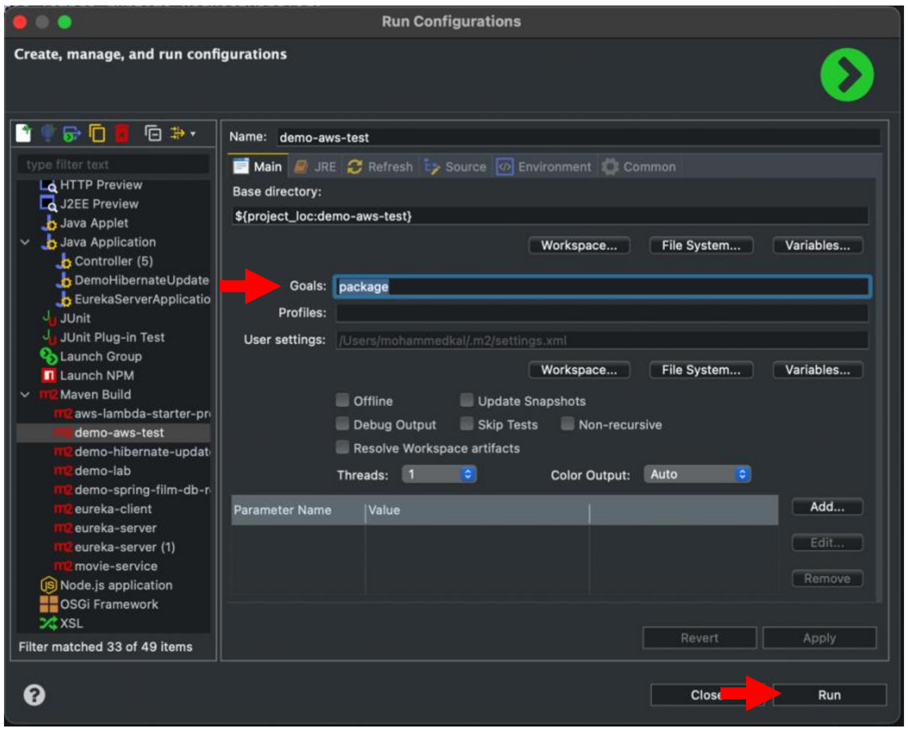
   Right click the project, **Run as->Maven Build**.



```
package lambda_functions;

import com.amazonaws.services.
import com.amazonaws.services.
import com.amazonaws.services.
import com.amazonaws.services.

public class HelloWorld imple

    public APIGatewayProxyResp

        APIGatewayProxyRespons

        // TODO: do some awes

        responseEvent.setStatu
        responseEvent.setBody(

        return responseEvent;
    }
}
```

7. The first time you run a maven build you will have to configure it so that it includes all the dependencies into the single application jar that it creates.

   The only configuration to have to do is add “package” to the Goals and hit the “**Run**” button.



Run Configurations

Create, manage, and run configurations

Name: demo-aws-test

Main | JRE | Refresh | Source | Environment | Common

Base directory:
${project_loc:demo-aws-test}

Goals: package

Profiles:

User settings: /Users/mohammedkal/.m2/settings.xml

Offline | Update Snapshots | Debug Output | Skip Tests | Non-recursive | Resolve Workspace artifacts

Threads: 1 Color Output: Auto

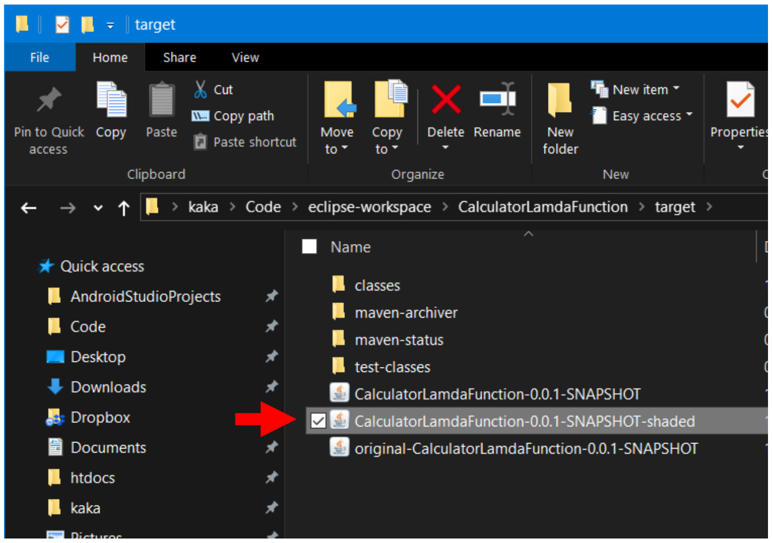
Parameter Name	Value

type filter text

- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
  - Controller (5)
  - DemoHibernateUpdate
  - EurekaServerApplicatio
- JUnit
- JUnit Plug-in Test
- Launch Group
- Launch NPM
- Maven Build
  - aws-lambda-starter-pr
  - demo-aws-test
  - demo-hibernate-updat
  - demo-lab
  - demo-spring-film-db-r
  - eureka-client
  - eureka-server
  - eureka-server (1)
  - movie-service
- Node.js application
- OSGi Framework
- XSL

Filter matched 33 of 49 items

This will generate a single JAR file which includes all the required dependencies used within the project. The file will be in the **“target”** folder in you working project folder (As shown below).

