

Quarkus - Funqy Amazon Lambda

The guide walks through quickstart code to show you how you can deploy Funqy functions to Amazon Lambda.

Funqy functions can be deployed using the Amazon Java Runtime, or you can build a native executable and use Amazon's Custom Runtime if you want a smaller memory footprint and faster cold boot startup time.



This technology is considered preview.

In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

Prerequisites

To complete this guide, you need:

- less than 30 minutes
- JDK 11 (AWS requires JDK 1.8 or 11)
- Apache Maven 3.6.3
- [An Amazon AWS account](#)
- [AWS CLI](#)
- [AWS SAM CLI](#), for local testing



Funqy Amazon Lambdas build off of our [Quarkus Amazon Lambda support](#).

Installing AWS bits

Installing all the AWS bits is probably the most difficult thing about this guide. Make sure that you follow all the steps for installing AWS CLI.

The Quickstart

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the `funqy-amazon-lambda-quickstart` directory.

The Code

There is nothing special about the code and more importantly nothing AWS specific. Funqy functions can be deployed to many different environments and AWS Lambda is one of them. The Java code is actually the same exact code as the [funqy-http-quickstart](#).

Choose Your Function

Only one Funqy function can be exported per Amazon Lambda deployment. If you only have one method annotated with `@Funq` in your project, then there is no worries. If you have multiple functions defined within your project, then you will need to choose the function within your Quarkus `application.properties`:

```
quarkus.funqy.export=greet
```

You can see how the quickstart has done it within its own [application.properties](#).

Alternatively, you can set the `QUARKUS_FUNQY_EXPORT` environment variable when you create the Amazon Lambda using the `aws` cli.

Deploy to AWS Lambda Java Runtime

There are a few steps to get your Funqy function running on AWS Lambda. The quickstart maven project generates a helpful script to create, update, delete, and invoke your functions for pure Java and native deployments. This script is generated at build time.

Build and Deploy

Build the project using maven.

```
./mvnw clean package
```

This will compile and package your code.

Create an Execution Role

View the [Getting Started Guide](#) for deploying a lambda with AWS CLI. Specifically, make sure you have created an `Execution Role`. You will need to define a `LAMBDA_ROLE_ARN` environment variable in your profile or console window, Alternatively, you can edit the `manage.sh` script that is generated by the build and put the role value directly there:

```
LAMBDA_ROLE_ARN="arn:aws:iam::1234567890:role/lambda-role"
```

Extra Build Generated Files

After you run the build, there are a few extra files generated by the `quarkus-funqy-amazon-lambda` extension. These files are in the the build directory: `target/` for maven, `build/` for gradle.

- `function.zip` - lambda deployment file
- `manage.sh` - wrapper around aws lambda cli calls
- `bootstrap-example.sh` - example bootstrap script for native deployments
- `sam.jvm.yaml` - (optional) for use with sam cli and local testing
- `sam.native.yaml` - (optional) for use with sam cli and native local testing

Create the function

The `target/manage.sh` script is for managing your Funqy function using the AWS Lambda Java runtime. This script is provided only for your convenience. Examine the output of the `manage.sh` script if you want to learn what aws commands are executed to create, delete, and update your functions.

`manage.sh` supports four operation: `create`, `delete`, `update`, and `invoke`.



To verify your setup, that you have the AWS CLI installed, executed `aws configure` for the AWS access keys, and setup the `LAMBDA_ROLE_ARN` environment variable (as described above), please execute `manage.sh` without any parameters. A usage statement will be printed to guide you accordingly.

To see the `usage` statement, and validate AWS configuration:

```
sh target/manage.sh
```

You can `create` your function using the following command:

```
sh target/manage.sh create
```

or if you do not have `LAMBDA_ROLE_ARN` already defined in this shell:

```
LAMBDA_ROLE_ARN="arn:aws:iam::1234567890:role/lambda-role" sh  
target/manage.sh create
```



Do not change the handler switch. This must be hardcoded to `io.quarkus.funqy.lambda.FunqyStreamHandler::handleRequest`. This special handler is Funqy's integration point with AWS Lambda.

If there are any problems creating the function, you must delete it with the `delete` function before re-running the `create` command.

```
sh target/manage.sh delete
```

Commands may also be stacked:

```
sh target/manage.sh delete create
```

Invoke the function

Use the `invoke` command to invoke your function.

```
sh target/manage.sh invoke
```

The example function takes input passed in via the `--payload` switch which points to a json file in the root directory of the project.

