Tutorial QUARKUS
Reactive Systems in JAVA

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to present the details related with the QUARKUS microservices framework.

The examples address the following aspects: environment set-up, creation of projects and adding elements to it, accessing databases, and controlling the database transaction, using messages in JSON format, how to use KAFKA, how to provide an OpenAPI interface, and finally, deploying in docker and deploying in AWS.

The following contents is presented in this document.

Contents

A. Setup steps for initial setup of QUARKUS environment

A.1. Clone this **repository** https://github.com/quarkusio/quarkus-quickstarts.git using GitHub desktop or web.

A.2. Install Maven **3.9.9** from https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/

(for detail on installing in different operating systems:
https://mkyong.com/maven/install-maven-on-mac-osx/
https://mkyong.com/maven/how-to-install-maven-in-windows/)

A.3. Install **docker** from https://www.docker.com/get-started and launch it.

A.4. Install **Mandrel** (for linux only) or **GraalVM** (other platforms) with **JDK 17** from https://github.com/graalvm/mandrel or https://www.oracle.com/java/technologies/downloads/#graalvmjava17 respectively

(for details on installing in different operating systems
https://github.com/graalvm/homebrew-tap
https://www.graalvm.org/latest/getting-started/linux/
https://www.graalvm.org/latest/getting-started/windows/)

A.5. Check java version with "javac -version" in command line, and change JDK if needed following these instructions:
https://mkyong.com/java/how-to-set-java_home-environment-variable-on-mac-os-x/
https://mkyong.com/java/how-to-set-java_home-on-windows-10/

A.6. In the repository base directory of step A.1. execute the following to assure that your environment is ok:

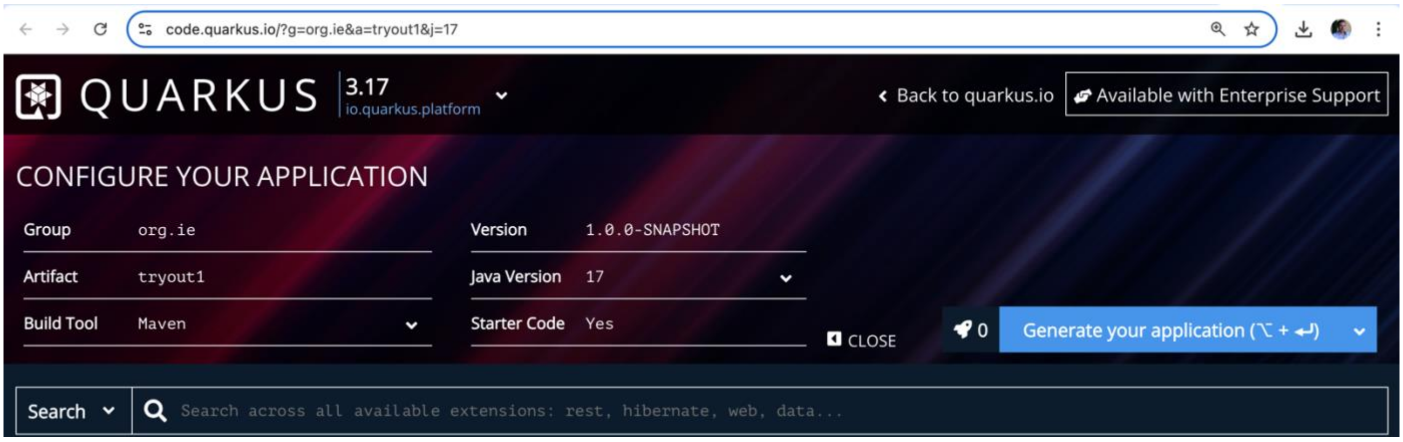
```
mvn clean install
```

A.7. (Optional) Install **mysql server** locally (optionally, for testing purposes of QUARKUS accessing data reactively) or, later, use AWS RDS.

A.8. (Optional) Install VS Code (https://code.visualstudio.com/) or other preferred IDE.

B. Create a Simple QUARKUS project with starter code

B.1.Navigate to the QUARKUS project creation website at https://code.quarkus.io and use the following configuration:


code.quarkus.io/?g=org.ie&a=tryout1&j=17

CONFIGURE YOUR APPLICATION

Group org.ie
Artifact tryout1
Build Tool Maven
Version 1.0.0-SNAPSHOT
Java Version 17
Starter Code Yes

QUARKUS 3.17 io.quarkus.platform
Back to quarkus.io
Available with Enterprise Support
CLOSE
0
Generate your application (⌥ + ↵)
Search
Search across all available extensions: rest, hibernate, web, data...

B.2. Generate, download the zip file and extract it to your local folder project

B.3. Import your newly created project to VS Code

B.4. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

B.5. Go to URL http://localhost:8080/hello and check the message that is produced.

B.6. Open the source code on your IDE, *e.g.*, VS Code and navigate to **GreetingResource.java** file. Study the source code and change the endpoint at line:

```
@Path("/hello")
```

To

```
@Path("/newapi")
```

B.7. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

B.8. Go to URL http://localhost:8080/newapi and http://localhost:8080/welcome and check the result that is produced.

C. Add a new JAVA class to and already existing QUARKUS project

C.1. Create a new class in your project with the name: **SecondClass.java**, in the same location as previous GreetingResource.java.

C.2. Update the new file SecondClass.java accordingly with this source code:

```
package org.ie;

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

@Path("/api2")
public class SecondClass {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "Hello to the second API";
    }
}
```

Hint: you may also add a test class to automate the code verification

C.3. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

C.4. Go to URL http://localhost:8080/newapi and http://localhost:8080/api2 and check the result that is produced.

C.5. If needed, you can change the port, adding the following configuration in **application.properties** of your project (resources directory of your Quarkus project) :

```
quarkus.http.port=9000
```

D. QUARKUS exposed by OpenAPI interface

The aim of this example is to provide QUARKUS APIs with OpenAPI (https://quarkus.io/guides/openapi-swaggerui).

D.1. Execute the following command to add the openapi extension:

```
                    ./mvnw quarkus:add-extension -Dextensions='quarkus-smallrye-openapi'
```

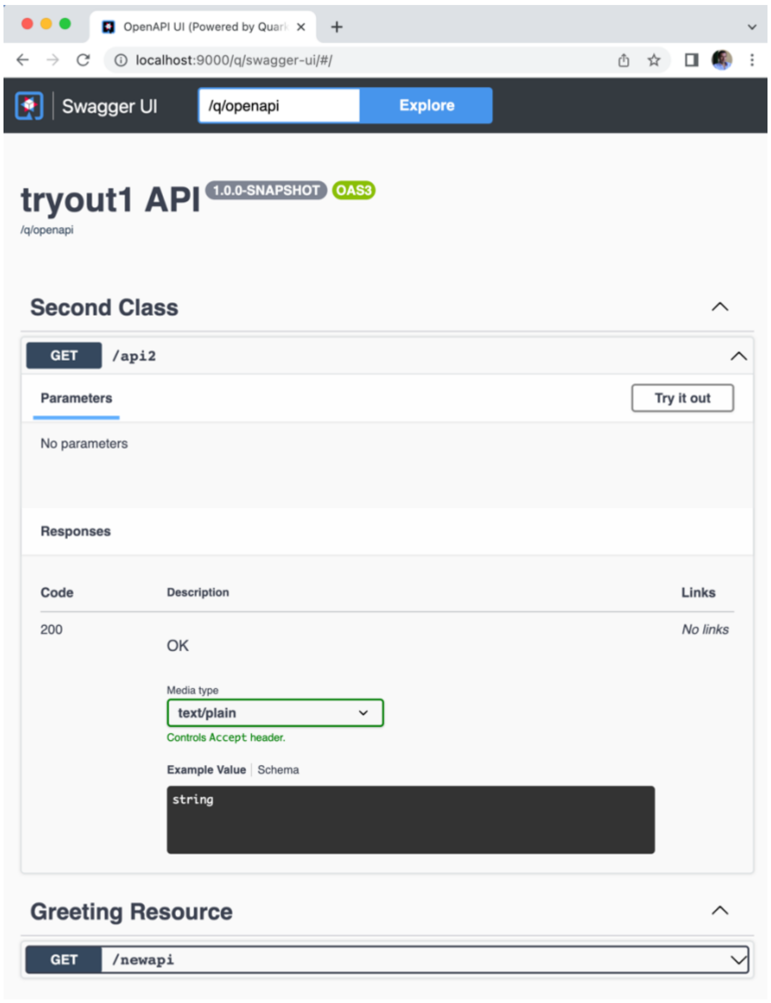
D.2. Change the end point for swagger-ui, adding the following configuration in application.properties:

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true
```

D.3. Execute the command on command line or calling maven inside VS Code:

