Lab 11 AWS Lambda

Activities:

- Create a HelloWorld serverless function with AWS Lambda
- Create a HelloWorld Lambda function with Eclipse IDE

Task 1: Create a HelloWorld serverless function with AWS Lambda

Introduction

In this tutorial, you will learn the basics of running code on AWS Lambda without provisioning or managing servers. We will walk through how to create a Hello World Lambda function using the AWS Lambda console. We will then show you how to manually invoke the Lambda function using sample event data and review your output metrics.

Everything done in this tutorial is free tier eligible.

Step 1: Enter the Lambda Console

When you click here, the AWS Management Console will open in a new browser window, so you can keep this step-by-step guide
open. Find Lambda under *Compute* and click to open the AWS Lambda Console.

Step 2: Select a Lambda Blueprint

Blueprints provide example code to do some minimal processing. Most blueprints process events from specific event sources, such as Amazon S3, DynamoDB, or a custom application.

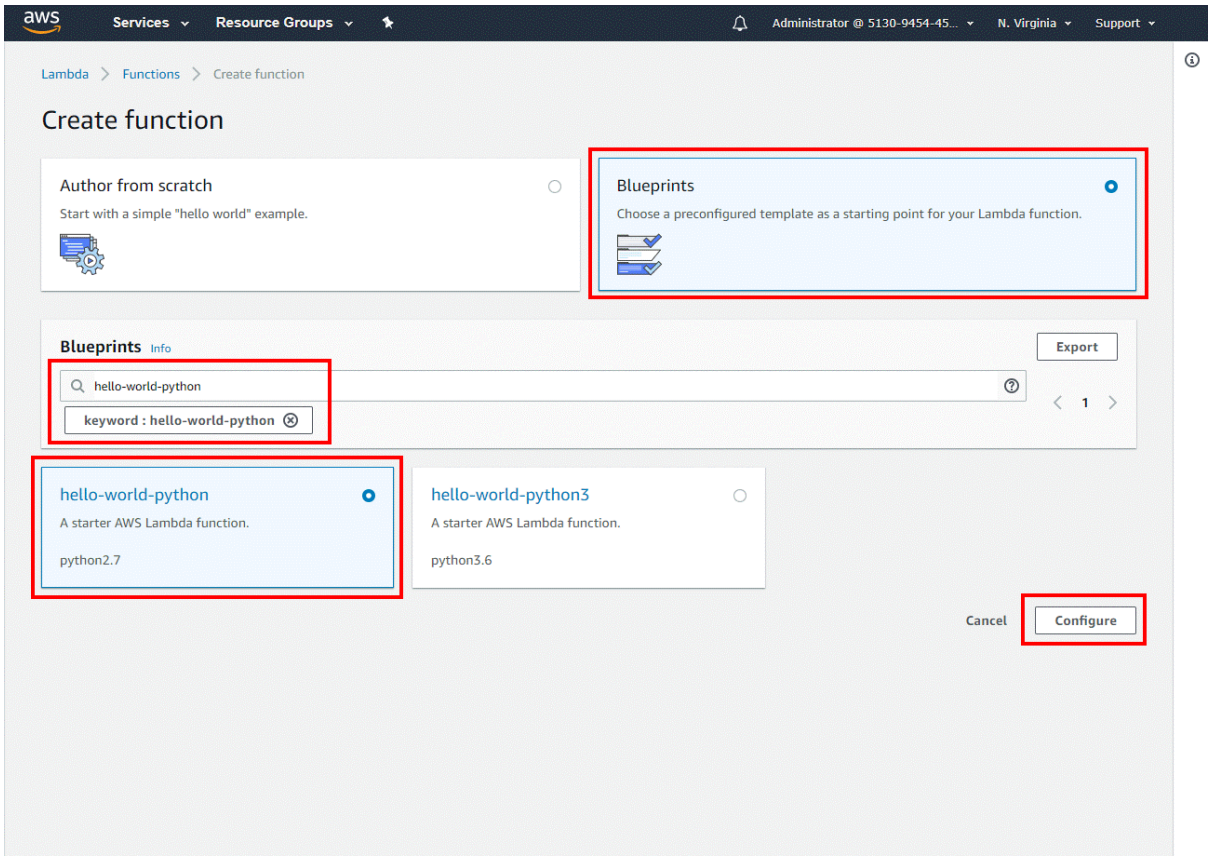
a. In the AWS Lambda console, select Create a Function.

Note:
The console shows this page only if you do not have any Lambda functions created. If you have created functions already, you will see the *Lambda* > *Functions* page. On the list page, choose *Create a function* to go to the Create function page.

b. Select Blueprints.

c. In the Filter box, type in *hello-world-python* and select the hello-world-python blueprint.

d. Then click Configure.



Step 3: Configure and Create Your Lambda Function

A Lambda function consists of code you provide, associated dependencies, and configuration. The configuration information you provide includes the compute resources you want to allocate (for example, memory), execution timeout, and an IAM role that AWS Lambda can assume to execute your Lambda function on your behalf.