The function can also be invoked locally with the SAM CLI like this:

```
sam local invoke --template target/sam.jvm.yaml --event  
payload.json
```

If you are working with your native image build, simply replace the template name with the native version:

```
sam local invoke --template target/sam.native.yaml --event  
payload.json
```

Update the function

You can update the Java code as you see fit. Once you've rebuilt, you can redeploy your function by executing the `update` command.

```
sh target/manage.sh update
```

Deploy to AWS Lambda Custom (native) Runtime

If you want a lower memory footprint and faster initialization times for your Funqy function, you can compile your Java code to a native executable. Just make sure to rebuild your project with the `-Pnative` switch.

For Linux hosts execute:

```
mvn package -Pnative
```



If you are building on a non-Linux system, you will need to also pass in a property instructing Quarkus to use a docker build as Amazon Lambda requires linux binaries. You can do this by passing this property to your Maven build: `-Dnative-image.docker-build=true`, or for Gradle: `--docker-build=true`. This requires you to have docker installed locally, however.

```
./mvnw clean install -Pnative -Dnative-image.docker-build=true
```

Either of these commands will compile and create a native executable image. It also generates a zip file `target/function.zip`. This zip file contains your native executable image renamed to `bootstrap`. This is a requirement of the AWS Lambda Custom (Provided) Runtime.

The instructions here are exactly as above with one change: you'll need to add `native` as the first parameter to the `manage.sh` script:

```
sh target/manage.sh native create
```

As above, commands can be stacked. The only requirement is that `native` be the first parameter should you wish to work with native image builds. The script will take care of the rest of the details necessary to manage your native image function deployments.

Examine the output of the `manage.sh` script if you want to learn what aws commands are executed to create, delete, and update your functions.

One thing to note about the create command for native is that the `aws lambda create-function` call must set a specific environment variable:

```
--environment 'Variables={DISABLE_SIGNAL_HANDLERS=true}'
```

Examine the POM

There is nothing special about the POM other than the inclusion of the `quarkus-funqy-amazon-lambda` and `quarkus-test-amazon-lambda` extensions as a dependencies. The extension automatically generates everything you might need for your lambda deployment.

Integration Testing

Funqy Amazon Lambda support leverages the Quarkus AWS Lambda test framework so that you can unit tests your Funqy functions. This is true for both JVM and native modes. This test framework provides similar functionality to the SAM CLI, without the overhead of Docker.

If you open up `FunqyTest.java` you'll see that the test replicates the AWS execution environment.

```
package org.acme.funqy;

import io.quarkus.amazon.lambda.test.LambdaClient;
import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

@QuarkusTest
public class FunqyTest {
    @Test
    public void testSimpleLambdaSuccess() throws Exception {
        Friend friend = new Friend("Bill");
        Greeting out = LambdaClient.invoke(Greeting.class, friend);
        Assertions.assertEquals("Hello Bill", out.getMessage());
    }
}
```

Testing with the SAM CLI

The [AWS SAM CLI](#) allows you to run your functions locally on your laptop in a simulated Lambda environment. This requires [docker](#) to be installed. This is an optional approach should you choose to take advantage of it. Otherwise, the Quarkus JUnit integration should be sufficient for most of your needs.

A starter template has been generated for both JVM and native execution modes.

Run the following SAM CLI command to locally test your function, passing the appropriate SAM **template**. The **event** parameter takes any JSON file, in this case the sample `payload.json`.

```
sam local invoke --template target/sam.jvm.yaml --event
payload.json
```

The native image can also be locally tested using the `sam.native.yaml` template:

```
sam local invoke --template target/sam.native.yaml --event
payload.json
```

Modifying `function.zip`

There are times where you may have to add additional entries to the `function.zip` lambda deployment that is generated by the build. To do this create a `zip.jvm` or `zip.native` directory within `src/main`. Create `zip.jvm/` if you are doing a pure Java. `zip.native/` if you are doing a native deployment.

Any files and directories you create under your zip directory will be included within `function.zip`

Custom `bootstrap` script

There are times you may want to set specific system properties or other arguments when lambda invokes your native Funqy deployment. If you include a `bootstrap` script file within `zip.native`, the Funqy extension will automatically rename the executable to `runner` within `function.zip` and set the unix mode of the `bootstrap` script to executable.



The native executable must be referenced as `runner` if you include a custom `bootstrap` script.

The extension generates an example script within `target/bootstrap-example.sh`.