```
./mvnw quarkus:dev
```

D.4. Navigate to http://localhost:9000/q/swagger-ui/ , a similar result will be presented to you:



tryout1 API 1.0.0-SNAPSHOT OAS3
/q/openapi

Second Class

GET /api2

Parameters
Try it out

No parameters

Responses

Code	Description	Links
200	OK Media type: text/plain Controls Accept header. Example Value / Schema string	No links

Greeting Resource

GET /newapi

E. QUARKUS accessing database with all CRUD Operations

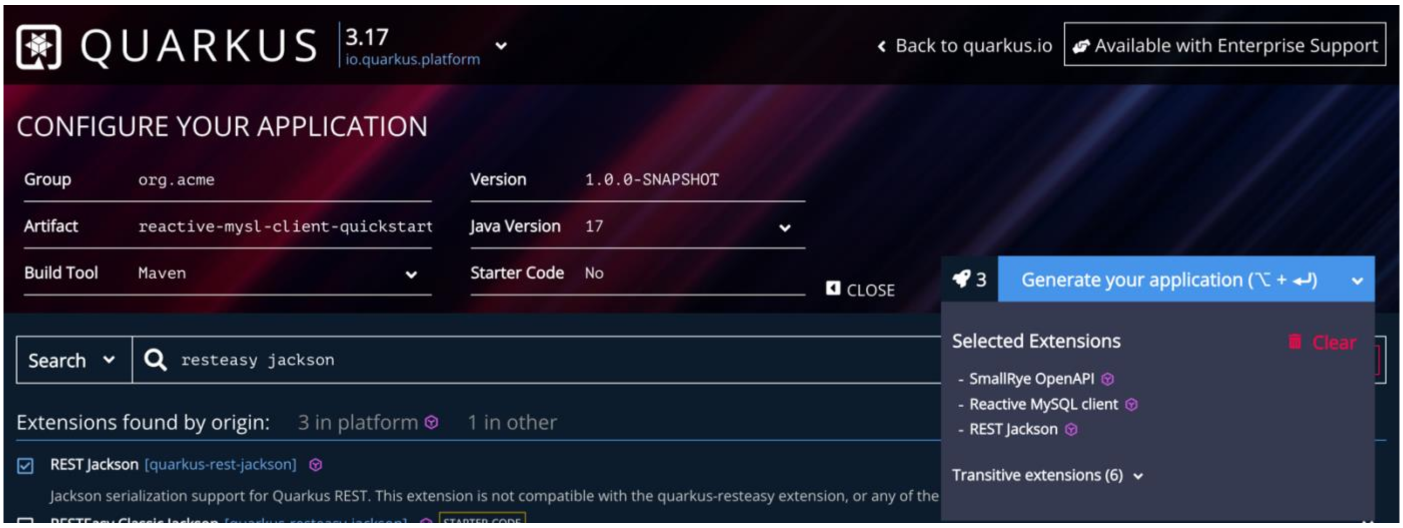
E.1. A ready-to-use MySQL server to try out this example.

```
    docker run -it --name mysqlInstance -e MYSQL_ROOT_PASSWORD=password -p 3306:3306 -d mysql:latest
```

E.2. Create a MySQL table to try out the examples with the following command in MYQSL Workbench;

```
CREATE DATABASE quarkus_test;
```

E.3. Navigate to the QUARKUS project creation website at https://code.quarkus.io and use the following configuration:



QUARKUS 3.17 io.quarkus.platform — Back to quarkus.io — Available with Enterprise Support

CONFIGURE YOUR APPLICATION

Group	org.acme	Version	1.0.0-SNAPSHOT
Artifact	reactive-mysl-client-quickstart	Java Version	17
Build Tool	Maven	Starter Code	No

CLOSE — 3 — Generate your application (⌥ + ⏎)

Search: resteasy jackson

Extensions found by origin: 3 in platform, 1 in other

REST Jackson [quarkus-rest-jackson]
Jackson serialization support for Quarkus REST. This extension is not compatible with the quarkus-resteasy extension, or any of the

Selected Extensions — Clear
- SmallRye OpenAPI
- Reactive MySQL client
- REST Jackson

Transitive extensions (6)

E.4. Generate, download the zip file and extract it to your local folder project

E.5. Import your newly created project to VS Code

E.6. Add the following configuration in src/main/resources/application.properties of your project

```
quarkus.datasource.db-kind=mysql
quarkus.datasource.username=root
quarkus.datasource.password=<your MySQL root password>
quarkus.datasource.reactive.url=mysql://localhost:3306/quarkus_test

quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true
```

E.7. Add the file Fruit.java to src/main/java/Fruit.java

```
import io.smallrye.mutiny.Multi;
import io.smallrye.mutiny.Uni;
import io.vertx.mutiny.mysqlclient.MySQLPool;
import io.vertx.mutiny.sqlclient.Row;
import io.vertx.mutiny.sqlclient.RowSet;
import io.vertx.mutiny.sqlclient.Tuple;

public class Fruit {

    public Long id;
    public String name;

         public Fruit() {
         }

         public Fruit(String name) {
             this.name = name;
         }

         public Fruit(Long id, String name) {
             this.id = id;
             this.name = name;
         }
```

```
        private static Fruit from(Row row) {
            return new Fruit(row.getLong("id"), row.getString("name"));
        }

        public static Multi<Fruit> findAll(MySQLPool client) {
            return client.query("SELECT id, name FROM fruits ORDER BY name ASC").execute()
                    .onItem().transformToMulti(set -> Multi.createFrom().iterable(set))
                    .onItem().transform(Fruit::from);
        }

        public static Uni<Fruit> findById(MySQLPool client, Long id) {
            return client.preparedQuery("SELECT id, name FROM fruits WHERE id = ?").execute(Tuple.of(id))
                    .onItem().transform(RowSet::iterator)
                    .onItem().transform(iterator -> iterator.hasNext() ? from(iterator.next()) : null);
        }

        public Uni<Boolean> save(MySQLPool client) {
            return client.preparedQuery("INSERT INTO fruits(name) VALUES (?)").execute(Tuple.of(name))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1 );
        }

        public static Uni<Boolean> delete(MySQLPool client, Long id) {
            return client.preparedQuery("DELETE FROM fruits WHERE id = ?").execute(Tuple.of(id))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1);
        }

        public static Uni<Boolean> update(MySQLPool client, Long id, String name) {
            return client.preparedQuery("UPDATE fruits SET name = ? WHERE id =
?").execute(Tuple.of(name,id))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1 );
        }
 }
```

E.8. Add the file FruitResource.java to src/main/java/FruitResource.java

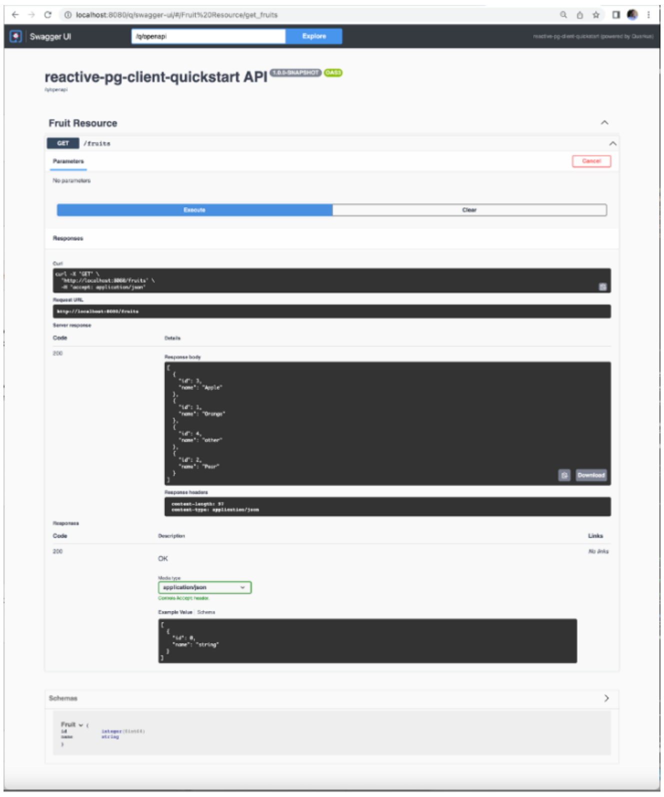
```
import java.net.URI;
import jakarta.enterprise.event.Observes;
import jakarta.inject.Inject;
import jakarta.ws.rs.*;
import org.eclipse.microprofile.config.inject.ConfigProperty;

import io.quarkus.runtime.StartupEvent;
import io.smallrye.mutiny.Multi;
import io.smallrye.mutiny.Uni;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.core.Response.ResponseBuilder;
import jakarta.ws.rs.core.MediaType;


@Path("fruits")
public class FruitResource {

    @Inject
    io.vertx.mutiny.mysqlclient.MySQLPool client;

    @Inject
    @ConfigProperty(name = "myapp.schema.create", defaultValue = "true")
    boolean schemaCreate;

    void config(@Observes StartupEvent ev) {
        if (schemaCreate) {
            initdb();
        }
    }

    private void initdb() {
       client.query("DROP TABLE IF EXISTS fruits").execute()
        .flatMap(r -> client.query("CREATE TABLE fruits (id SERIAL PRIMARY KEY, name TEXT NOT
NULL)").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Orange')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Pear')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Apple')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('other')").execute())
        .await().indefinitely();
    }