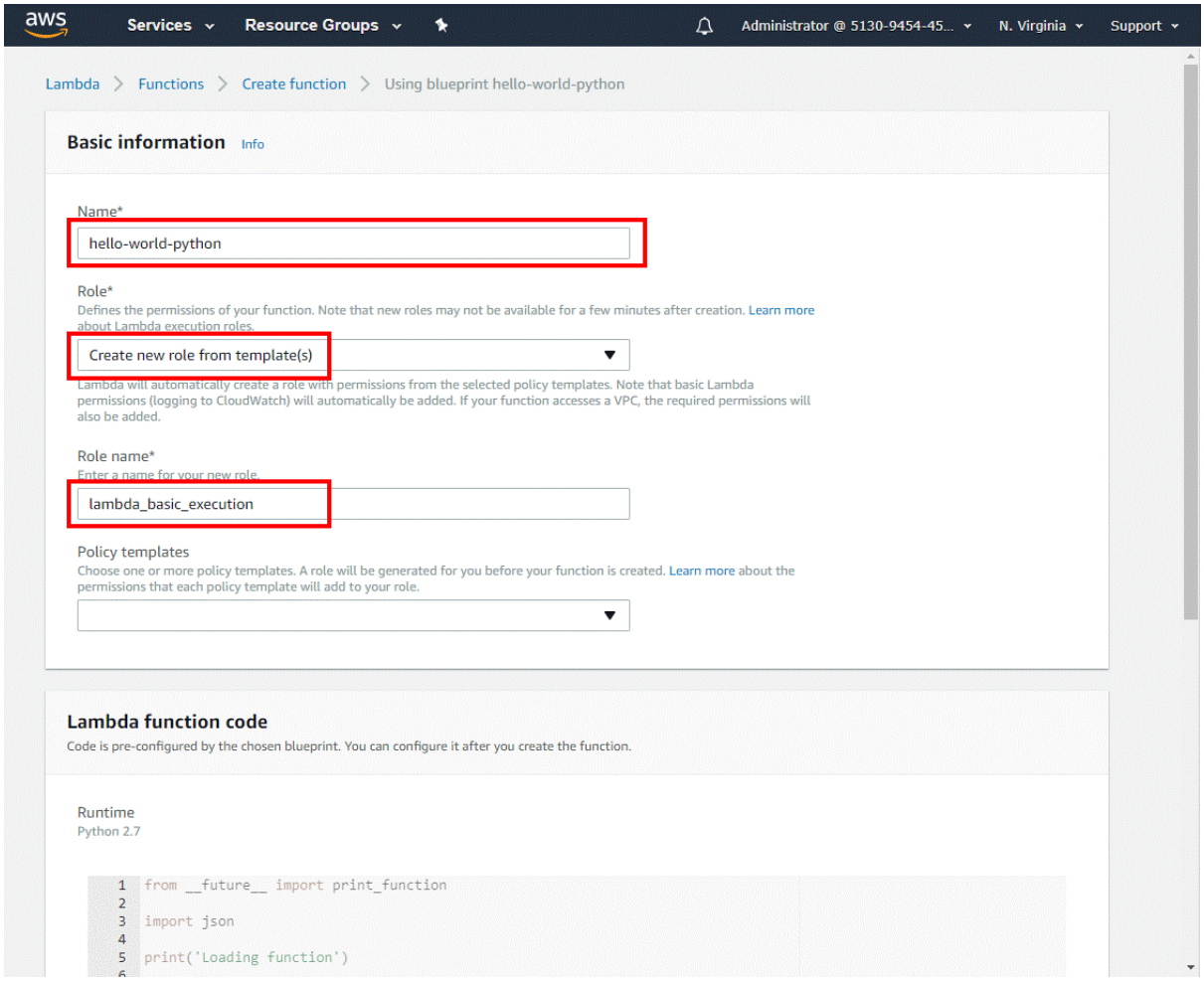
a. You will now enter *Basic Information* about your Lambda function.

Basic Information:

- Name: You can name your Lambda function here. For this tutorial, enter *hello-world-python*.
- Role: You will create an IAM role (referred as the execution role) with the necessary permissions that AWS Lambda can assume to invoke your Lambda function on your behalf. Select *Create new role from template(s).*

- Role name: type *lambda_basic_execution*
  Lambda Function Code:
- In this section, you can review the example code authored in Python.

b. Go to the bottom of the page and select Create Function.



c. Runtime: Currently, you can author your Lambda function code in Java, Node.js, C#, Go or Python. For this tutorial, leave this on *Python 2.7* as the runtime.

d. Handler: You can specify a handler (a method/function in your code) where AWS Lambda can begin executing your code. AWS Lambda provides event data as input to this handler, which processes the event.

In this example, Lambda identifies this from the code sample and this should be pre-populated with *lambda_function.lambda_handler*.

e. Scroll down to configure your memory, timeout, and VPC settings. For this tutorial, leave the default Lambda function configuration values.

Step 4: Invoke Lambda Function and Verify Results

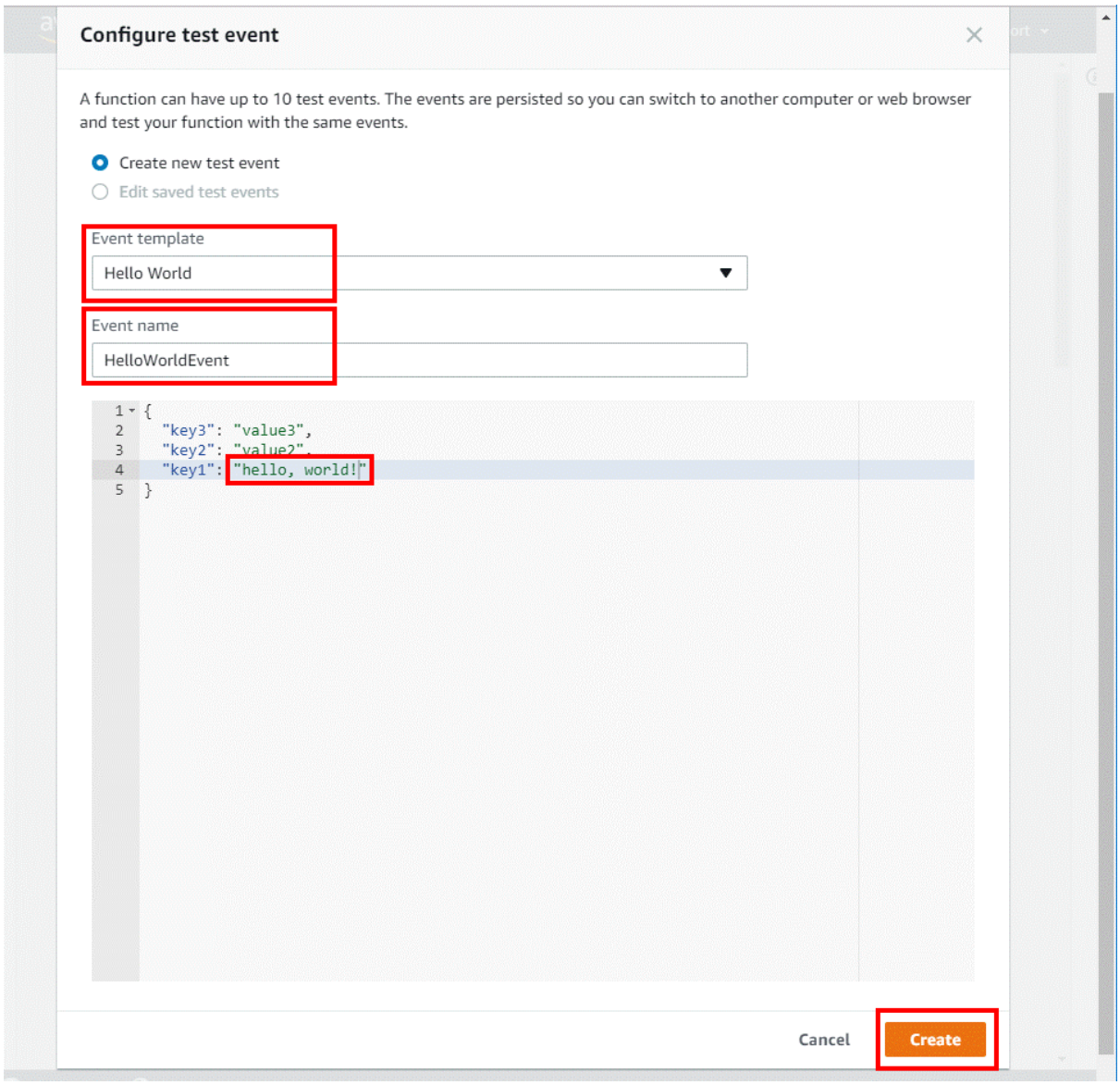
The console shows the hello-world-python Lambda function - you can now test the function, verify results, and review the logs.

