

Quarkus - Google Cloud Functions (Serverless) with RESTEasy, Undertow, or Vert.x Web

The `quarkus-google-cloud-functions-http` extension allows you to write microservices with RESTEasy (JAX-RS), Undertow (Servlet) or Vert.x Web, and make these microservices deployable to the Google Cloud Functions runtime.

One Google Cloud Functions deployment can represent any number of JAX-RS, Servlet, or Vert.x Web endpoints.

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 three HTTP endpoints written with JAX-RS APIs, Servlet APIs or Vert.x Web APIs. 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-http-quickstart](#) directory.

Creating the Maven Deployment Project

Create an application with the [quarkus-google-cloud-functions-http](#) extension. You can use the following Maven command to create it:

```
mvn io.quarkus:quarkus-maven-plugin:1.8.0.CR1:create \
  -DprojectId=org.acme \
  -DprojectId=google-cloud-functions-http \
  -DclassName="org.acme.quickstart.GreetingResource" \
  -Dpath="/hello" \
  -Dextensions="google-cloud-functions-http,resteasy
  -json,undertow,vertx-web"
```

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 endpoints

For this example project, we will create three endpoints, one for RESTEasy (JAX-RS), one for Undertow (Servlet) and one for Vert.x Web (reactive routes).

If you don't need endpoints of each type, you can remove the corresponding extensions from your [pom.xml](#).

The JAX-RS endpoint

```

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}

```

The Servlet endpoint

```

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "ServletGreeting", urlPatterns =
"/servlet/hello")
public class GreetingServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        resp.setStatus(200);
        resp.addHeader("Content-Type", "text/plain");
        resp.getWriter().write("hello");
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
        String name = req.getReader().readLine();
        resp.setStatus(200);
        resp.addHeader("Content-Type", "text/plain");
        resp.getWriter().write("hello " + name);
    }
}

```

The Vert.x Web endpoint

```
import static io.vertx.core.http.HttpMethod.GET;

import io.quarkus.vertx.web.Route;
import io.vertx.ext.web.RoutingContext;

public class GreetingRoutes {
    @Route(path = "/vertx/hello", methods = GET)
    void hello(RoutingContext context) {
        context.response().headers().set("Content-Type",
"text/plain");
        context.response().setStatusCode(200).end("hello");
    }
}
```

Build and Deploy to Google Cloud



Quarkus forces a packaging of type `uber-jar` for your function as Google Cloud Function deployment requires a single JAR.

Package your application using the standard `mvn clean package` command. The result of the previous command is a single JAR file inside the `target/deployment` directory that contains the classes and the dependencies of the project.

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

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



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

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.

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

You can then call your endpoints via:

- For JAX-RS: `{httpsTrigger.url}/hello`
- For servlet: `{httpsTrigger.url}/servlet/hello`
- For Vert.x Web: `{httpsTrigger.url}/vertx/hello`

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`.

Then you can use it to launch your function locally.

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

What's next?

You can use our [Google Cloud Functions Funqy binding](#) to use Funqy, a provider agnostic function as a service framework, that allow to deploy HTTP function or Background function to Google Cloud.