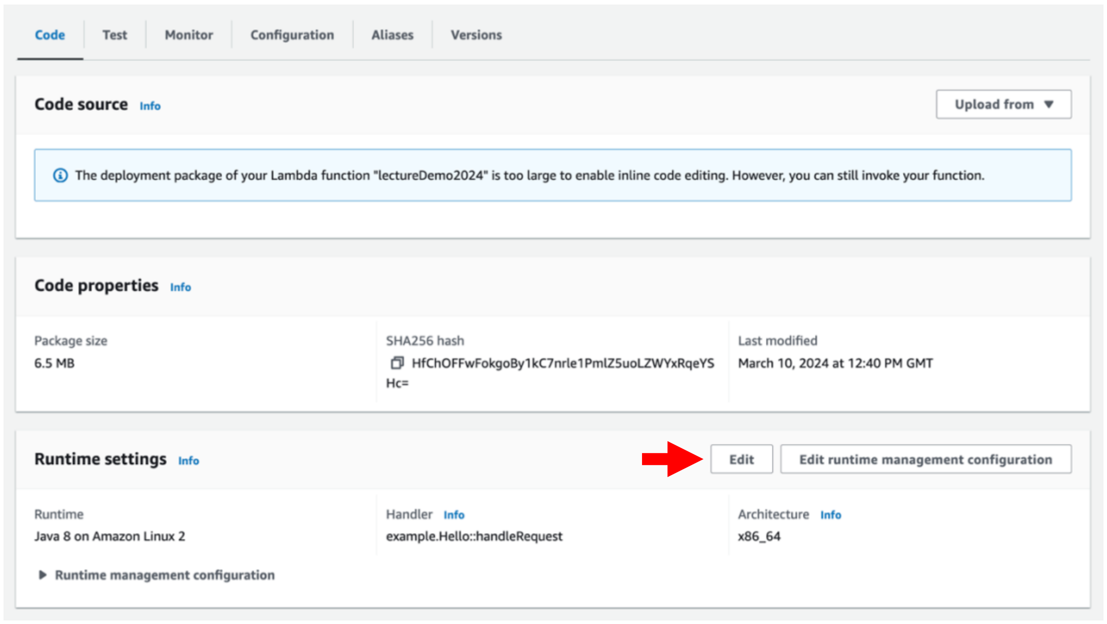


The JAR file that you need to upload to AWS is the file that is larger in size, this file has all the required dependencies package along with it also, the small file is just your application code (no dependencies).

8. In AWS create a new lambda function called "HelloWorldJava".

9. Upload the **JAR** file by clicking on the upload button and selecting the JAR file you created in step 6,

10. Then edit the handler which is the name of the java package followed by the name of the class where the handler function is implemented (package.class_name::handleFunction), then click the "**save**" button.


Code
Test
Monitor
Configuration
Aliases
Versions
Upload from
Info
Edit
Edit runtime management configuration
Runtime management configuration