a. Select Configure Test Event from the drop-down menu called "Select a test event...".

b. The editor pops up to enter an event to test your function.

- Choose *Hello World* from the Sample event template list from the Input test event page.
- Type in an event name like *HelloWorldEvent.*
- You can change the values in the sample JSON, but don't change the event structure. For this tutorial, replace *value1* with *hello, world!.*

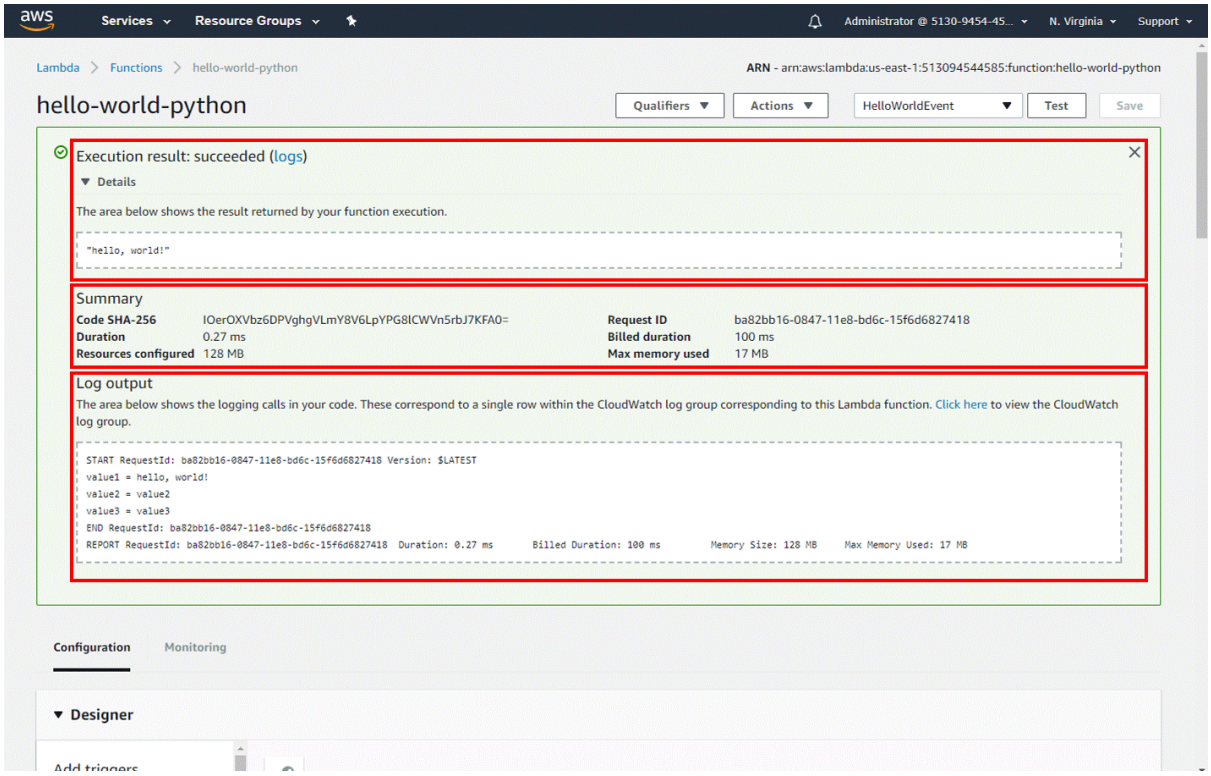
Select Create.



c. Select Test.

d. Upon successful execution, view the results in the console:

- The Execution results section verifies that the execution succeeded.
- The Summary section shows the key information reported in the Log output.
- The Log output section will show the logs generated by the Lambda function execution.



Step 5: Monitor Your Metrics

AWS Lambda automatically monitors Lambda functions and reports metrics through Amazon CloudWatch. To help you monitor your code as it executes, Lambda automatically tracks the number of requests, the latency per request, and the number of requests resulting in an error and publishes the associated metrics.

a. Invoke the Lambda function a few more times by repeatedly clicking the Test button. This will generate the metrics that can be viewed in the next step.

b. Select Monitoring to view the results.

c. Scroll down to view the metrics for your Lambda function. Lambda metrics are reported through Amazon CloudWatch. You can leverage these metrics to set custom alarms. For more information about CloudWatch, see the Amazon CloudWatch Developer Guide.

