

# ICWA - Camel 4 Hands-On

Install VSCode extensions:

- Language Support for Apache Camel by Red Hat
- Debug Adapter for Apache Camel by Red Hat

## 1.0 Camel 4 on Quarkus

<https://github.com/apache/camel-quarkus-examples>

### 1.1 A Simple Rest Service using Camel Rest DSL

This project illustrates how to use the Camel Rest DSL component to build a simple Rest API.

Generate your first Quarkus project at site:

Go to: <https://code.quarkus.redhat.com/>

Select:

- camel-quarkus-rest
- camel-quarkus-jackson
- Camel-quarkus-direct

Generate and download the generated project as a zip file.

Unzip the file and change the directory name from “code-with-quarkus” to “simple-rest-quarkus”

Open that folder using your VSCode IDE

Open the pom.xml file

Change the line from:

```
Unset
<artifactId>code-with-quarkus</artifactId>
```

To

```
Unset
<artifactId>simple-rest-quarkus</artifactId>
```

Add path: com/redhat to src/main/java

Add Routes.java with content:

Java

```
package com.redhat;

import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.Set;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.rest.RestBindingMode;

public class Routes extends RouteBuilder {

    private final Set<Fruit> fruits = Collections.synchronizedSet(new
LinkedHashSet<>());
    public Routes() {

        /* Add some initial fruits */
        this.fruits.add(new Fruit("Apple", "Winter fruit"));
        this.fruits.add(new Fruit("Pineapple", "Tropical fruit"));
    }

    @Override
    public void configure() throws Exception {
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/fruits")
            .get()
            .to("direct:getFruits")

            .post()
            .type(Fruit.class)
            .to("direct:addFruit");

        from("direct:getFruits")
            .setBody().constant(fruits);

        from("direct:addFruit")
            .process().body(Fruit.class, fruits::add)
            .setBody().constant(fruits);
    }
}
```

Add file Fruit.java with content:

```
Java
package com.redhat;

import java.util.Objects;
import io.quarkus.runtime.annotations.RegisterForReflection;

@RegisterForReflection // Lets Quarkus register this class for reflection during
the native build
public class Fruit {
    private String name;
    private String description;

    public Fruit() {
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Fruit)) {
            return false;
        }
    }
}
```

```

        Fruit other = (Fruit) obj;

        return Objects.equals(other.name, this.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.name);
    }
}

```

Add path: test/java/com/redhat to src  
 Add file RestTest.java with content:

```

Java
package com.redhat;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.Matchers.containsInAnyOrder;

@QuarkusTest
public class RestTest {

    @Test
    public void fruits() {

        /* Assert the initial fruits are there */
        given()
            .when().get("/fruits")
            .then()
            .statusCode(200)
            .body(
                "$.size()", is(2),
                "name", containsInAnyOrder("Apple", "Pineapple"),
                "description", containsInAnyOrder("Winter fruit", "Tropical fruit"));

        /* Add a new fruit */
    }
}

```

```

given()
    .body("{\"name\": \"Pear\", \"description\": \"Winter fruit\"}")
    .header("Content-Type", "application/json")
    .when()
    .post("/fruits")
    .then()
    .statusCode(200)
    .body(
        "$.size()", is(3),
        "name", containsInAnyOrder("Apple", "Pineapple", "Pear"),
        "description", containsInAnyOrder("Winter fruit", "Tropical fruit",
"Winter fruit"));
    }
}

```

And add the following dependency to the pom.xml file:

Note: In Quarkus, a dependency in the pom.xml file is called an extension. There are 3 different ways to add an extension. This is one of them. The other 2 will be shown in a later section.

```

Unset
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>

```

Perform Unit testing from terminal:

```

Unset
./mvnw test

```

Enter Quarkus dev mode and compile/run the app from a terminal:

```

Unset
./mvnw clean compile quarkus:dev

```

It takes a while to download all dependencies the first time.

From browser: <http://localhost:8080/fruits>

Command to get all fruits:

```
Unset
curl http://localhost:8080/fruits 2>/dev/null | python -m json.tool
Or
curl http://localhost:8080/fruits 2>/dev/null | jq
```

Output should be:

```
Unset
[
  {
    "name": "Apple",
    "description": "Winter fruit"
  },
  {
    "name": "Pineapple",
    "description": "Tropical fruit"
  }
]
```

Command to add a new fruit:

```
Unset
curl -X POST http://localhost:8080/fruits \
  -H 'Content-Type: application/json' \
  -d '{"name": "Avocado", "description": "Super fruit"}' 2>/dev/null | python -m
json.tool

Or

curl -X POST http://localhost:8080/fruits \
  -H 'Content-Type: application/json' \
  -d '{"name": "Orange", "description": "Popular fruit"}' 2>/dev/null | jq
```

Output should be:

```
Unset
[
  {
    "name": "Apple",
    "description": "Winter fruit"
  },
  {
    "name": "Pineapple",
    "description": "Tropical fruit"
  },
  {
    "name": "Avocado",
    "description": "Super fruit"
  }
]
```

## **Deploying to OpenShift**

### ***Deploy the Application***

Install extension:

```
Unset
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-openshift"
```

Add to pom.xml

```
Unset
<maven.compiler.source>17</maven.compiler.source>
<maven.compiler.target>17</maven.compiler.target>
```

To avoid certificate problems, add the following entry to the application.properties file:

```
Unset
quarkus.kubernetes-client.trust-certs=true
```