    @GET
    public Multi<Fruit> get() {
        return Fruit.findAll(client);
    }

    @GET
    @Path("{id}")
    public Uni<Response> getSingle(Long id) {
        return Fruit.findById(client, id)
                .onItem().transform(fruit -> fruit != null ? Response.ok(fruit) :
Response.status(Response.Status.NOT_FOUND))
                .onItem().transform(ResponseBuilder::build);
    }

    @POST
    public Uni<Response> create(Fruit fruit) {
        return fruit.save(client)
                .onItem().transform(id -> URI.create("/fruits/" + id))
                .onItem().transform(uri -> Response.created(uri).build());
    }

    @DELETE
    @Path("{id}")
    public Uni<Response> delete(Long id) {
        return Fruit.delete(client, id)
                .onItem().transform(deleted -> deleted ? Response.Status.NO_CONTENT :
Response.Status.NOT_FOUND)
                .onItem().transform(status -> Response.status(status).build());
    }

    @PUT
    @Path("/{id}/{name}")
    public Uni<Response> update(Long id , String name) {
        return Fruit.update(client, id , name)
                .onItem().transform(updated -> updated ? Response.Status.NO_CONTENT :
Response.Status.NOT_FOUND)
                .onItem().transform(status -> Response.status(status).build());
    }
}
```

E.9. Compile and execute the project:

```
cd reactive-mysql-CRUD-client-quickstart

./mvnw quarkus:dev
```

E.10. Navigate to http://localhost:8080/q/swagger-ui/ and execute the all the CRUD APIs.

Try to create, update, and delete an item.



```
curl -X 'GET' \
  'http://localhost:8080/fruits' \
  -H 'accept: application/json'
```

```
http://localhost:8080/fruits
```

200

```
[
  {
    "id": 3,
    "name": "Apple"
  },
  {
    "id": 1,
    "name": "Orange"
  },
  {
    "id": 4,
    "name": "other"
  },
  {
    "id": 2,
    "name": "Pear"
  }
]
```

```
content-length: 97
content-type: application/json
```

200 OK

application/json

```
[
  {
    "id": 0,
    "name": "string"
  }
]
```

```
Fruit ∨ {
id        integer($int64)
name      string
}
```

F. Deploying a QUARKS project with docker locally

F.1. Execute the following command in your project directory:

```
./mvnw quarkus:add-extension -Dextensions="quarkus-container-image-docker"
```

F.2. Add the following configuration to application.properties in order to include all the dependencies to execute in a different environment from dev:

```
quarkus.container-image.build=true
```

F.3. Build docker image

*Hint 1: if you are not logged in on docker use the "**docker login**" command*
Hint 2: have in mind that all the network requirements for the docker image need to be met beforehand. To facilitate start, e.g., with a simple tryout project as in section B of this document.

```
./mvnw clean package
```

F.4. Check that image has been created successfully.

```
% docker image ls -a
REPOSITORY                TAG               IMAGE ID       CREATED         SIZE
sergioguerreiro/tryout1   1.0.0-SNAPSHOT    eaf911ae3716   15 hours ago    492MB
```

F.5. Create container. Have in mind that the port available and the name of the project depends on your previous settings.

```
docker run -d --name tryout1 -p 9000:9000 sergioguerreiro/tryout1:1.0.0-SNAPSHOT
```

Test the container opening the following URL in your browser:

```
# open browser with url: http://localhost:9000/q/swagger-ui/
```

Hint 1: Later if you would like to list all the available container use the following command:

```
> docker container ls --all

CONTAINER ID   IMAGE                                     COMMAND                   CREATED         STATUS
PORTS                                                    NAMES
2d4cc67021e3   sergioguerreiro/tryout1:1.0.0-SNAPSHOT    "/usr/local/s2i/run"      2 minutes ago   Up 2 minutes
8080/tcp, 8443/tcp, 8778/tcp, 0.0.0.0:9000->9000/tcp     tryout1
```

Hint 2: Later if you would like to stop the container use the following command:

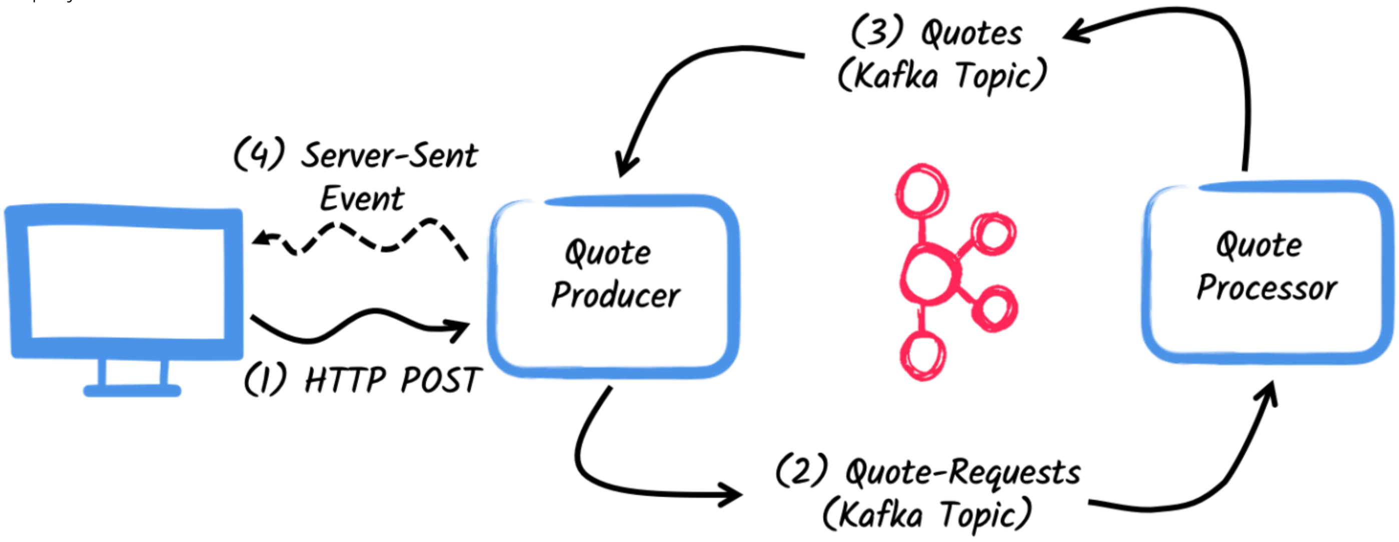
```
docker container stop <containerID_hash>
```

Hint 3: Later if you would like to restart the container use the following command:

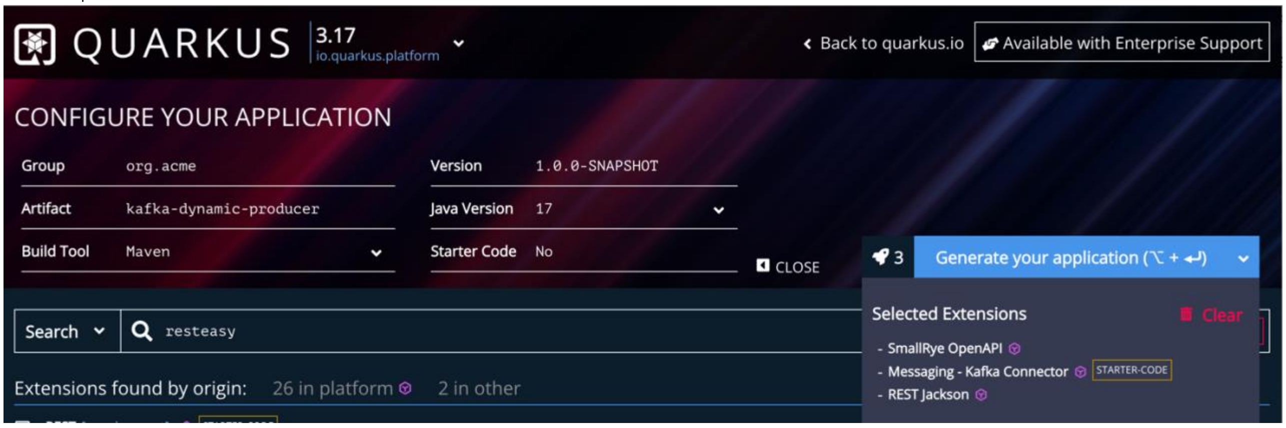
```
docker container start <containerID_hash>
```

G. QUARKUS with Kafka messaging embedded (in development environment)

The following example aims at implementing the following messaging architecture. Kafka is embedded in QUARKUS deployment.



G.1. Create an empty project (KAFKA message producer) at https://code.quarkus.io/ with the following dependencies.



QUARKUS 3.17 io.quarkus.platform
Back to quarkus.io
Available with Enterprise Support
CLOSE
Generate your application (⌥ + ↵)
Clear
STARTER-CODE

CONFIGURE YOUR APPLICATION

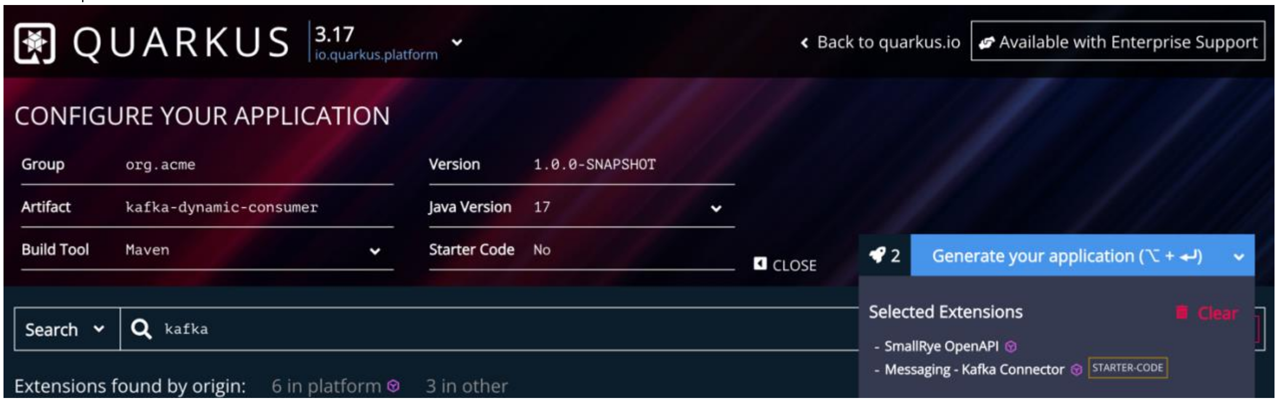
Group org.acme
Version 1.0.0-SNAPSHOT
Artifact kafka-dynamic-producer
Java Version 17
Build Tool Maven
Starter Code No

Search resteasy

Extensions found by origin: 26 in platform 2 in other

Selected Extensions
- SmallRye OpenAPI
- Messaging - Kafka Connector
- REST Jackson

G.2.Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.



QUARKUS 3.17 io.quarkus.platform
Back to quarkus.io
Available with Enterprise Support
CLOSE
Generate your application (⌥ + ↵)
Clear
STARTER-CODE

CONFIGURE YOUR APPLICATION

Group org.acme
Version 1.0.0-SNAPSHOT
Artifact kafka-dynamic-consumer
Java Version 17
Build Tool Maven
Starter Code No

Search kafka

Extensions found by origin: 6 in platform 3 in other

Selected Extensions
- SmallRye OpenAPI
- Messaging - Kafka Connector

G.3.In project kafka-dynamic-producer create the following JAVA classes at src/main/java/org/acme/kafka/model/Quote.java