The Monitoring tab will show six CloudWatch metrics: *Invocation count, Invocation duration, Invocation errors*, *Throttled invocations, Iterator age, and DLQ errors.*

With AWS Lambda, you pay for what you use. After you hit your AWS Lambda free tier limit, you are charged based on the number of requests for your functions (invocation count) and the time your code executes (invocation duration). For more information, see AWS Lambda Pricing.

Step 6: Delete the Lambda Function

While you will not get charged for keeping your Lambda function, you can easily delete it from the AWS Lambda console.

a. Select the Actions button and click Delete Function.

b. You will be asked to confirm your termination - select Delete.

Task 2: Create a HelloWorld Lambda function with Eclipse IDE

Introduction

This tutorial guides you through the process of a typical AWS Lambda workflow, and provides you with first-hand experience using Lambda with the AWS Toolkit for Eclipse.

The tutorial assumes that you have an AWS account, have already installed the AWS Toolkit for Eclipse, and that you understand the basic concepts and features of Lambda.

Step 0: Java 8 Runtime Environment Setup

This project requires JAVA 8 runtime environment.

To set JAVA 8 Runtime Environment in Eclipse

1. Download and install the latest version of JAVA 8 JDK from https://www.oracle.com/java/technologies/javase-jdk8-downloads.html
2. Click on the Window tab in Eclipse, go to Preferences → Java → Installed JREs → Search → go to your JRE1.8 installation directory → Select Folder → Check "jre1.8.0" → Apply and Close.

Step 1: Create an AWS Lambda Project

To begin a Lambda project, you first implement the code as a method in a handler class. The AWS Toolkit for Eclipse provides a new project wizard to help you create a new handler class. The Lambda project is a Maven project that uses a POM.xml file to manage package dependencies. You can use the Maven command line tool for building, testing, and deploying your application. For more information about Maven, see the Maven project documentation.

To create an AWS Lambda project

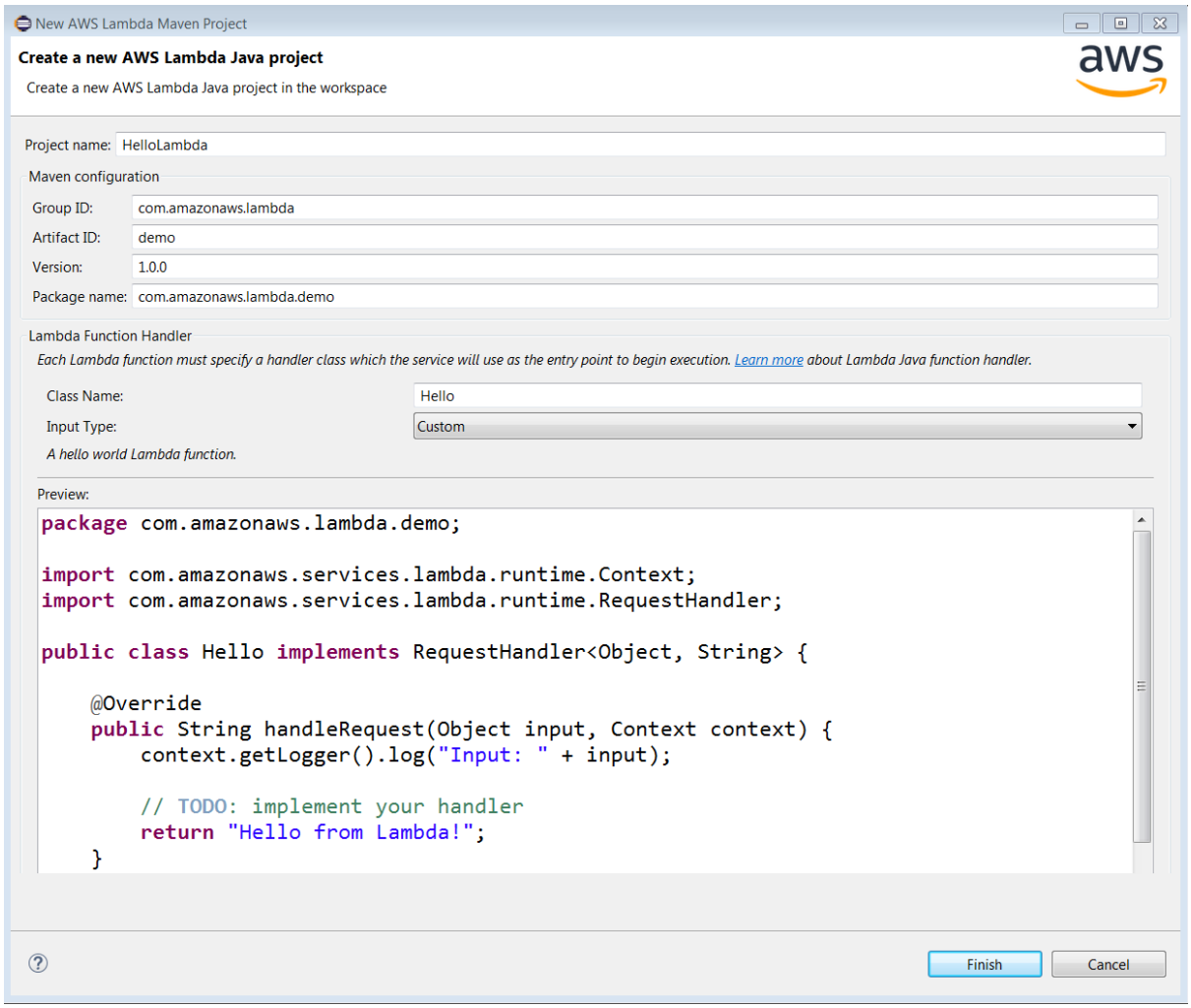
1. On the Eclipse toolbar, open the Amazon Web Services menu (identified by the AWS homepage icon), and then choose **New AWS Lambda Java project**. Or on the Eclipse menu bar, choose **File**, **New**, **AWS Lambda Java Project**.
2. Add a *Project name*, *Group ID*, *Artifact ID*, and *class name* in the associated input boxes. The Group ID and Artifact ID are the IDs that identify a Maven build artifact. This tutorial uses the following example values:

- **Project name**: *HelloLambda*
- **Group ID**: *com.amazonaws.lambda*

- **Artifact ID**: *demo*
- **Class name**: *Hello*

  The **Package Name** field is the package namespace for the AWS Lambda handler class. The default value for this field is a concatination of the Group ID and Artifact ID, following Maven project conventions. This field is automatically updated when the **Group ID** and **Artifact ID** fields are updated.