And execute the following command:

```
Unset
oc login -u yourUserName OpenShiftApiServerURL

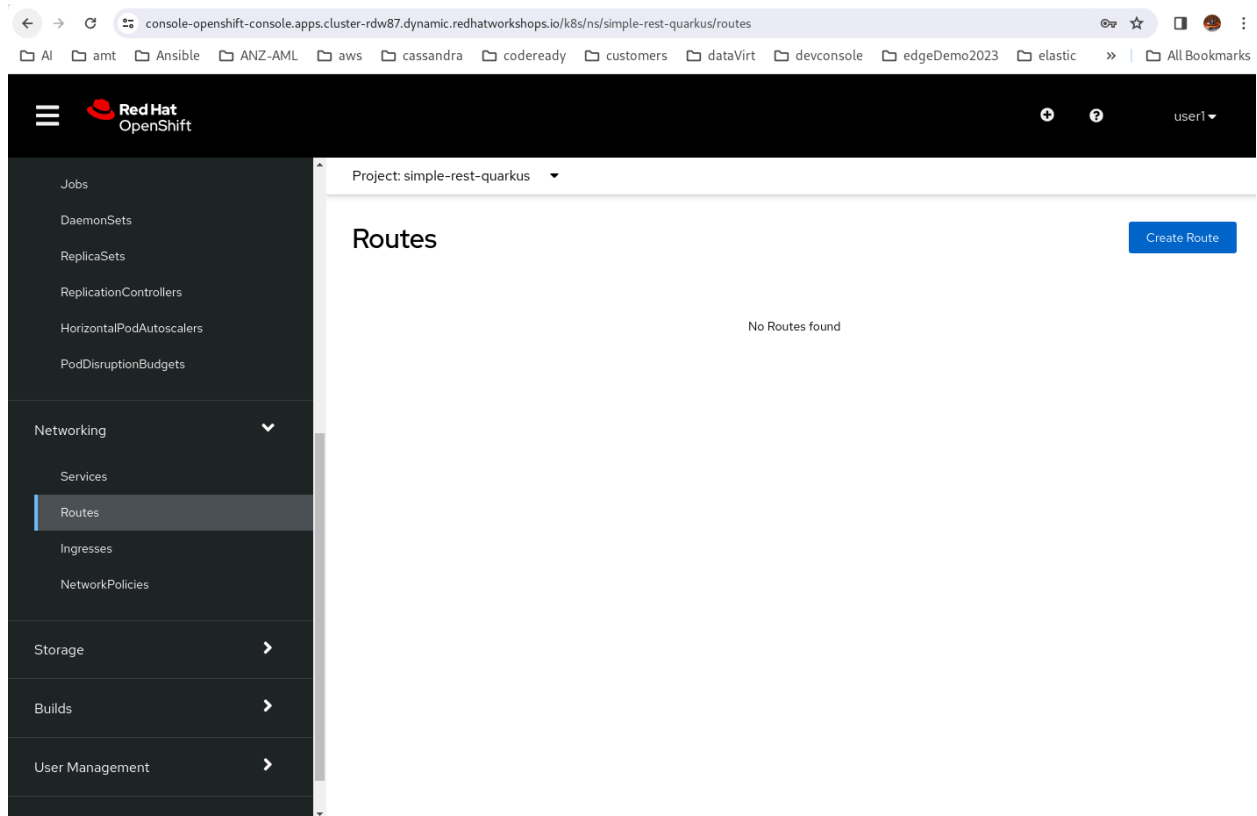
oc new-project simple-rest-quarkus

./mvnw clean install -Dquarkus.kubernetes.deploy=true -DskipTests
```

Log in the the OpenShift console:

Select the Administrator tab and then Networking->Routes.

Create a route for the service which has been created by the mvnw command above.



Click on 'Create Route'.



← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/k8s/ns/simple-rest-quarkus/routes/~new 🔍 ☆ 📱 🧑

📁 AI 📁 amt 📁 Ansible 📁 ANZ-AML 📁 aws 📁 cassandra 📁 codeready 📁 customers 📁 dataVirt 📁 devconsole 📁 edgeDemo2023 📁 elastic » 📁 All Bookmarks

☰

Red Hat OpenShift

Deployments

DeploymentConfigs

StatefulSets

Secrets

ConfigMaps

CronJobs

Jobs

DaemonSets

ReplicaSets

ReplicationControllers

HorizontalPodAutoscalers

PodDisruptionBudgets

Networking ▾

Services

Routes

Ingresses

NetworkPolicies

Storage >

Builds >

User Management >

Administration >

Project: simple-rest-quarkus ▾

Create Route

Routing is a way to make your application publicly visible.

Configure via: ☒ Form view ☐ YAML view

Name \*

rest

A unique name for the Route within the project.

Hostname

www.example.com

Public hostname for the Route. If not specified, a hostname is generated.

Path

/

Path that the router watches to route traffic to the service.

Service \*

simple-rest-quarkus

Service to route to.

Target port \*

80 → 8080 (TCP)

Target port for traffic.

Security

☒ Secure Route

Routes can be secured using several TLS termination types for serving certificates.

