Quarkus - Google Cloud Functions (Serverless)

The `quarkus-google-cloud-functions` extension allows you to use Quarkus to build your Google Cloud Functions. Your functions can use injection annotations from CDI or Spring and other Quarkus facilities as you need them.

As the Google Cloud Function Java engine is a new Beta feature of Google Cloud, this extension is flagged as experimental.

> This technology is considered experimental.
>
> In *experimental* mode, early feedback is requested to mature the idea. There is no guarantee of stability nor long term presence in the platform until the solution matures. 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 15 minutes

- JDK 11 (Google Cloud Functions requires JDK 11)

- Apache Maven 3.6.3

- A Google Cloud Account. Free accounts work.

- Cloud SDK CLI Installed

Solution

This guide walks you through generating a sample project followed by creating multiple functions showing how to implement `HttpFunction`, `BackgroundFunction` and `RawBackgroundFunction` in Quarkus. Once built, you will be able to deploy the project to Google Cloud.

If you don't want to follow all these steps, you can go right to the completed example.

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

The solution is located in the `google-cloud-functions-quickstart` directory.

Creating the Maven Deployment Project

Create an application with the `quarkus-google-cloud-functions` extension. You can use the following Maven command to create it:

```
mvn io.quarkus:quarkus-maven-plugin:1.8.0.CR1:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=google-cloud-functions \
    -DclassName="org.acme.quickstart.GreetingResource" \
    -Dpath="/hello" \
    -Dextensions="google-cloud-functions"
```

Now, let's remove what's not needed inside the generated application:

- Remove the dependency `io.quarkus:quarkus-reasteasy` from your `pom.xml` file.

- Remove the generated `org.acme.quickstart.GreetingResource` class.

- Remove the `index.html` from `resources/META-INF/resources` or it will be picked up instead of your Function.

- Remove the existing tests.

Login to Google Cloud

Login to Google Cloud is necessary for deploying the application and it can be done as follows:

```
gcloud auth login
```

At the time of this writing, Cloud Functions are still in beta so make sure to install the `beta` command group.

```
gcloud components install beta
```

Creating the functions

For this example project, we will create three functions, one `HttpFunction`, one `BackgroundFunction` (Storage event) and one `RawBakgroundFunction` (PubSub event).

Choose Your Function

The `quarkus-google-cloud-functions` extension scans your project for a class that directly implements the Google Cloud `HttpFunction`, `BackgroundFunction` or `RawBakgroundFunction` interface. It must find a class in your project that implements one of these

interfaces or it will throw a build time failure. If it finds more than one function classes, a build time exception will also be thrown.

Sometimes, though, you might have a few related functions that share code and creating multiple maven modules is just an overhead you don't want to do. The extension allows you to bundle multiple functions in one project and use configuration or an environment variable to pick the function you want to deploy.

To configure the name of the function, you can use the following configuration property:

```
quarkus.google-cloud-functions.function=test
```

The `quarkus.google-cloud-functions.function` property tells Quarkus which function to deploy. This can be overridden with an environment variable too.

The CDI name of the function class must match the value specified within the `quarkus.google-cloud-functions.function` property. This must be done using the `@Named` annotation.

```
@Named("test")
public class TestHttpFunction implements HttpFunction {
}
```

The HttpFunction

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.io.Writer;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.inject.Named;
import com.google.cloud.functions.HttpFunction;
import com.google.cloud.functions.HttpRequest;
import com.google.cloud.functions.HttpResponse;
import io.quarkus.gcp.function.test.service.GreetingService;

@Named("httpFunction") ①
@ApplicationScoped ②
public class HttpFunctionTest implements HttpFunction { ③
    @Inject GreetingService greetingService; ④

    @Override
    public void service(HttpRequest httpRequest, HttpResponse
httpResponse) throws Exception { ⑤
        Writer writer = httpResponse.getWriter();
        writer.write(greetingService.hello());
    }
}
```

1. The @Named annotation allows to name the CDI bean to be used by the `quarkus.google-cloud-functions.function` property, this is optional.

2. The function must be a CDI bean

3. This is a regular Google Cloud Function implementation, so it needs to implement `com.google.cloud.functions.HttpFunction`.

4. Injection works inside your function.

5. This is standard Google Cloud Function implementation, nothing fancy here.

The BackgroundFunction

This `BackgroundFunction` is triggered by a Storage event, you can use any events supported by Google Cloud instead.