3. For **Input Type**, choose **Custom**. For information about each of the available input types, see New AWS Lambda Java Project Dialog.
4. Verify that your entries look like the following screenshot (modify them if they are not), and then choose **Finish**.



   As you type, the code in the **Source preview** changes to reflect the changes you make in the dialog box.

5. After you choose **Finish**, your project's directory and source files are generated in your Eclipse workspace. A new web browser window opens, displaying `README.html` (which was created for you in your project's root directory). `README.html` provides instructions

to guide you through the next steps of implementing, testing, uploading, and invoking your new Lambda function. Read through it to gain some familiarity with the steps that are described here.

Next, you implement the function in the `HelloLambda` Java project that was just created for you in Eclipse.

Step 2: Implement the Handler Method

You use the **Create New Project** dialog box to create a skeleton project. Now fill in the code that will be run when your Lambda function is invoked. (In this case, by a custom event that sends a String to your function, as you specified when setting your method's input parameter.)

To implement your Lambda handler method

1. In the Eclipse **Project Explorer**, open `Hello.java` in the **HelloLambda** project. It will contain code similar to the following.

```
package com.amazonaws.lambda.demo;

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

public class Hello implements RequestHandler<Object, String> {

    @Override
    public String handleRequest(Object input, Context context) {
        context.getLogger().log("Input: " + input);

        // TODO: implement your handler
        return "Hello from Lambda";
    }

}
```

2. Replace the contents of the `handleRequest` function with the following code.

```
@Override
public String handleRequest(Object input, Context context) {
  context.getLogger().log("Input: " + input);
```

```
  String output = "Hello, " + input + "!";
  return output;
}
```

Step 3: Allow Lambda to Assume an IAM Role

For Lambda to be able to access your Lambda function, you have to create an IAM role that gives it access to your AWS resources. You can create the role in two ways, either through the AWS Management Console or by using the AWS Toolkit for Eclipse. This section describes how to create the IAM role in the console.
See Upload the Code to create one using the AWS Toolkit for Eclipse.

To create an IAM role for Lambda

1. Sign in to the AWS Management Console.
2. From the **Services** menu, open the IAM console.
3. In the Navigation pane, choose **Roles**, and then choose **Create role**.
4. For **Select type of trusted entity**, choose **AWS service**, and then choose **Lambda** for the service that will use this role. Then choose **Next: Permissions**.
5. For **Attach permissions policy**, choose **AWSLambdaBasicExecutionRole**. This allows Lambda to write to your CloudWatch Logs resources. Then choose **Next: Review**.
6. Add a name for your role, such as `hello-lambda-role`, and a description for the role. Then choose **Create role** to finish creating the IAM role.

Step 4: Create an Amazon S3 Bucket for Your Lambda Code

AWS Lambda requires an Amazon S3 bucket to store your Java project when you upload it. You can either use a bucket that already exists in the AWS Region in which you'll run your code, or you can create a new one specifically for Lambda to use (recommended).

You can create an Amazon S3 bucket in two ways, either through the AWS Management Console or by using the AWS Toolkit for Eclipse. This section describes how to create an Amazon S3 bucket in the console. See Upload the Code to create one using the AWS Toolkit for Eclipse.