```
package org.acme.kafka.model;

public class Quote {

    public String id;
    public int price;
    /**
    * Default constructor required for Jackson serializer
    */
    public Quote() { }

    public Quote(String id, int price) {
        this.id = id;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Quote{" +
                "id='" + id + '\'' +
                ", price=" + price +
                '}';
    }
}
```

Add to kafka-dynamic-producer the **src/main/java/org/acme/kafka/producer/QuotesResource.java**

```
package org.acme.kafka.producer;

import java.util.UUID;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.acme.kafka.model.Quote;
import org.eclipse.microprofile.reactive.messaging.Channel;
import org.eclipse.microprofile.reactive.messaging.Emitter;

import io.smallrye.mutiny.Multi;

@Path("/quotes")
public class QuotesResource {

    @Channel("quote-requests")
    Emitter<String> quoteRequestEmitter;

    @Channel("quotes")
    Multi<Quote> quotes;
    /**
     * Endpoint to generate a new quote request id and send it to "quote-requests" Kafka topic using the
emitter.
     */
    @POST
    @Path("/request")
    @Produces(MediaType.TEXT_PLAIN)
    public String createRequest() {
        UUID uuid = UUID.randomUUID();
        quoteRequestEmitter.send(uuid.toString());
        return uuid.toString();
    }

    /**
     * Endpoint retrieving the "quotes" Kafka topic and sending the items to a server sent event.
     */
    @GET
    @Produces(MediaType.SERVER_SENT_EVENTS)
    public Multi<Quote> stream() {
        return quotes;
    }

}
```

Add to kafka-dynamic-producer the following **index.html** file to **/src/main/resources/META-INF/resources/**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>kafka-quickstart - 1.0.0-SNAPSHOT</title>
    <style>
        h1, h2, h3, h4, h5, h6 {
            margin-bottom: 0.5rem;
            font-weight: 400;
            line-height: 1.5;
        }

        h1 {
            font-size: 2.5rem;
        }

        h2 {
            font-size: 2rem
        }

        h3 {
            font-size: 1.75rem
        }

        h4 {
            font-size: 1.5rem
        }

        h5 {
            font-size: 1.25rem
        }

        h6 {
            font-size: 1rem
        }

        .lead {
            font-weight: 300;
            font-size: 2rem;
        }

        .banner {
            font-size: 2.7rem;
            margin: 0;
            padding: 2rem 1rem;
            background-color: #00A1E2;
            color: white;
        }

        body {
            margin: 0;
            font-family: -apple-system, system-ui, "Segoe UI", Roboto, "Helvetica Neue", Arial, sans-serif,
"Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
        }

        code {
            font-family: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New",
monospace;
            font-size: 87.5%;
            color: #e83e8c;
            word-break: break-word;
        }

        .left-column {
            padding: .75rem;
            max-width: 75%;
            min-width: 55%;
        }

        .right-column {
            padding: .75rem;
            max-width: 25%;
        }

        .container {
            display: flex;
            width: 100%;
        }

        li {
            margin: 0.75rem;
        }

        .right-section {
            margin-left: 1rem;
            padding-left: 0.5rem;
        }

        .right-section h3 {
            padding-top: 0;
            font-weight: 200;
        }

        .right-section ul {
            border-left: 0.3rem solid #00A1E2;
            list-style-type: none;
            padding-left: 0;
        }

    </style>
</head>
<body>

<div class="banner lead">
    Your new Cloud-Native application is ready!
</div>

<div class="container">
    <div class="left-column">
        <p class="lead"> Congratulations, you have created a new Quarkus application.</p>

        <h2>Why do you see this?</h2>

        <p>This page is served by Quarkus. The source is in
            <code>src/main/resources/META-INF/resources/index.html</code>.</p>

        <h2>What can I do from here?</h2>

        <p>If not already done, run the application in <em>dev mode</em> using: <code>mvn quarkus:dev</code>.
        </p>
        <ul>
          <li>Add REST resources, Servlets, functions and other services in <code>src/main/java</code>.</li>
          <li>Your static assets are located in <code>src/main/resources/META-INF/resources</code>.</li>
          <li>Configure your application in <code>src/main/resources/application.properties</code>.
          </li>
        </ul>

        <h2>How do I get rid of this page?</h2>
        <p>Just delete the <code>src/main/resources/META-INF/resources/index.html</code> file.</p>
    </div>
    <div class="right-column">
        <div class="right-section">
            <h3>Application</h3>
            <ul>
                <li>GroupId: org.acme.quarkus.sample</li>
                <li>ArtifactId: kafka-quickstart</li>
                <li>Version: 1.0.0-SNAPSHOT</li>
                <li>Quarkus Version: 999-SNAPSHOT</li>
            </ul>
        </div>
        <div class="right-section">
            <h3>Next steps</h3>
            <ul>
                <li><a href="https://quarkus.io/guides/maven-tooling.html">Setup your IDE</a></li>
                <li><a href="https://quarkus.io/guides/getting-started.html">Getting started</a></li>
                <li><a href="https://quarkus.io">Quarkus Web Site</a></li>
            </ul>
        </div>
    </div>
</div>


</body>
</html>
```

Add to kafka-dynamic-producer the following quotes.html file to **/src/main/resources/META-INF/resources/**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prices</title>

    <link rel="stylesheet" type="text/css"
          href="https://cdnjs.cloudflare.com/ajax/libs/patternfly/3.24.0/css/patternfly.min.css">
    <link rel="stylesheet" type="text/css"
          href="https://cdnjs.cloudflare.com/ajax/libs/patternfly/3.24.0/css/patternfly-additions.min.css">
</head>
<body>
<div class="container">
    <div class="card">
```

```
        <div class="card-body">
            <h2 class="card-title">Quotes</h2>
            <button class="btn btn-info" id="request-quote">Request Quote</button>
            <div class="quotes"></div>
        </div>
    </div>
</div>
</body>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $("#request-quote").click((event) => {
        fetch("/quotes/request", {method: "POST"})
        .then(res => res.text())
        .then(qid => {
            var row = $(`<h4 class='col-md-12' id='${qid}'>Quote # <i>${qid}</i> |
<strong>Pending</strong></h4>`);
            $(".quotes").prepend(row);
        });
    });

    var source = new EventSource("/quotes");
    source.onmessage = (event) => {
      var json = JSON.parse(event.data);
      $(`#${json.id}`).html((index, html) => {
        return html.replace("Pending", `\$\xA0${json.price}`);
      });
    };
</script>
</html>
```

G.4. Add the following configuration in src/main/resources/application.properties of your consumer project:

```
%dev.quarkus.http.port=8081

# Go bad to the first records, if it's out first access
mp.messaging.incoming.requests.auto.offset.reset=earliest

# Set the Kafka topic, as it's not the channel name
mp.messaging.incoming.requests.topic=quote-requests
```

Hint: For now, the producer project do not need any configuration in the application.properties.

G.5. In project kafka-dynamic-consumer create the following JAVA classes at
src/main/java/org/acme/kafka/model/Quote.java

```
package org.acme.kafka.model;

public class Quote {

    public String id;
    public int price;

    /**
    * Default constructor required for Jackson serializer
    */
    public Quote() { }

    public Quote(String id, int price) {
        this.id = id;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Quote{" +
                "id='" + id + '\'' +
                ", price=" + price +
                '}';
    }
}
```

Add to kafka-dynamic-consumer the **src/main/java/org/acme/kafka/processor/QuotesProcessor.java**

```
package org.acme.kafka.processor;
```

```
import java.util.Random;

import jakarta.enterprise.context.ApplicationScoped;

import org.acme.kafka.model.Quote;
import org.eclipse.microprofile.reactive.messaging.Incoming;
import org.eclipse.microprofile.reactive.messaging.Outgoing;

import io.smallrye.reactive.messaging.annotations.Blocking;

/**
 * A bean consuming data from the "quote-requests" Kafka topic (mapped to "requests" channel) and giving out
a random quote.
 * The result is pushed to the "quotes" Kafka topic.
 */
@ApplicationScoped
public class QuotesProcessor {

    private Random random = new Random();

    @Incoming("requests")
    @Outgoing("quotes")
    @Blocking
    public Quote process(String quoteRequest) throws InterruptedException {
        // simulate some hard working task
        Thread.sleep(200);
        return new Quote(quoteRequest, random.nextInt(100));
    }
}
```

G.6. Execute in a separate command line:

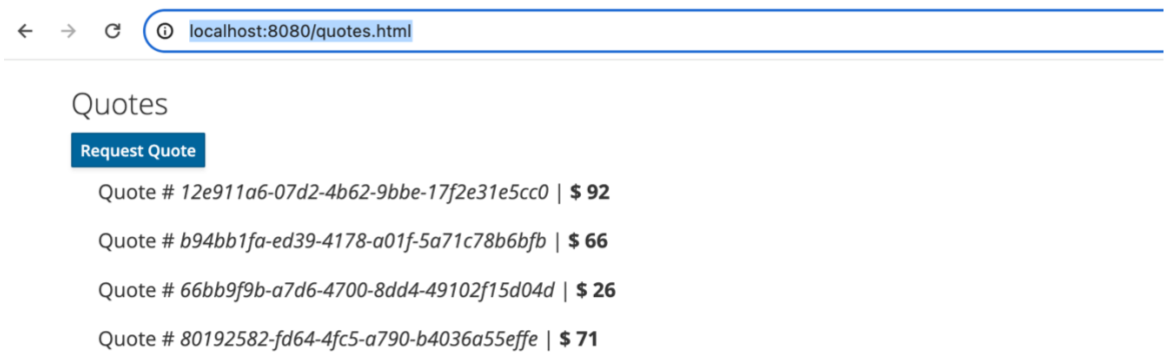
```
cd kafka-quickstart-consumer

./mvnw quarkus:dev
```

G.7. Execute in a separate command line:

```
cd kafka-quickstart-producer

./mvnw quarkus:dev
```

G.8. Navigate to http://localhost:8080/quotes.html

Then, click “Request Quote” and check in the consumer window that the message has been printed.



Quotes

Quote # *12e911a6-07d2-4b62-9bbe-17f2e31e5cc0* | **$ 92**

Quote # *b94bb1fa-ed39-4178-a01f-5a71c78b6bfb* | **$ 66**

Quote # *66bb9f9b-a7d6-4700-8dd4-49102f15d04d* | **$ 26**

Quote # *80192582-fd64-4fc5-a790-b4036a55effe* | **$ 71**

H. QUARKUS with Kafka messaging not embedded

Using the previous project of "**G. QUARKUS with Kafka messaging embedded (in development environment)**" execute the following configuration changes:

H.1. Add the following configuration in src/main/resources/application.properties of your consumer project. You are adding a remote integration with a previously installed Kafka server (or broker) in AWS EC2 at port 9092.

```
kafka.bootstrap.servers=<YOUR EC2 NAME>:9092

mp.messaging.incoming.requests.topic=quote-requests
mp.messaging.incoming.requests.auto.offset.reset=earliest
```

H.2. Then, add the following configuration in src/main/resources/application.properties of your producer project. You are adding a remote integration with a previously installed Kafka server (or broker) in AWS EC2 at port 9092.

```
kafka.bootstrap.servers=<YOUR EC2 NAME>:9092
```

H.3. Execute in a separate command line

```
cd kafka-quickstart-consumer

./mvnw quarkus:dev
```

H.4.Execute in a separate command line

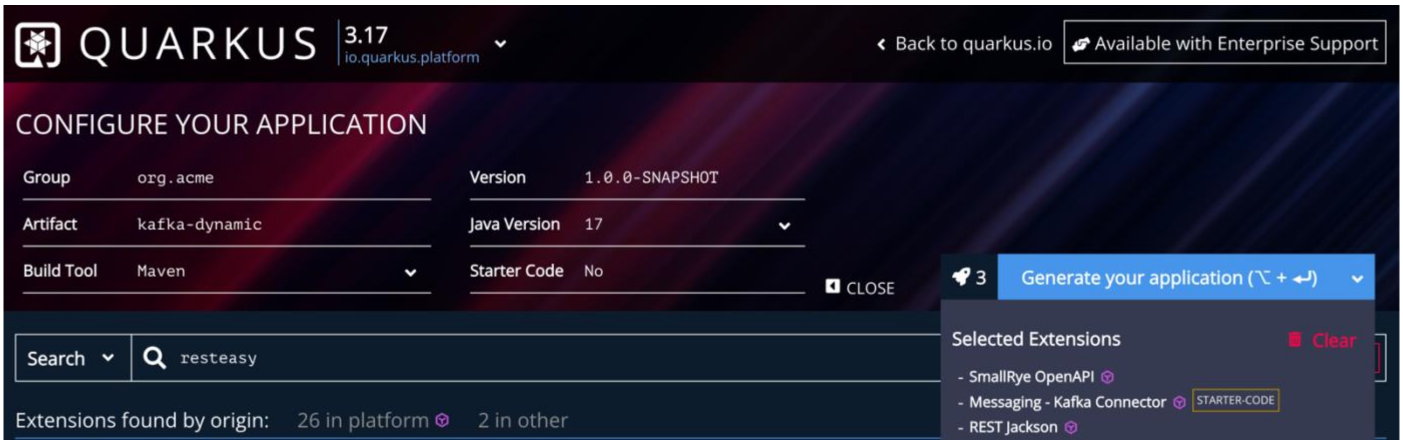
```
cd kafka-quickstart-producer

./mvnw quarkus:dev
```

H.5.Navigate to http://localhost:8080/quotes.html

I. Consuming a Kafka Topic that changes dynamically

In Quarkus Kafka, the @Incoming annotation defines a method that acts as a Kafka consumer. You are not allowed to change it dynamically at runtime, as the topic is determined at compile time. A simple workaround solution is to use the old JAVA way of dealing with consuming Kafka messages. For exemplification consider a REST API exposing a POST where the topic is received, then a new JAVA Thread is started containing the Kafka consuming source code for that topic.

I.1. Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.



CONFIGURE YOUR APPLICATION

Group org.acme
Version 1.0.0-SNAPSHOT
Artifact kafka-dynamic
Java Version 17
Build Tool Maven
Starter Code No

3

resteasy

Extensions found by origin: 26 in platform 2 in other

Selected Extensions
- SmallRye OpenAPI
- Messaging - Kafka Connector
- REST Jackson

And then, add the following files to your project.

I.2. The KafkaProvisioningResource.java to src/main/java/org/acme/KafkaProvisioningResource.java

```
package org.acme;

import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;
import org.acme.model.Topic;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/createTopic")
public class KafkaProvisioningResource {

    @ConfigProperty(name = "kafka.bootstrap.servers")
    String kafka_servers;

    @POST
    @Produces(MediaType.TEXT_PLAIN)
    public String ProvisioningConsumer(Topic topic) {
        Thread worker = new DynamicTopicConsumer(topic.TopicName , kafka_servers );
        worker.start();
        return "New worker started";
    }
}
```

I.3. The DynamicTopicConsumer.java to src/main/java/org/acme/DynamicTopicConsumer.java