Code source

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

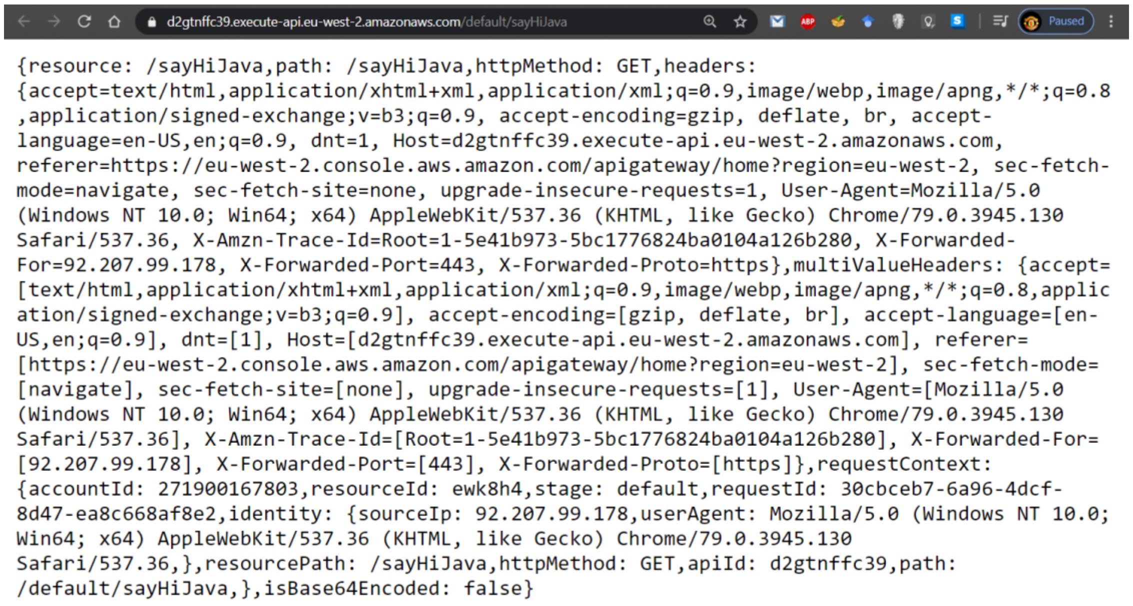
Package size	SHA256 hash	Last modified
6.5 MB	HfChOFFwFokgoBy1kC7nrle1PmlZ5uoLZWYxRqeYSHc=	March 10, 2024 at 12:40 PM GMT

Runtime settings

Runtime	Handler	Architecture
Java 8 on Amazon Linux 2	example.Hello::handleRequest	x86_64

11. Add an API gateway trigger this time we are using “Lambda Proxy Integration” (instead of mapping all the individual request parameters) so there is no need to do any json mapping templates.

12. Deploy the API and test it in a browser. You should see the following output:



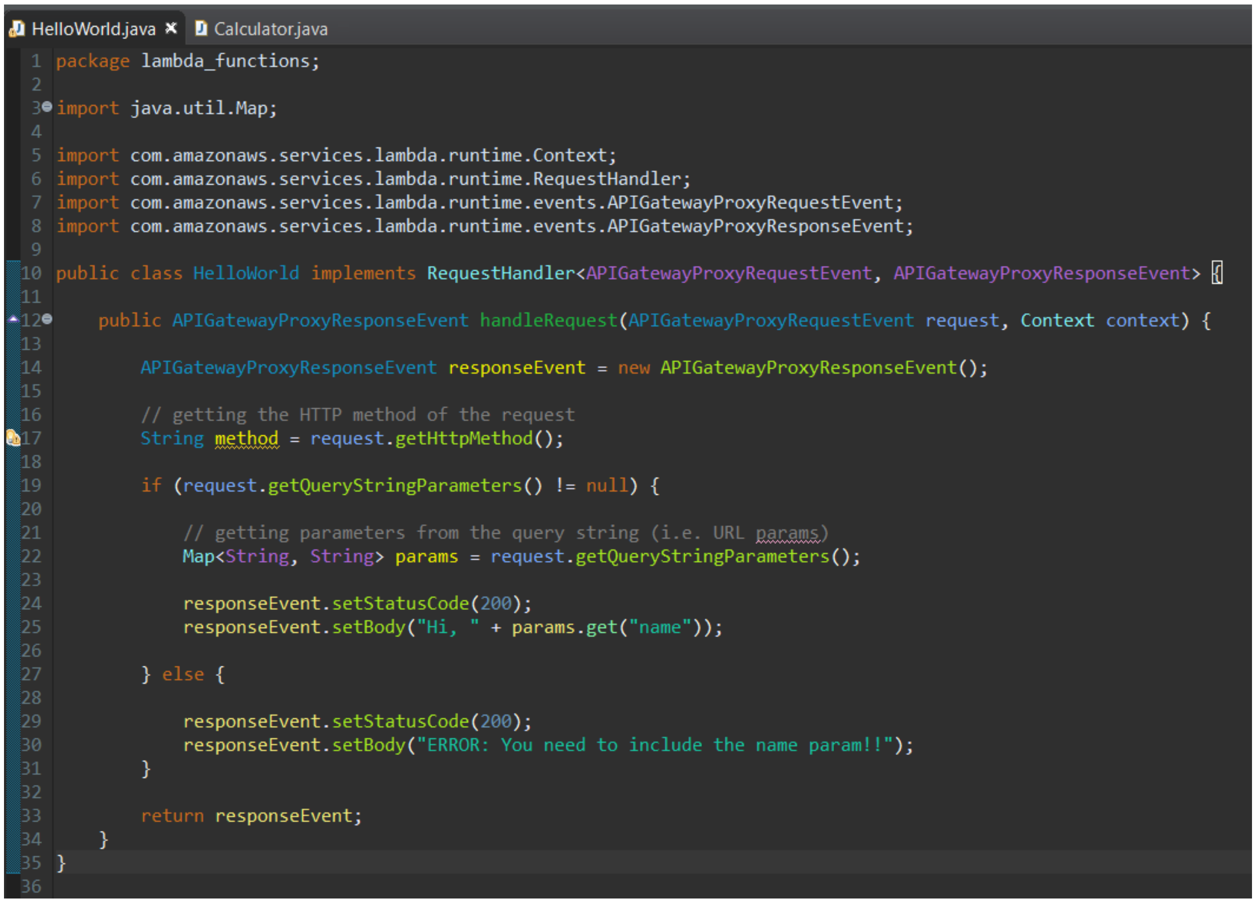
d2gtnffc39.execute-api.eu-west-2.amazonaws.com/default/sayHiJava

```
{resource: /sayHiJava,path: /sayHiJava,httpMethod: GET,headers:
{accept=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
,application/signed-exchange;v=b3;q=0.9, accept-encoding=gzip, deflate, br, accept-
language=en-US,en;q=0.9, dnt=1, Host=d2gtnffc39.execute-api.eu-west-2.amazonaws.com,
referer=https://eu-west-2.console.aws.amazon.com/apigateway/home?region=eu-west-2, sec-fetch-
mode=navigate, sec-fetch-site=none, upgrade-insecure-requests=1, User-Agent=Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130
Safari/537.36, X-Amzn-Trace-Id=Root=1-5e41b973-5bc1776824ba0104a126b280, X-Forwarded-
For=92.207.99.178, X-Forwarded-Port=443, X-Forwarded-Proto=https},multiValueHeaders: {accept=
[text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,applic
ation/signed-exchange;v=b3;q=0.9], accept-encoding=[gzip, deflate, br], accept-language=[en-
US,en;q=0.9], dnt=[1], Host=[d2gtnffc39.execute-api.eu-west-2.amazonaws.com], referer=
[https://eu-west-2.console.aws.amazon.com/apigateway/home?region=eu-west-2], sec-fetch-mode=
[navigate], sec-fetch-site=[none], upgrade-insecure-requests=[1], User-Agent=[Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130
Safari/537.36], X-Amzn-Trace-Id=[Root=1-5e41b973-5bc1776824ba0104a126b280], X-Forwarded-For=
[92.207.99.178], X-Forwarded-Port=[443], X-Forwarded-Proto=[https]},requestContext:
{accountId: 271900167803,resourceId: ewk8h4,stage: default,requestId: 30cbceb7-6a96-4dcf-
8d47-ea8c668af8e2,identity: {sourceIp: 92.207.99.178,userAgent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130
Safari/537.36,},resourcePath: /sayHiJava,httpMethod: GET,apiId: d2gtnffc39,path:
/default/sayHiJava,},isBase64Encoded: false}
```