To create an Amazon S3 bucket for use with Lambda

1. Sign in to the AWS Management Console.
2. From the **Services** menu, open the S3 console.

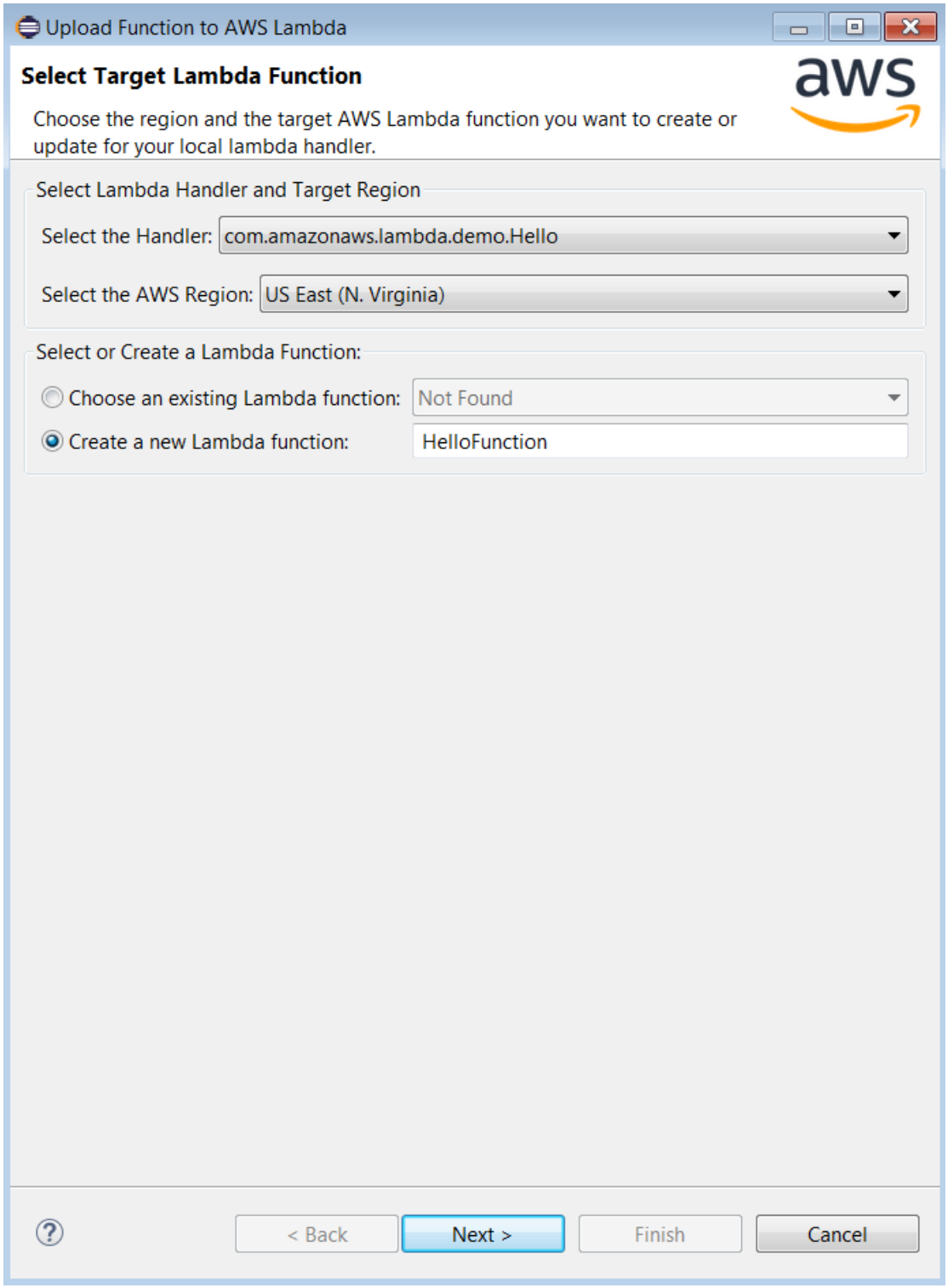
3. Choose **Create bucket**.
4. Enter a bucket name, and then choose a region for your bucket. This region should be the same one in which you intend to run your Lambda function. For a list of regions supported by Lambda see Regions and Endpoints in the Amazon Web Services General Reference.
5. Choose **Create** to finish creating your bucket.

Step 5: Upload the Code

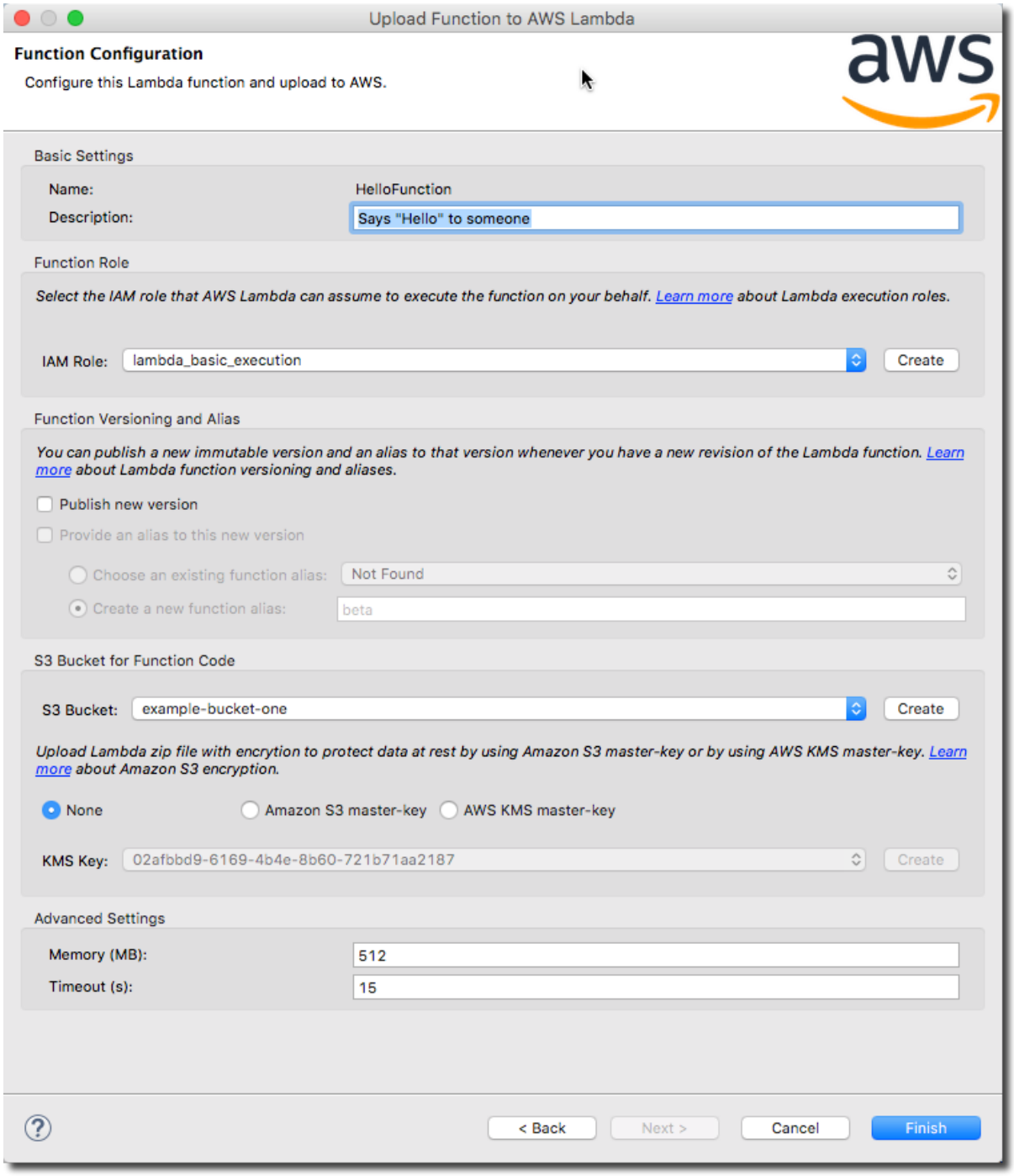
Next, you upload your code to AWS Lambda in preparation for invoking it using the AWS Management Console.