TLS termination \*

Edge

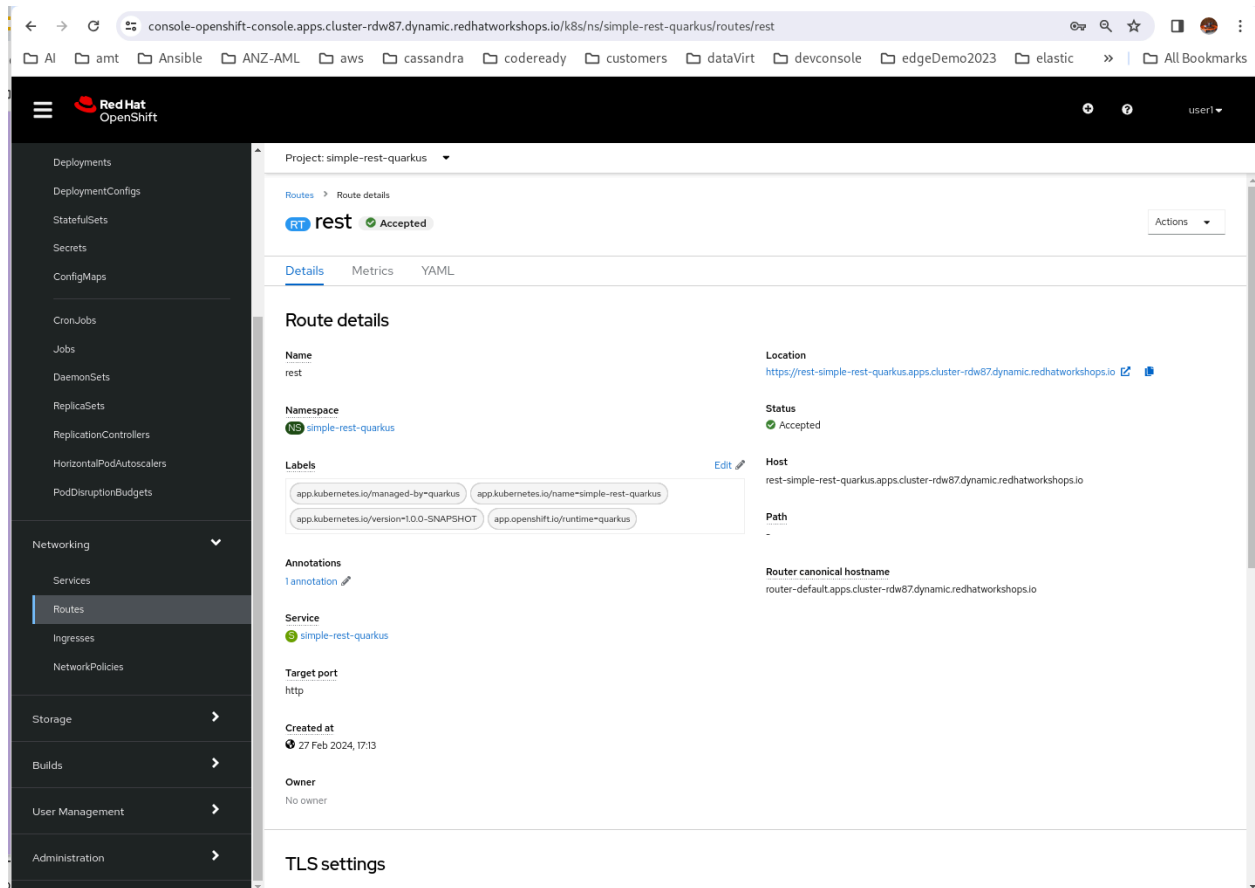
Insecure traffic

Select insecure traffic type

Create

Cancel

Enter the name and select from the drop down boxes as shown above and click 'Create'



The route is shown at the 'Location'. Use it to interact with your application.

### Testing:

Similar to what you did when testing locally. Only the URL is different. Your URL will be different when running in a different OpenShift environment.

Unset

```
curl
```

```
https://rest-simple-rest-quarkus.apps.cluster-rdw87.dynamic.redhatworkshops.io/fruits 2>/dev/null | jq
```

```
curl -X POST
```

```
https://rest-simple-rest-quarkus.apps.cluster-rdw87.dynamic.redhatworkshops.io/fruits \
```

```
-H 'Content-Type: application/json' \
```

```
-d '{"name": "Orange", "description": "Popular fruit"}' 2>/dev/null | jq
```

## 1.2 A Simple Camel SQL Rest Service

This project shows you how one can use the Camel Rest DSL together with the SQL component to create a database application.

You can generate another project at <https://code.quarkus.redhat.com/>, or you can make a copy of the previous project and start from there. Change the name of the copy's directory to simple-sql-quarkus.

Open that folder using your VSCode IDE  
Open the pom.xml file  
Change the line from:

```
Unset  
<artifactId>simple-rest-quarkus</artifactId>
```

To

```
Unset  
<artifactId>simple-sql-quarkus</artifactId>
```

Delete the simple-rest-quarkus/src/main/java/com/redhat/Fruit.java file.

We need to add Quarkus extensions (dependencies in the maven pom.xml file) for this project. There are multiple ways of adding a Quarkus extension:

```
Unset  
quarkus ext add org.apache.camel.quarkus:camel-quarkus-sql
```

Or

Unset

```
./mvnw quarkus:add-extension  
-Dextensions="org.apache.camel.quarkus:camel-quarkus-sql"
```

Or add dependency directly to pom.xml

Unset

```
<dependency>  
  <groupId>org.apache.camel.quarkus</groupId>  
  <artifactId>camel-quarkus-sql</artifactId>  
</dependency>
```

We are going to add the following extensions using the second approach although any of the mentioned ways will work.

Unset

```
./mvnw quarkus:add-extension  
-Dextensions="org.apache.camel.quarkus:camel-quarkus-sql"  
./mvnw quarkus:add-extension -Dextensions="jdbc-h2"
```

Replace the content of Router.java with that shown below:

Java

```
package com.redhat;  
  
import org.apache.camel.builder.RouteBuilder;  
import org.apache.camel.model.rest.RestBindingMode;  
  
import jakarta.inject.Inject;  
import org.apache.camel.model.rest.RestParamType;
```

```

public class Router extends RouteBuilder {

    @Override
    public void configure() {

        // Accept the default host and port
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/")
            .get("customers")
            .produces("application/json")
            .to("direct:getallcustomer");

        rest("/customer")
            .get("{custid}")
            .produces("application/json")
            .to("direct:getcustomer");

        rest("/customer")
            .post()
            .produces("application/json")
            .to("direct:addcustomer");

        from("direct:getallcustomer")
            .routeId("allcustomer")
            .to("sql:select * from customerdemo")
            .log("getallcustomer result: ${body}")
            .end();

        from("direct:getcustomer")
            .routeId("specificcustomer")
            .log("getcustomer - ${header.custid}")
            .to("sql:select * from customerdemo where customerID =
:#{header.custid}")
            .log("specificcustomer: ${body}")
            .end();

        from("direct:addcustomer")
            .routeId("addnewcustomer")
            .setHeader("customerID", simple("customerID"))
            .setHeader("vipStatus", simple("vipStatus"))
            .setHeader("balance", simple("balance"))
    }
}

```

```

        .to("sql:insert into customerdemo (customerID,vipStatus,balance)
VALUES (:#customerID, :#vipStatus, :#balance)")
        .log("addcustomer: ${body}")
        .end();
    }
}

```

Replace the content of the simple-rest-quarkus/src/main/resources/application.properties file with that shown below:

```

Unset
%dev.quarkus.datasource.db-kind=h2
%dev.quarkus.datasource.username=sa
%dev.quarkus.datasource.password=
%dev.quarkus.datasource.jdbc.url=jdbc:h2:mem:mydb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;DATABASE_TO_UPPER=false;INIT=RUNSCRIPT FROM 'classpath:schema.sql'
%dev.quarkus.datasource.jdbc.max-size=12

```

Create a new file named simple-rest-quarkus/src/main/resources/schema.sql and paste in the content below:

```

Unset
DROP TABLE IF EXISTS customerdemo;
CREATE TABLE customerdemo (
    customerID varchar(10) NOT NULL,
    vipStatus varchar(10) NOT NULL ,
    balance integer NOT NULL
);
INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES
('A01','Diamond',1000);
INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES
('A02','Gold',500);

```

Run project in Quarkus dev mode:

Unset

```
./mvnw clean compile quarkus:dev
```

In another terminal, run the following commands to:

1. List all customers in the database. Remove the “| jq” part if you don’t have jq installed. Jq beautifies the output json. Without it, you will still see the result.