This is the entire request/event object. Try adding some parameters to the URL, you will then see that they will be added to the output mapped to the queryStringParameters object (for a GET request with a URL parameters).

Task 2 – Implement the hello world example

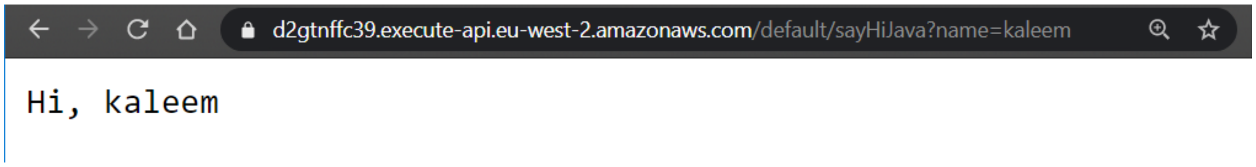
1. Amend the function so that it accepts a URL parameter called name and then returns a greeting message to that supplied name parameter (code example below).



HelloWorld.java Calculator.java

```java
package lambda_functions;

import java.util.Map;

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 request, Context context) {

        APIGatewayProxyResponseEvent responseEvent = new APIGatewayProxyResponseEvent();

        // getting the HTTP method of the request
        String method = request.getHttpMethod();

        if (request.getQueryStringParameters() != null) {

            // getting parameters from the query string (i.e. URL params)
            Map<String, String> params = request.getQueryStringParameters();

            responseEvent.setStatusCode(200);
            responseEvent.setBody("Hi, " + params.get("name"));

        } else {

            responseEvent.setStatusCode(200);
            responseEvent.setBody("ERROR: You need to include the name param!!");
        }

        return responseEvent;
    }
}
```

2. Upload the function and test:



d2gtnffc39.execute-api.eu-west-2.amazonaws.com/default/sayHiJava?name=kaleem

```
Hi, kaleem
```

Task 3 – Implement the calculator

1. Implement a cloud function that takes 3 arguments (num1, num2, and operator).
2. Upload, deploy and test it.

Task 4 – Attempt the extension task on Moodle