To upload your function to Lambda

1. Right-click in your Eclipse code window, choose **AWS Lambda**, and then choose **Upload function to AWS Lambda**.
2. On the **Select Target Lambda Function** page, choose the AWS Region to use. This should be the same region that you chose for your Amazon S3 bucket.

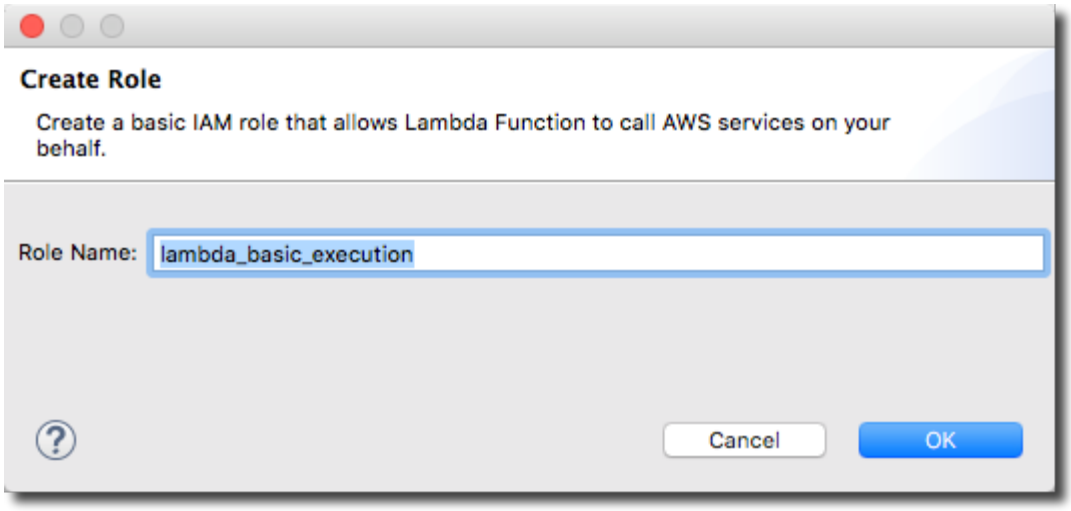


3. Choose **Create a new Lambda function**, and then type a name for your function (for example, `HelloFunction`).
4. Choose **Next**.
5. On the **Function Configuration** page, enter a description for your target Lambda function, and then choose the IAM role and Amazon S3 bucket that your function will use.

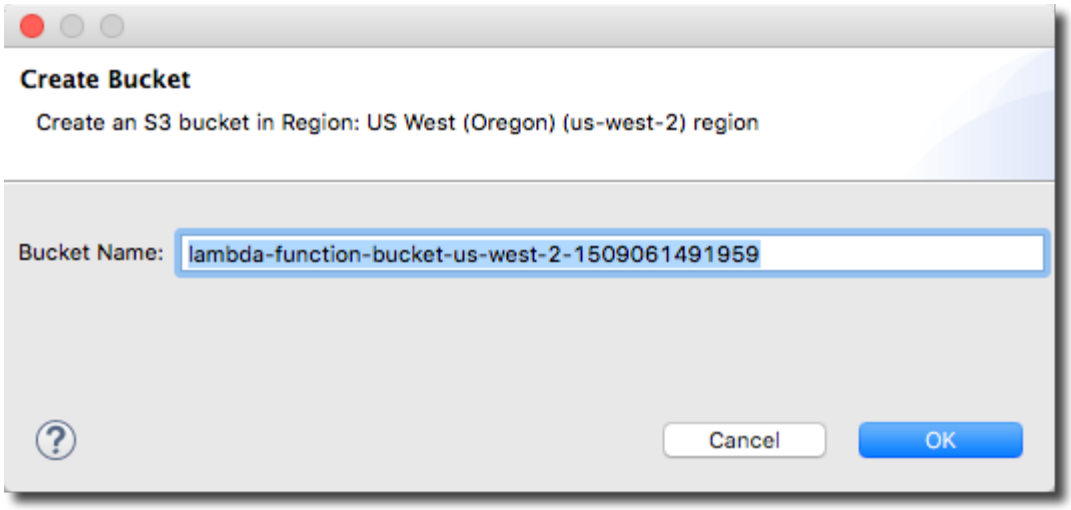


For more information about the available options, see Upload Function to AWS Lambda Dialog Box.

6. On the **Function Configuration** page, choose **Create** in **Function Role** if you want to create a new IAM role for your Lambda function. Enter a role name in the dialogue box the **Create Role** dialogue box.



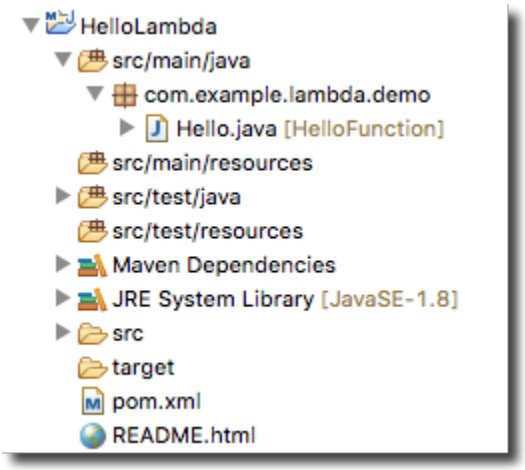
7. On the **Function Configuration** page, choose **Publish new version** if you want the upload to create a new version of the Lambda function. To learn more about versioning and aliases in Lambda, see AWS Lambda Function Versioning and Aliases in the AWS Lambda Developer Guide.
8. If you chose to publish a new version, the **Provide an alias to this new version** option is enabled. Choose this option if you want to associate an alias with this version of the Lambda function.
9. On the **Function Configuration** page, choose **Create** in the **S3 Bucket for Function Code** section if you want to create a new Amazon S3 bucket for your Lambda function. Enter a bucket name in the **Create Bucket** dialogue box.