Unset

```
curl http://localhost:8080/customers 2>/dev/null | jq
```

You will see the result below:

Unset

```
[
  {
    "customerID": "A01",
    "vipStatus": "Diamond",
    "balance": 1000
  },
  {
    "customerID": "A02",
    "vipStatus": "Gold",
    "balance": 500
  }
]
```

2. List a specific customer.

Unset

```
curl http://localhost:8080/customer/A01 2>/dev/null | jq
```

With result:

Unset

```
[
```

```
{
  "customerID": "A01",
  "vipStatus": "Diamond",
  "balance": 1000
}
]
```

### 3. Add a new customer:

```
Unset
curl -X POST http://localhost:8080/customer \
  -H 'Content-Type: application/json' \
  -d '{"customerID": "A04", "vipStatus": "Silver", "balance": 300 }'
2>/dev/null | jq
```

With result

```
Unset
{
  "customerID": "A04",
  "vipStatus": "Silver",
  "balance": 300
}
```

If you run the list all customers command again, you will see that customer A04 has been added to the database.

```
Unset
[
  {
    "customerID": "A01",
    "vipStatus": "Diamond",
    "balance": 1000
  },
  {
    "customerID": "A02",
    "vipStatus": "Gold",
    "balance": 500
  },
  {
    "customerID": "A04",
    "vipStatus": "Silver",
    "balance": 300
  }
]
```



```
{
  "customerID": "A04",
  "vipStatus": "Silver",
  "balance": 300
}
```

### Observed Limitations

A database application usually supports CRUD (Create, Read, Update, Delete) operations. Our application only supports C and R.

You may have noticed that you can add the same customer more than once.

### Suggested TODO List

Here is a list of enhancements that you may work on yourselves.

1. Add an Update operation
2. Add a Delete operation
3. The add customer endpoint currently only creates one customer per call. Investigate how to process a list of customers.
4. Disallow the same customer to be entered multiple times. HINT: you have to change the database schema.

### Deploying to OpenShift

Log in to OpenShift and create a new project:

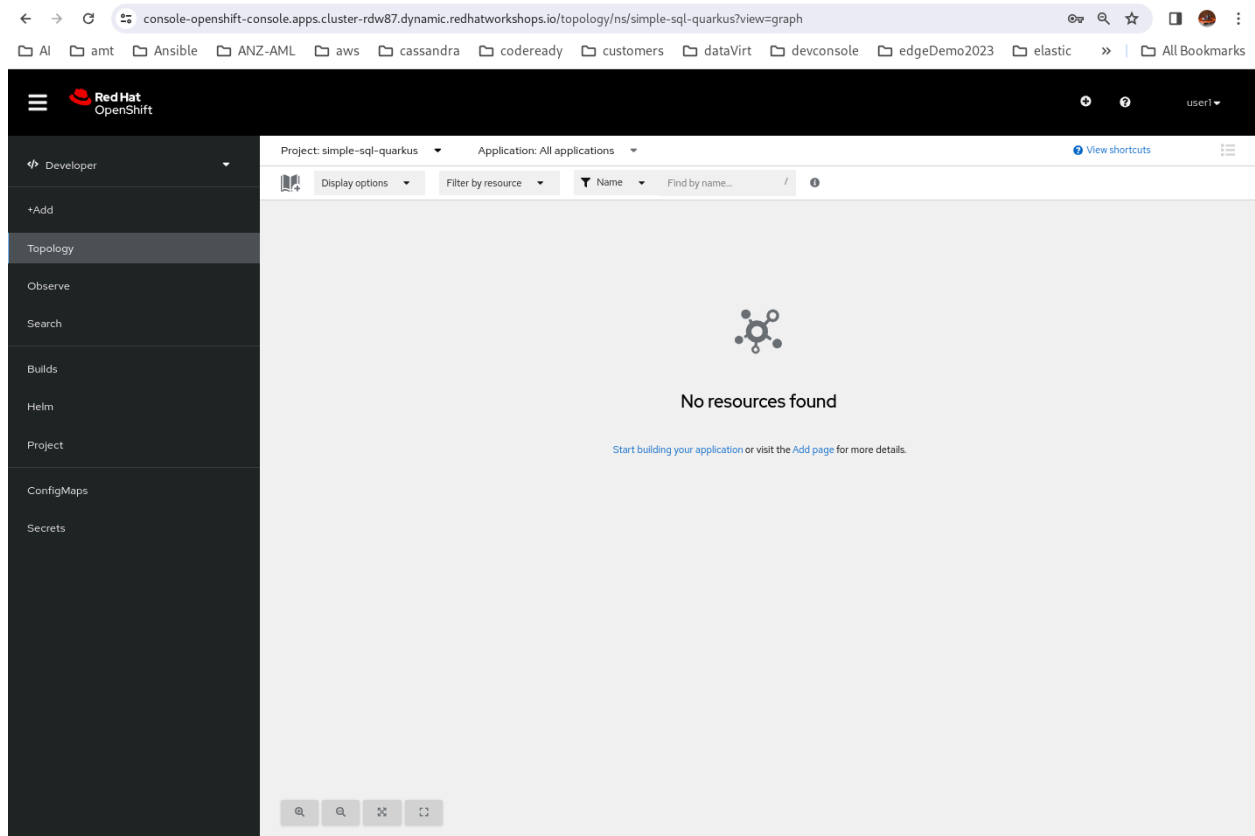
```
Unset
oc login -u yourUserName OpenShiftApiServerURL

oc new-project simple-sql-quarkus
```

### ***Deploy a MySQL Database and Populate It***

Before you deploy your Quarkus app, you have to install a MySQL database. H2 is not meant for production. However, you can continue to use the H2 database when deploying to OpenShift. If that is what you want, you may skip this part and jump directly to the section: Deploy Your Quarkus App.

Log into the OpenShift Console and change to the Developer tab.



Select +Add on the left-hand pane.

← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/add/ns/simple-sql-quarkus 🔍 ⌵ 🧑 user1

📁 AI 📁 amt 📁 Ansible 📁 ANZ-AML 📁 aws 📁 cassandra 📁 codeready 📁 customers 📁 dataVirt 📁 devconsole 📁 edgeDemo2023 📁 elastic » 📁 All Bookmarks

☰ Red Hat OpenShift

Developer ▾

+Add

Topology

Observe

Search

Builds

Helm

Project

ConfigMaps

Secrets

Project: simple-sql-quarkus ▾

Add

🔍 Details on

Getting started resources ⓘ

Create applications using samples

Choose a code sample to get started creating an application with.

Basic Quarkus →

Basic Spring Boot →

View all samples

Build with guided documentation

Follow guided documentation to build applications and familiarize yourself with key features.

Get started with Quarkus using s2i →

Get started with Spring →

View all quick starts

Explore new developer features

Explore new features and resources within the developer perspective.

Discover certified Helm Charts →

Start building your application quickly in topology →

What's new in OpenShift ↗

Developer Catalog

All services

Browse the catalog to discover, deploy and connect to services

Database

Browse the catalog to discover database services to add to your application

Operator Backed

Browse the catalog to discover and deploy operator managed services

Git Repository

Import from Git

Import code from your Git repository to be built and deployed

Container images

Deploy an existing Image from an image registry or image stream tag

From Local Machine

Import YAML

Create resources from their YAML or JSON definitions

Upload JAR file

Sharing

Project access allows you to add or remove a user's access to the project

Samples

Create an application from a code sample

Select Database and then MySQL (Ephemeral).

← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/catalog/ns/simple-sql-quarkus?category=databases 🔍 ☆ 📱 🌐

AI amt Ansible ANZ-AML aws cassandra codeready customers dataVirt devconsole edgeDemo2023 elastic » All Bookmarks

☰

Red Hat OpenShift

user1 ▼

Developer ▼

+Add

Topology

Observe

Search

Builds

Helm

Project

ConfigMaps

Secrets

Project: simple-sql-quarkus ▼

## Developer Catalog

Add shared applications, services, event sources, or source-to-image builders to your Project from the developer catalog. Cluster administrators can customize the content made available in the catalog.

All items  
CI/CD  
Databases  
MariaDB  
MySQL  
Postgres  
Languages  
Middleware  
Other

Type ⓘ  
Templates (6)

Databases

Filter by keyword...

A-Z ▼

6 items

🐳 Templates

**MariaDB**  
Provided by Red Hat, Inc.

MariaDB database service, with persistent storage. For more information about using this...

🐳 Templates

**MariaDB (Ephemeral)**  
Provided by Red Hat, Inc.

MariaDB database service, without persistent storage. For more information about using...

MySQL Templates

**MySQL**  
Provided by Red Hat, Inc.

MySQL database service, with persistent storage. For more information about using...

MySQL Templates

**MySQL (Ephemeral)**  
Provided by Red Hat, Inc.

MySQL database service, without persistent storage. For more information about using...

🐘 Templates

**PostgreSQL**  
Provided by Red Hat, Inc.

PostgreSQL database service, with persistent storage. For more information about using...

🐘 Templates

**PostgreSQL (Ephemeral)**  
Provided by Red Hat, Inc.

PostgreSQL database service, without persistent storage. For more information about using...

Select Instantiate Template.

← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/catalog/ns/simple-sql-quarkus?category=databases&selectedId=07d1882c-c658-4a5a-8d... 🔍 ☆ 📱 🧑 : user1

📁 AI 📁 amt 📁 Ansible 📁 ANZ-AML 📁 aws 📁 cassandra 📁 codeready 📁 customers 📁 dataVirt 📁 devconsole 📁 edgeDemo2023 📁 elastic » 📁 All Bookmarks

☰ Red Hat OpenShift

Developer

+Add

Topology

Observe

Search

Builds

Helm

Project

ConfigMaps

Secrets

Project: simple-sql-quarkus

Developer Catalog

Add shared applications, services, event sources, or source-to-image builders to your project

All items

Ci/CD

Databases

MariaDB

MySQL

Postgres

Languages

Middleware

Other

Type ⓘ

Templates (6)

Databases

Filter by keyword...

Templates

MariaDB

Provided by Red Hat, Inc.

MariaDB database service, with persistent storage. For more information about using this...

Templates

PostgreSQL

Provided by Red Hat, Inc.

PostgreSQL database service, with persistent storage. For more information about using...

MySQL (Ephemeral)

Provided by Red Hat, Inc.

Instantiate Template

Provider

Red Hat, Inc.

Created at

🕒 27 Feb 2024, 12:08

Support

[Get support](#)

Documentation

[Refer documentation](#)

Description

MySQL database service, without persistent storage. For more information about using this template, including OpenShift considerations, see <https://github.com/sclorg/mysql-container/blob/master/README.md>.

WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing

Fill in the page as shown. The info has to match that in the application.properties file described in the next section.

← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/catalog/instantiate-template?template=mysql-ephemeral&template-ns=openshift&presele... 🔍 ☆ 📱 ⋮

📁 AI 📁 amt 📁 Ansible 📁 ANZ-AML 📁 aws 📁 cassandra 📁 codeready 📁 customers 📁 dataVirt 📁 devconsole 📁 edgeDemo2023 📁 elastic » 📁 All Bookmarks

☰

Red Hat  
OpenShift

Developer ▾

+Add

Topology

Observe

Search

Builds

Helm

Project

ConfigMaps

Secrets

Namespace \*

simple-sql-quarkus ▾

Memory Limit \*

512Mi

Maximum amount of memory the container can use.

Namespace

openshift

The OpenShift Namespace where the ImageStream resides.

Database Service Name \*

mysql

The name of the OpenShift Service exposed for the database.

MySQL Connection Username

dbuser

Username for MySQL user that will be used for accessing the database.

MySQL Connection Password

password

Password for the MySQL connection user.

MySQL root user Password

password

Password for the MySQL root user.

MySQL Database Name \*

sampledb

Name of the MySQL database accessed.

Version of MySQL Image \*

8.0-el8

Version of MySQL image to be used (8.0-el7, 8.0-el8, or latest).