```
package org.acme;

import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class DynamicTopicConsumer extends Thread  {
    private String kafka_servers;
    private String topic;

    public DynamicTopicConsumer(String topic_received , String kafka_servers_received)
    {
        topic = topic_received;
        kafka_servers = kafka_servers_received;
    }

    public void run()
       {
           try
              {
            Properties properties = new Properties();
            properties.put("bootstrap.servers", kafka_servers);
            properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
            properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
            properties.put("group.id", "your-group-id");

            try (Consumer<String, String> consumer = new KafkaConsumer<>(properties)) {
                consumer.subscribe(Collections.singletonList(topic));

                while (true) {
                    ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
                    for (ConsumerRecord<String, String> record : records)
                    {
                      System.out.printf("topic = %s, partition = %s, offset = %d,key = %s, value = %s\n",
                      record.topic(), record.partition(), record.offset(),
                      record.key(), record.value());
                    }
                }
            }
        }
        catch (Exception e)
               {  System.out.println("Exception is caught");  }
    }
}
```

I.4. The Topic.java to src/main/java/org/acme/Topic.java

```
package org.acme.model;

public class Topic {
    public String TopicName;
    public Topic() {  }
    public Topic(String topicName) {  TopicName = topicName;  }
    @Override
    public String toString()
    {  return "Topic [TopicName=" + TopicName + "]";  }
}
```

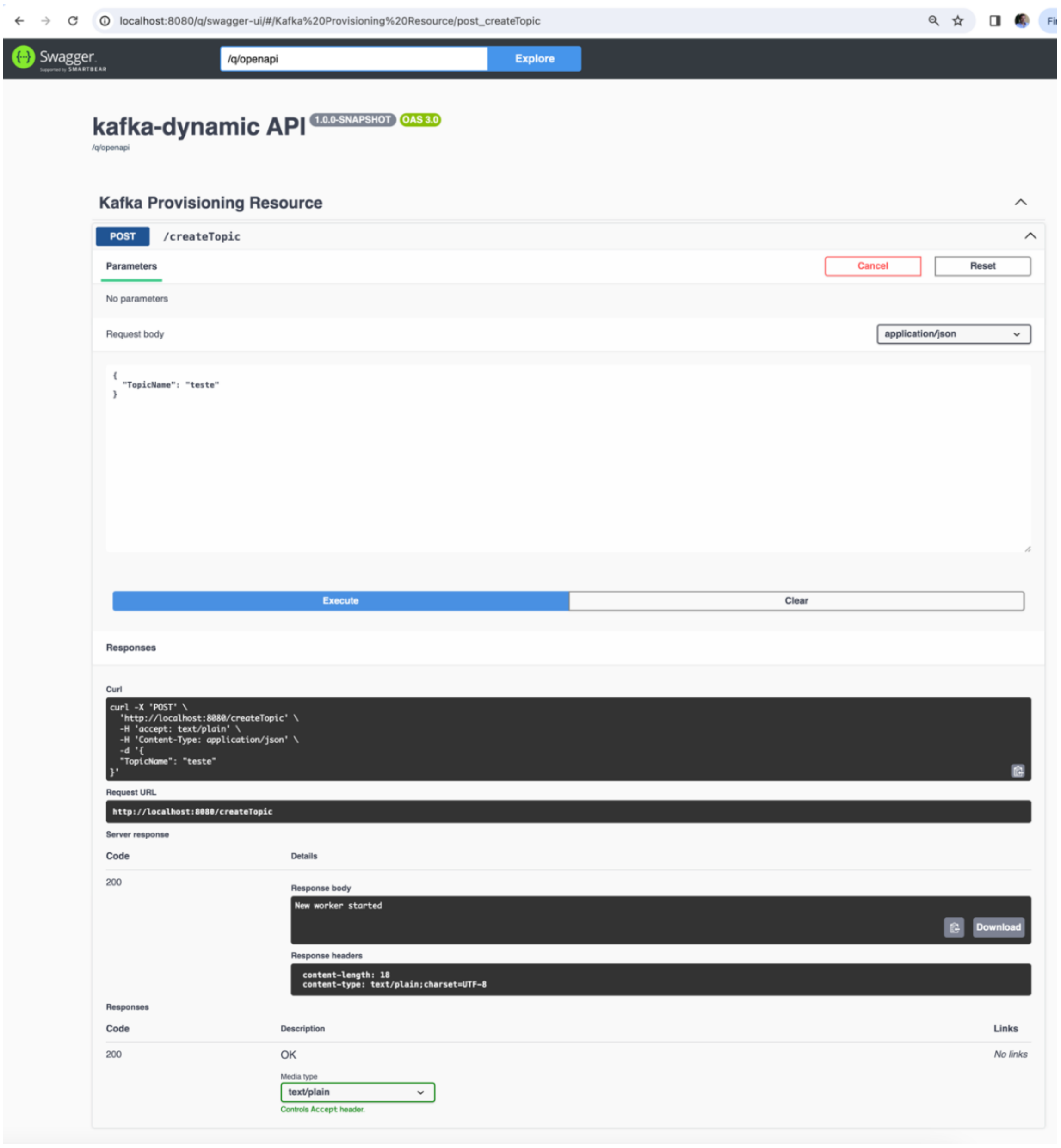
I.5. Add the following configuration in src/main/resources/application.properties

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

kafka.bootstrap.servers=<YOUR KAFKA NAME>:9092
```

I.6. Execute in a separate command line

```
./mvnw quarkus:dev
```

I.7. Navigate to http://localhost:8080/q/swagger-ui/ and execute the POST API. Then, check that the new kafka topic is being consumed, sending some new messages to that topic. At this moment, you can change dynamically the topic that is being consumed.


localhost:8080/q/swagger-ui/#/Kafka%20Provisioning%20Resource/post_createTopic
Swagger
/q/openapi
Explore

kafka-dynamic API 1.0.0-SNAPSHOT OAS 3.0

/q/openapi

Kafka Provisioning Resource

POST /createTopic

No parameters

Request body application/json

```
{
  "TopicName": "teste"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/createTopic' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
  "TopicName": "teste"
}'
```

Request URL

```
http://localhost:8080/createTopic
```

Server response

Code	Details
200	Response body: `New worker started`

Response headers

```
content-length: 18
content-type: text/plain;charset=UTF-8
```

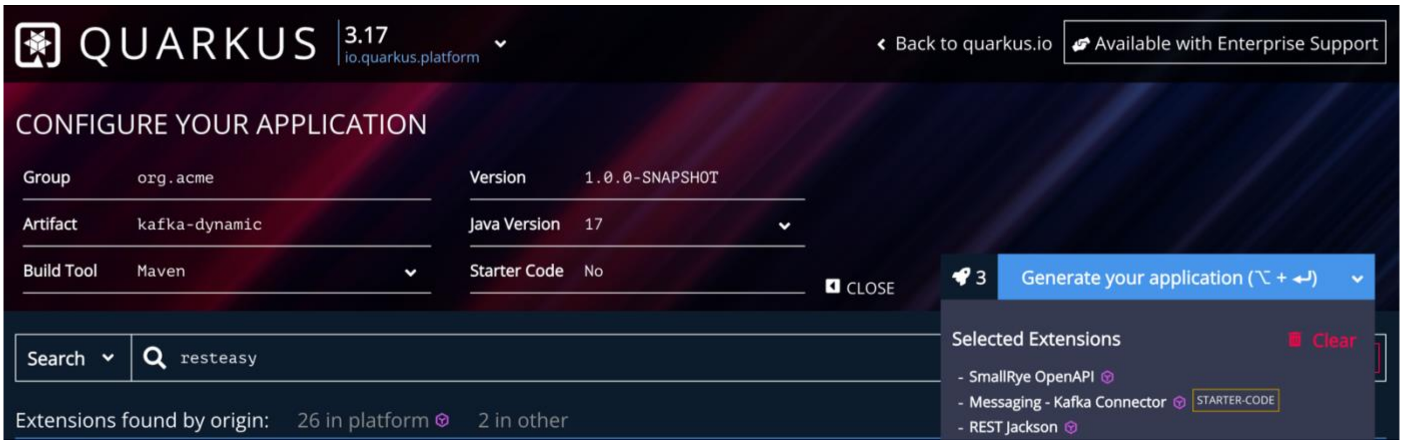
Responses

Code	Description	Links
200	OK Media type: text/plain Controls Accept header.	*No links*

J. Creating and Removing Kafka Topics dynamically

Before starting the kafka message consumption it is required to create the specific kafka topic. Using a microservice approach, consider a REST API exposing a POST where the topic, to be created or deleted, is received, then the library kafka-client provides APIs to perform the desired operation on a kafka cluster.

J.1. Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.


QUARKUS
3.17
io.quarkus.platform
Back to quarkus.io
Available with Enterprise Support
CONFIGURE YOUR APPLICATION
CLOSE
3
Generate your application (⌥ + ↵)
Clear
Search
Extensions found by origin:
26 in platform
2 in other

Group org.acme
Artifact kafka-dynamic
Build Tool Maven
Version 1.0.0-SNAPSHOT
Java Version 17
Starter Code No
resteasy

Selected Extensions
- SmallRye OpenAPI
- Messaging - Kafka Connector STARTER-CODE
- REST Jackson

And then, add the following files to your project.