10. In the **S3 Bucket for Function Code** section, you can also choose to encrypt the uploaded code. For this example, leave **None** selected. To learn more about Amazon S3 encryption, see Protecting Data Using Server-Side Encryption in the Amazon S3 Developer Guide.

11. Leave the **Advanced Settings** options as they are. The AWS Toolkit for Eclipse selects default values for you. Choose **Finish** to upload your Lambda function to AWS.

    If the upload succeeds, you will see the Lambda function name that you chose appear next to your Java handler class in the **Project Explorer** view.



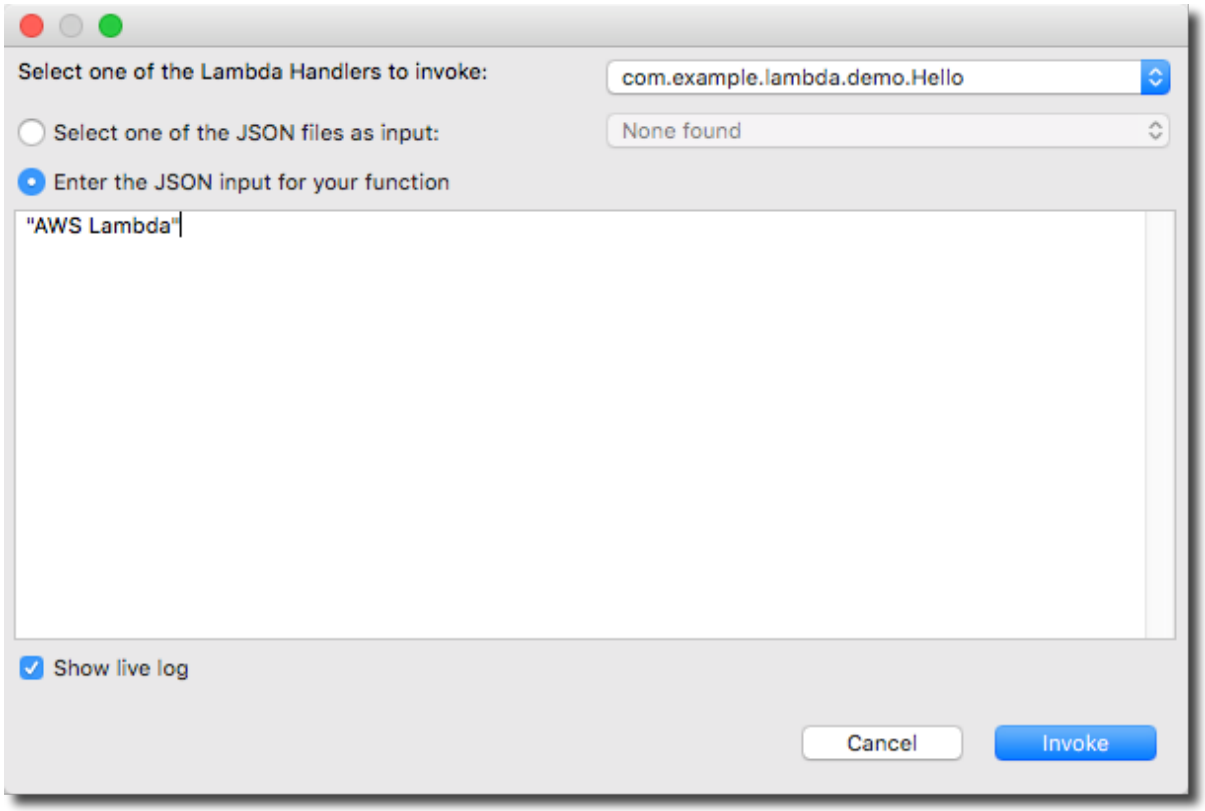
    If you don't see this happen, open the Eclipse **Error Log** view. Lambda writes information about failures to upload or run your function to this error log so you can debug them.

Step 6: Invoke the Lambda Function

You can now invoke the function on AWS Lambda.

To invoke your Lambda function

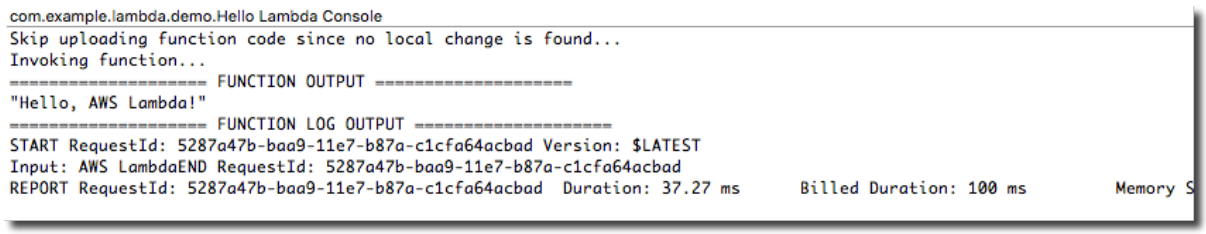
1. Right-click in the Eclipse code window, choose **AWS Lambda**, and then choose **Run Function on AWS Lambda**.
2. Choose the handler class you want to invoke.
3. In the input box, type a valid JSON string, such as "AWS Lambda".



Note

You can add new JSON input files to your project, and they will show up in this dialog box if the file name ends with .json. You can use this feature to provide standard input files for your Lambda functions.

4. The **Show Live Log** box is checked by default. This displays the logs from the Lambda function output in the Eclipse **Console**.
5. Choose **Invoke** to send your input data to your Lambda function. If you have set up everything correctly, you should see the return value of your function printed out in the Eclipse **Console** view (which automatically appears if it isn't already shown).



Congratulations, you've just run your first Lambda function directly from the Eclipse IDE!