Create Cancel

MySQL (Ephemeral)

DATABASE: MYSQL

[View documentation](#) [Get support](#)

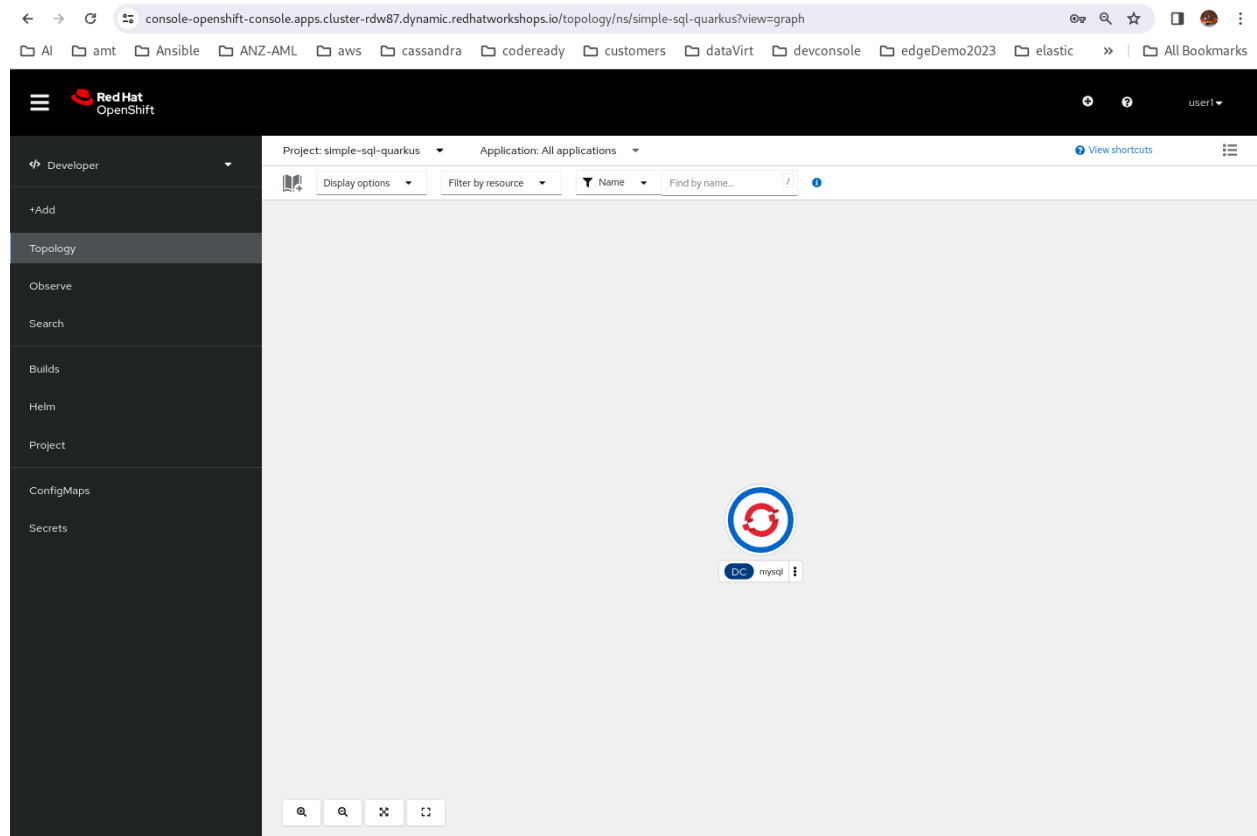
MySQL database service, without persistent storage. For more information about using this template, including OpenShift considerations, see <https://github.com/sclorg/mysql-container/blob/master/8.0/root/usr/share/container-scripts/mysql/README.md>.

WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing

The following resources will be created:

- DeploymentConfig
- Secret
- Service

Click Create and after a while, you will see the database is up (icon with solid deep blue circle)



Select the Administrator tab and the Workloads->Pods

console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/k8s/ns/simple-sql-quarkus/core-v1-Pod

AI amt Ansible ANZ-AML aws cassandra codeready customers dataVirt devconsole edgeDemo2023 elastic All Bookmarks

Red Hat OpenShift

Project: simple-sql-quarkus

### Pods

Create Pod

Filter Name Search by name...

Name	Status	Ready	Restarts	Owner	Memory	CPU	Created
mysql-1-deploy	Completed	0/1	0	mysql-1	-	-	28 Feb 2024, 13:20
mysql-1-r4t5k	Running	1/1	0	mysql-1	-	-	28 Feb 2024, 13:20

https://console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/k8s/ns/simple-sql-quarkus/core-v1-Pod

Click on the running mysql pod.



console-openshift-console.apps.cluster-jw4xl.dynamic.redhatworkshops.io/k8s/ns/simple-sql-quarkus/pods/mysql-1-rgdr6

AI amt Ansible ANZ-AML aws cassandra codeready customers dataVirt devconsole edgeDemo2023 elastic All Bookmarks