J.2. The KafkaProvisioningResource.java to src/main/java/org/acme/KafkaProvisioningResource.java

```
package org.acme;

import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ExecutionException;
import org.acme.model.Topic;
import org.eclipse.microprofile.config.inject.ConfigProperty;
import org.apache.kafka.clients.admin.AdminClient;
import org.apache.kafka.clients.admin.CreateTopicsResult;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.clients.admin.DeleteTopicsResult;
import org.apache.kafka.common.errors.TopicExistsException;
import org.apache.kafka.common.errors.UnknownTopicOrPartitionException;

@Path("/TopicManagement")
public class KafkaProvisioningResource {

    @ConfigProperty(name = "kafka.bootstrap.servers")
    String kafka_servers;

    @POST
    @Path("/createTopic")
    @Produces(MediaType.TEXT_PLAIN)
    public String CreateNewTopic(Topic topic) {
           Properties properties = new Properties();
           properties.put("bootstrap.servers", kafka_servers);
           properties.put("connections.max.idle.ms", 10000);
           properties.put("request.timeout.ms", 5000);
           try (AdminClient client = AdminClient.create(properties))
        {
            CreateTopicsResult result = client.createTopics(Arrays.asList(
                        new NewTopic(topic.TopicName, 1, (short) 1)  ));
            try {    result.all().get();    }
            catch ( org.apache.kafka.common.errors.TopicExistsException exc )
            {   throw exc; }
            catch (  ExecutionException | InterruptedException e )
            {   throw new IllegalStateException(e);   }
           }
        return ("New Topic created = " + topic);
    }
```

P6-Quarkus_v1.9.docx

```
    @POST
    @Path("/DeleteTopic")
    @Produces(MediaType.TEXT_PLAIN)
    public String RemoveOneTopic(Topic topic) {
           Properties properties = new Properties();
           properties.put("bootstrap.servers", kafka_servers);
           properties.put("connections.max.idle.ms", 10000);
           properties.put("request.timeout.ms", 5000);
           try (AdminClient client = AdminClient.create(properties))
        {
            DeleteTopicsResult result = client.deleteTopics(Arrays.asList( topic.TopicName ));
            try {    result.all().get();    }
            catch ( org.apache.kafka.common.errors.UnknownTopicOrPartitionException exc)
            { throw exc; }
            catch (  ExecutionException | InterruptedException e )
            {   throw new IllegalStateException(e);  }
           }
        return ("New Topic deleted = " + topic);
    }
}
```

J.3. The Topic.java to src/main/java/org/acme/Topic.java

```
package org.acme.model;

public class Topic {
    public String TopicName;
    public Topic() {  }
    public Topic(String topicName) {  TopicName = topicName;  }
    @Override
    public String toString()
    {  return "Topic [TopicName=" + TopicName + "]";  }
}
```

J.4. Add the following configuration in src/main/resources/application.properties

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

kafka.bootstrap.servers=<YOUR KAFKA NAME>:9092
```

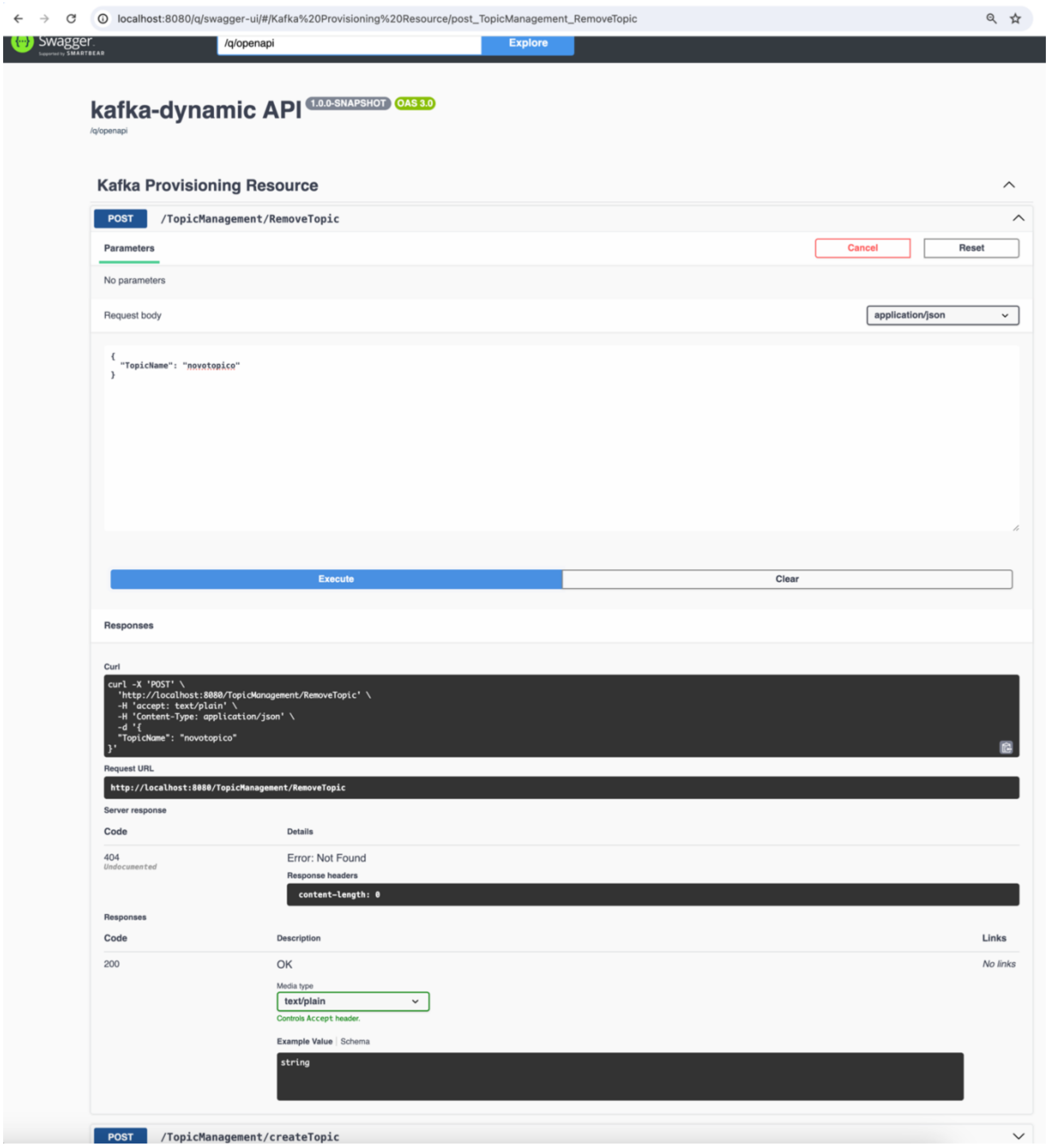
J.5. Add the following dependency in pom.xml

```
    <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
    </dependency>
```

J.6. Execute in a separate command line

```
./mvnw quarkus:dev
```

Navigate to http://localhost:8080/q/swagger-ui/ and execute the POST APIs. Then, check that the new kafka topic is created or deleted, listing the topics in that kafka cluster.


localhost:8080/q/swagger-ui/#/Kafka%20Provisioning%20Resource/post_TopicManagement_RemoveTopic

kafka-dynamic API 1.0.0-SNAPSHOT OAS 3.0
/q/openapi

Kafka Provisioning Resource

POST /TopicManagement/RemoveTopic

No parameters

Request body: application/json

```
{
  "TopicName": "novotopico"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/TopicManagement/RemoveTopic' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
  "TopicName": "novotopico"
}'
```

Request URL

```
http://localhost:8080/TopicManagement/RemoveTopic
```

Server response

Code	Details
404 *Undocumented*	Error: Not Found Response headers `content-length: 0`

Responses

Code	Description	Links
200	OK Media type: text/plain Controls Accept header. Example Value | Schema `string`	*No links*

POST /TopicManagement/createTopic

P6-Quarkus_v1.9.docx

K. Deploying QUARKUS in AWS EC2 environment with github and terraform!

K.1. Create your new Quarkus project as before.

K.2. Execute the following command in your project directory to add the docker dependencies:

```
./mvnw quarkus:add-extension -Dextensions="quarkus-container-image-docker"
```

K.3. Add the following configuration to **application.properties** in order to include all the dependencies to execute in a different environment from dev:

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

quarkus.container-image.build=true
quarkus.container-image.push=true
```

Also, add your Git Hub accountID in the **application.properties** of your project, to identify it correctly and push it directly:

```
quarkus.container-image.group=YOUR DOCKER USERNAME
```