```
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.inject.Named;
import com.google.cloud.functions.BackgroundFunction;
import com.google.cloud.functions.Context;
import io.quarkus.gcp.function.test.service.GreetingService;


@Named("storageTest") ①
@ApplicationScoped ②
public class BackgroundFunctionStorageTest implements
BackgroundFunction<BackgroundFunctionStorageTest.StorageEvent> { ③
    @Inject GreetingService greetingService;    ④

    @Override
    public void accept(StorageEvent event, Context context) throws
Exception { ⑤
        System.out.println("Receive event: " + event);
        System.out.println("Be polite, say " +
greetingService.hello());
    }

    //
    public static class StorageEvent { ⑥
        public String name;
    }
}
```

1. The `@Named` annotation allows to name the CDI bean to be used by the `quarkus.google-cloud-functions.function` property, this is optional.

2. The function must be a CDI bean

3. This is a regular Google Cloud Function implementation, so it needs to implement `com.google.cloud.functions.BackgroundFunction`.

4. Injection works inside your function.

5. This is standard Google Cloud Function implementation, nothing fancy here.

6. This is the class the event will be deserialized to.

The RawBackgroundFunction

This `RawBackgroundFunction` is triggered by a PubSub event, you can use any events supported by Google Cloud instead.

```
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.inject.Named;
import com.google.cloud.functions.Context;
import com.google.cloud.functions.RawBackgroundFunction;
import io.quarkus.gcp.function.test.service.GreetingService;

@Named("rawPubSubTest") ①
@ApplicationScoped ②
public class RawBackgroundFunctionPubSubTest implements
RawBackgroundFunction { ③
@Inject GreetingService greetingService; ④

    @Override
    public void accept(String event, Context context) throws
Exception { ⑤
        System.out.println("PubSub event: " + event);
        System.out.println("Be polite, say " +
greetingService.hello());
    }
}
```

1. The @Named annotation allows to name the CDI bean to be used by the `quarkus.google-cloud-functions.function` property, this is optional.

2. The function must be a CDI bean

3. This is a regular Google Cloud Function implementation, so it needs to implement `com.google.cloud.functions.RawBackgroundFunction`.

4. Injection works inside your function.

5. This is standard Google Cloud Function implementation, nothing fancy here.

Build and Deploy to Google Cloud

To build your application, you can package it using the standard `mvn clean package` command.

The result of the previous command is a single JAR file inside the `target/deployment` repository that contains classes and dependencies of the project.

Then you will be able to use `gcloud beta functions deploy` command to deploy your function to Google Cloud.

The first time you launch this command, you can have the following error message:

⚠️

```
ERROR: (gcloud.beta.functions.deploy) OperationError:
code=7, message=Build Failed: Cloud Build has not been
used in project <project_name> before or it is
disabled. Enable it by visiting
https://console.developers.google.com/apis/api/cloudbui
ld.googleapis.com/overview?project=<my-project> then
retry.
```

This means that Cloud Build is not activated yet. To overcome this error, open the URL shown in the error, follow the instructions and then wait a few minutes before retrying the command.

The HttpFunction

This is an example command to deploy your `HttpFunction` to Google Cloud:

```
gcloud beta functions deploy quarkus-example-http \
  --entry-point=io.quarkus.gcp.functions.QuarkusHttpFunction \
  --runtime=java11 --trigger-http --source=target/deployment
```

🛑

The entry point must always be set to `io.quarkus.gcp.functions.QuarkusHttpFunction` as this is the class that integrates Cloud Functions with Quarkus.

This command will give you as output a `httpsTrigger.url` that points to your function.

The BackgroundFunction

Before deploying your function, you need to create a bucket.