Red Hat OpenShift

Administrator

Home

Operators

Workloads

Pods

Deployments

DeploymentConfigs

StatefulSets

Secrets

ConfigMaps

CronJobs

Jobs

DaemonSets

ReplicaSets

ReplicationControllers

HorizontalPodAutoscalers

Project: simple-sql-quarkus

Pods > Pod details

mysql-1-rgdr6 Running

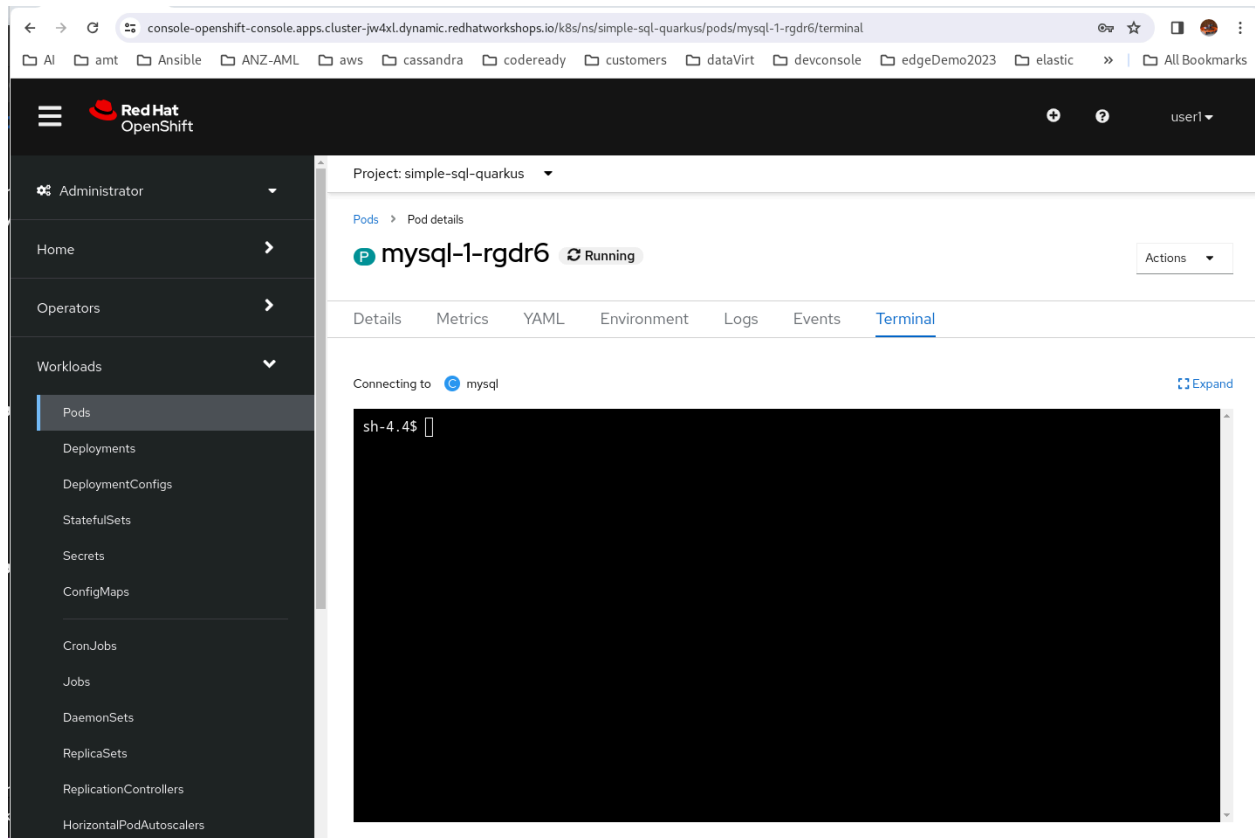
Actions

Details Metrics YAML Environment Logs Events Terminal

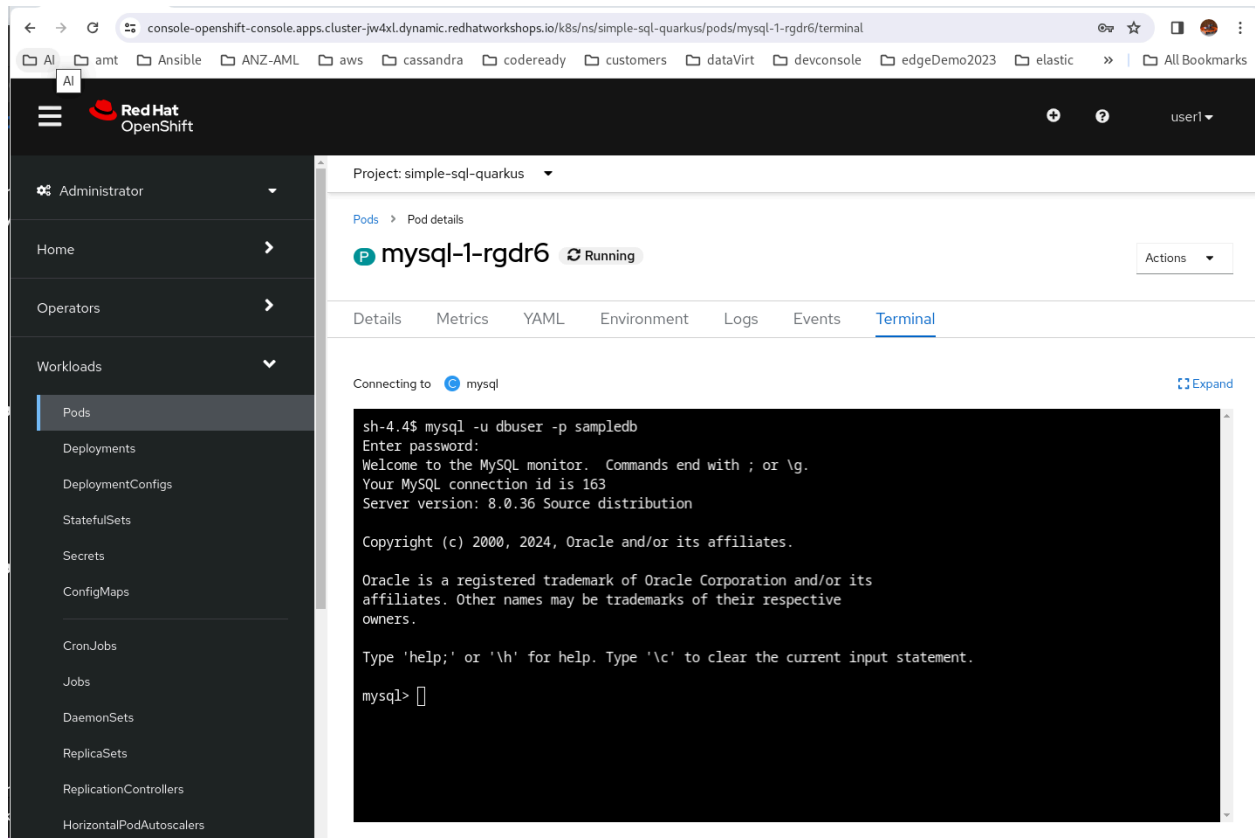
### Pod details

<b>Name</b> mysql-1-rgdr6	<b>Status</b> Running
<b>Namespace</b> simple-sql-quarkus	<b>Restart policy</b> Always restart
<b>Labels</b> deployment=mysql-1 deploymentconfig=mysql name=mysql	<b>Active deadline seconds</b> Not configured
<b>Node selector</b> No selector	<b>Pod IP</b> 10.132.2.18
<b>Tolerations</b> 3 tolerations	<b>Host IP</b> 10.10.10.22
<b>Annotations</b> 7 annotations	<b>Node</b> worker-cluster-jw4xl-3
	<b>Image pull secret</b>

Click on Terminal to open a terminal window to the pod.