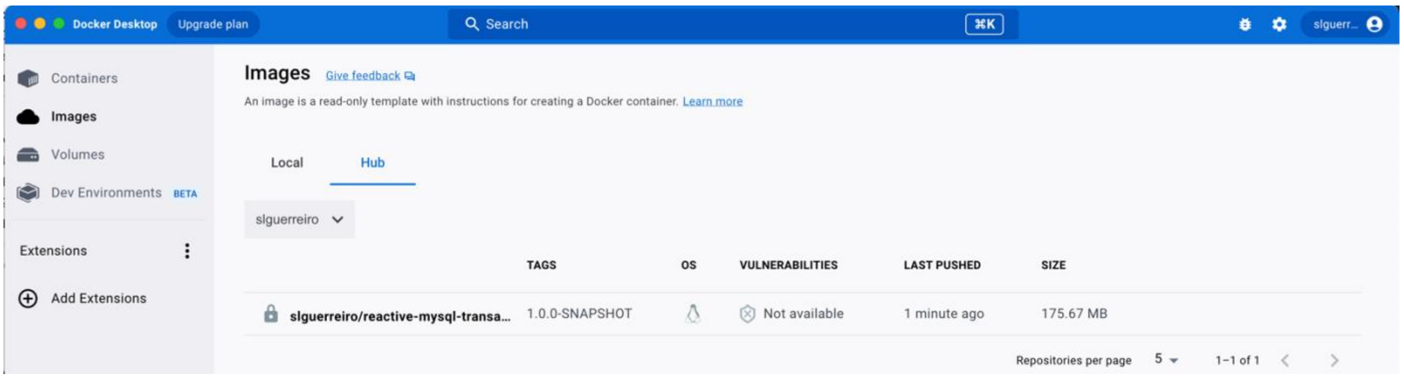
K.4. Login in your account (you can create one in http://hub.docker.com if needed)

```
                          docker login -u "your docker username" -p "your password"
```

K.5. Build docker image and push it directly to you github account:

```
./mvnw clean package
```

K.6. Check that image has been created successfully on github or on the docker dashboard (optional):



Images Give feedback

An image is a read-only template with instructions for creating a Docker container. Learn more

	TAGS	OS	VULNERABILITIES	LAST PUSHED	SIZE
slguerreiro/reactive-mysql-transa...	1.0.0-SNAPSHOT		Not available	1 minute ago	175.67 MB

K.7. Now, to deploy your Quarkus project in EC2, using terraform, and based on the image previously pushed to git, create a .tf file with the following instructions. Note that the AMI to create should be compatible with the operating systems where you previously compiled the docker image.

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region        = "us-east-1"
  access_key    = "YOUR ACESS KEY"
  secret_key    = "YOUR SECRET KEY"
  token         = "YOUR TOKEN"
}
```

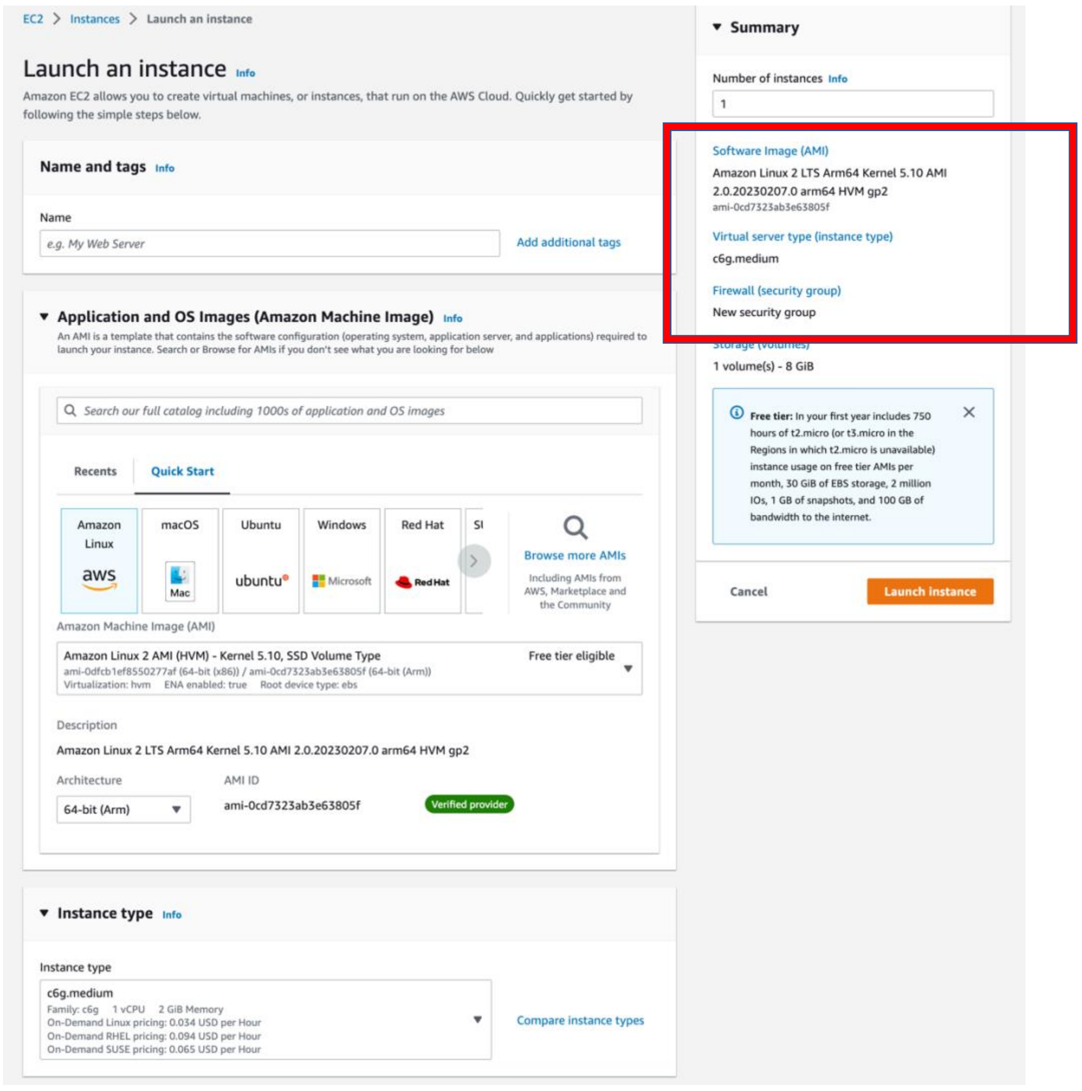
```
resource "aws_instance" "exampleDeployQuarkus" {
  ami                    = "ami-08cf815cff6ee258a" # Amazon Linux ARM AMI built by Amazon Web Services - FOR
DOCKER image COMPATIBILITY if compiled previously on ARM
  instance_type          = "t4g.nano"
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name               = "vockey"

  user_data = "${file("quarkus.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-deploy-QuarkusProject"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-Quarkus-instance"
}

output "address" {
  value       = aws_instance.exampleDeployQuarkus.public_dns
  description = "Address of the Quarkus EC2 machine"
}
```

And the correspondingly script "`quarkus.sh`" with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum install -y docker

sudo service docker start

sudo docker login -u "YOUR DOCKER USERNAME" -p "YOUR DOCKER PASSWORD"

sudo docker pull YOUR DOCKER USERNAME/tryout1:1.0.0-SNAPSHOT

sudo docker run -d --name tryout2 -p 9000:9000 YOUR DOCKER USERNAME/YOUR ARTIFACT ID:1.0.0-SNAPSHOT

echo "Finished."
```

Docker images built for ARM architecture may not work on AMD architecture machines. This is because ARM and AMD processors use different instruction sets and have different hardware architectures.

For this example, the AMI identifier and instance type were extracted from the launching instance AWS GUI.


EC2 > Instances > Launch an instance

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: e.g. My Web Server

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images
Recents
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type — Free tier eligible
ami-0dfcb1ef8550277af (64-bit (x86)) / ami-0cd7323ab3e63805f (64-bit (Arm))
Virtualization: hvm   ENA enabled: true   Root device type: ebs

Description: Amazon Linux 2 LTS Arm64 Kernel 5.10 AMI 2.0.20230207.0 arm64 HVM gp2

Architecture: 64-bit (Arm)

AMI ID: ami-0cd7323ab3e63805f — Verified provider

▼ Instance type Info

Instance type: c6g.medium
Family: c6g   1 vCPU   2 GiB Memory
On-Demand Linux pricing: 0.034 USD per Hour
On-Demand RHEL pricing: 0.094 USD per Hour
On-Demand SUSE pricing: 0.065 USD per Hour

▼ Summary

Number of instances Info: 1

Software Image (AMI): Amazon Linux 2 LTS Arm64 Kernel 5.10 AMI 2.0.20230207.0 arm64 HVM gp2
ami-0cd7323ab3e63805f

Virtual server type (instance type): c6g.medium

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

References

Full official documentation:

- https://quarkus.io/guides/
- https://quarkus.io/guides/container-image#building
- https://dev.mysql.com/downloads/
- https://quarkus.io/guides/reactive-sql-clients
- https://quarkus.io/guides/kafka-reactive-getting-started
- https://quarkus.io/guides/amazon-lambda
- https://aws.amazon.com/pt/blogs/aws-brasil/integracao-do-aws-lambda-com-o-quarkus/
- https://dev.to/marcuspaulo/tutorial-publish-a-quarkus-application-in-kubernetes-minikube-and-dockerhub-36nd
- https://stackoverflow.com/questions/23935141/how-to-copy-docker-images-from-one-host-to-another-without-using-a-repository
- https://quarkus.io/guides/container-image#quarkus-container-image-docker_quarkus.docker.executable-name