```
gsutil mb gs://quarkus-hello
```

This is an example command to deploy your `BackgroundFunction` to Google Cloud, as the function is triggered by a Storage event, it needs to use `--trigger-event google.storage.object.finalize` and the `--trigger-resource` parameter with the name of a previously created bucket:

```
gcloud beta functions deploy quarkus-example-storage \
    --entry
-point=io.quarkus.gcp.functions.QuarkusBackgroundFunction \
    --trigger-resource quarkus-hello --trigger-event
google.storage.object.finalize \
    --runtime=java11 --source=target/deployment
```

⛔ The entry point must always be set to `io.quarkus.gcp.functions.QuarkusBackgroundFunction` as this is the class that integrates Cloud Functions with Quarkus.

To trigger the event, you can send a file to the GCS `quarkus-hello` bucket or you can use gcloud to simulate one:

```
gcloud functions call quarkus-example-storage  --data
'{"name":"test.txt"}'
```

ℹ️ `--data` contains the GCS event, it is a JSON document with the name of the file added to the bucket.

The RawBackgroundFunction

This is an example command to deploy your `RawBackgroundFunction` to Google Cloud, as the function is triggered by a PubSub event, it needs to use `--trigger-event google.pubsub.topic.publish` and the `--trigger-resource` parameter with the name of a previously created topic:

```
gcloud beta functions deploy quarkus-example-pubsub \
  --entry-point=io.quarkus.gcp.functions.QuarkusBackgroundFunction
\
  --runtime=java11 --trigger-resource hello_topic --trigger-event
google.pubsub.topic.publish --source=target/deployment
```

⛔ The entry point must always be set to `io.quarkus.gcp.functions.QuarkusBackgroundFunction` as this is the class that integrates Cloud Functions with Quarkus.

To trigger the event, you can send a file to the `hello_topic` topic of you can use gcloud to simulate one:

```
gcloud functions call quarkus-example-pubsub --data
'{"data":{"greeting":"world"}}'
```

Testing locally

The easiest way to locally test your function is using the Cloud Function invoker JAR.

You can download it via Maven using the following command:

```
mvn dependency:copy \
  -Dartifact='com.google.cloud.functions.invoker:java-function
-invoker:1.0.0-beta1' \
  -DoutputDirectory=.
```

Before using the invoker, you first need to build your function via `mvn package`.

The HttpFunction

To test an `HttpFunction`, you can use this command to launch your function locally.

```
java -jar java-function-invoker-1.0.0-beta1.jar \
  --classpath target/google-cloud-functions-1.0.0-SNAPSHOT-
runner.jar \
  --target io.quarkus.gcp.functions.QuarkusHttpFunction
```

> ❗ The `--classpath` parameter needs to be set to the previously packaged JAR that contains your function class and all Quarkus related classes.

Your endpoints will be available on http://localhost:8080.

The BackgroundFunction

For background functions, you launch the invoker with a target class of `io.quarkus.gcp.functions.BackgroundFunction`.

```
java -jar java-function-invoker-1.0.0-beta1.jar \
  --classpath target/google-cloud-functions-1.0.0-SNAPSHOT-
runner.jar \
  --target io.quarkus.gcp.functions.QuarkusBackgroundFunction
```

> ❗ The `--classpath` parameter needs to be set to the previously packaged JAR that contains your function class and all Quarkus related classes.

Then you can call your background function via an HTTP call with a payload containing the event:

```
curl localhost:8080 -d '{"data":{"name":"hello.txt"}}'
```

This will call your Storage background function with an event `{"name":"hello.txt"}`, so an event on the `hello.txt` file.

The RawBackgroundFunction

For background functions, you launch the invoker with a target class of `io.quarkus.gcp.functions.BackgroundFunction`.

```
java -jar java-function-invoker-1.0.0-beta1.jar \
  --classpath target/google-cloud-functions-1.0.0-SNAPSHOT-
runner.jar \
  --target io.quarkus.gcp.functions.QuarkusBackgroundFunction
```

> ❗ The `--classpath` parameter needs to be set to the previously packaged JAR that contains your function class and all Quarkus related classes.

Then you can call your background function via an HTTP call with a payload containing the event:

```
curl localhost:8080 -d '{"data":{"greeting":"world"}}'
```

This will call your PubSub background function with a PubSubMessage `{"greeting":"world"}`.