Log in to mysql.



Copy and paste the content of schema.sql into the terminal.

The screenshot shows the Red Hat OpenShift console interface. On the left is a sidebar with navigation options: Administrator, Home, Operators, Workloads, and Pods (selected). The main area displays the 'Pod details' for 'mysql-1-rgdr6', which is in a 'Running' state. Below this, there are tabs for Details, Metrics, YAML, Environment, Logs, Events, and Terminal (selected). The terminal window shows a MySQL prompt where several SQL commands have been executed: a DROP TABLE statement, a CREATE TABLE statement for 'customerdemo', and two INSERT statements. The terminal output shows the results of these queries.

```
Project: simple-sql-quarkus

Pods > Pod details

mysql-1-rgdr6 Running Actions

Details Metrics YAML Environment Logs Events Terminal

Connecting to mysql Expand

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> DROP TABLE IF EXISTS customerdemo;
Query OK, 0 rows affected, 1 warning (0.01 sec)

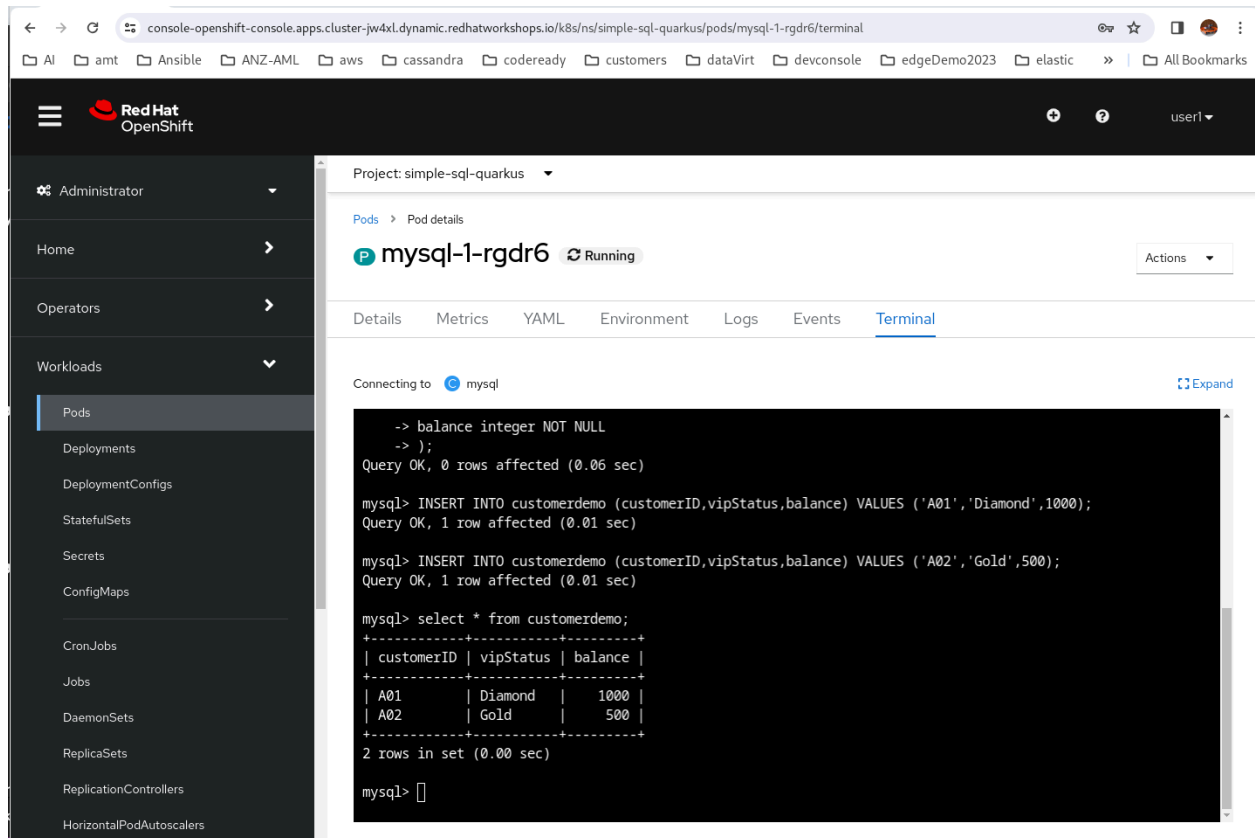
mysql> CREATE TABLE customerdemo (
-> customerID varchar(10) NOT NULL,
-> vipStatus varchar(10) NOT NULL ,
-> balance integer NOT NULL
-> );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES ('A01','Diamond',1000);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO customerdemo (customerID,vipStatus,balance) VALUES ('A02','Gold',500);
Query OK, 1 row affected (0.01 sec)

mysql>
```

Run a simple SQL statement, “select \* from customerdemo;”, to list all entries in the database.



Now you have completed the MySQL database setup on OpenShift

### Deploy your Quarkus App

cd to your project.

Install extensions:

Unset

```

./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-openshift"
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-jdbc-mysql"

```

Add to pom.xml

Inside the <properties> element.

Unset

```

<maven.compiler.source>17</maven.compiler.source>
<maven.compiler.target>17</maven.compiler.target>

```

Add the following entries to the application.properties file:

Unset

```
quarkus.kubernetes-client.trust-certs=true
```

```
# Use this section if you are not using the YySQL database
# If you do that you do not have to do the previous section to
# create and populate a MySQL database in OpenShift.
# #####
#quarkus.datasource.db-kind=h2
#quarkus.datasource.username=sa
#quarkus.datasource.password=
#quarkus.datasource.jdbc.url=jdbc:h2:mem:mydb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;DATABASE_TO_UPPER=false;INIT=RUNSCRIPT FROM 'classpath:schema.sql'
#quarkus.datasource.jdbc.max-size=12

# Use this section if you are using the MySQL database
# Make sure you have created and populated the database as described
# in the previous section.
# database in OpenShift
# #####
quarkus.datasource.db-kind=mysql
quarkus.datasource.username=dbuser
quarkus.datasource.password=password
quarkus.datasource.jdbc.url=jdbc:mysql://mysql:3306/sampledb
quarkus.datasource.jdbc.max-size=12
```

And execute the following command:

Unset

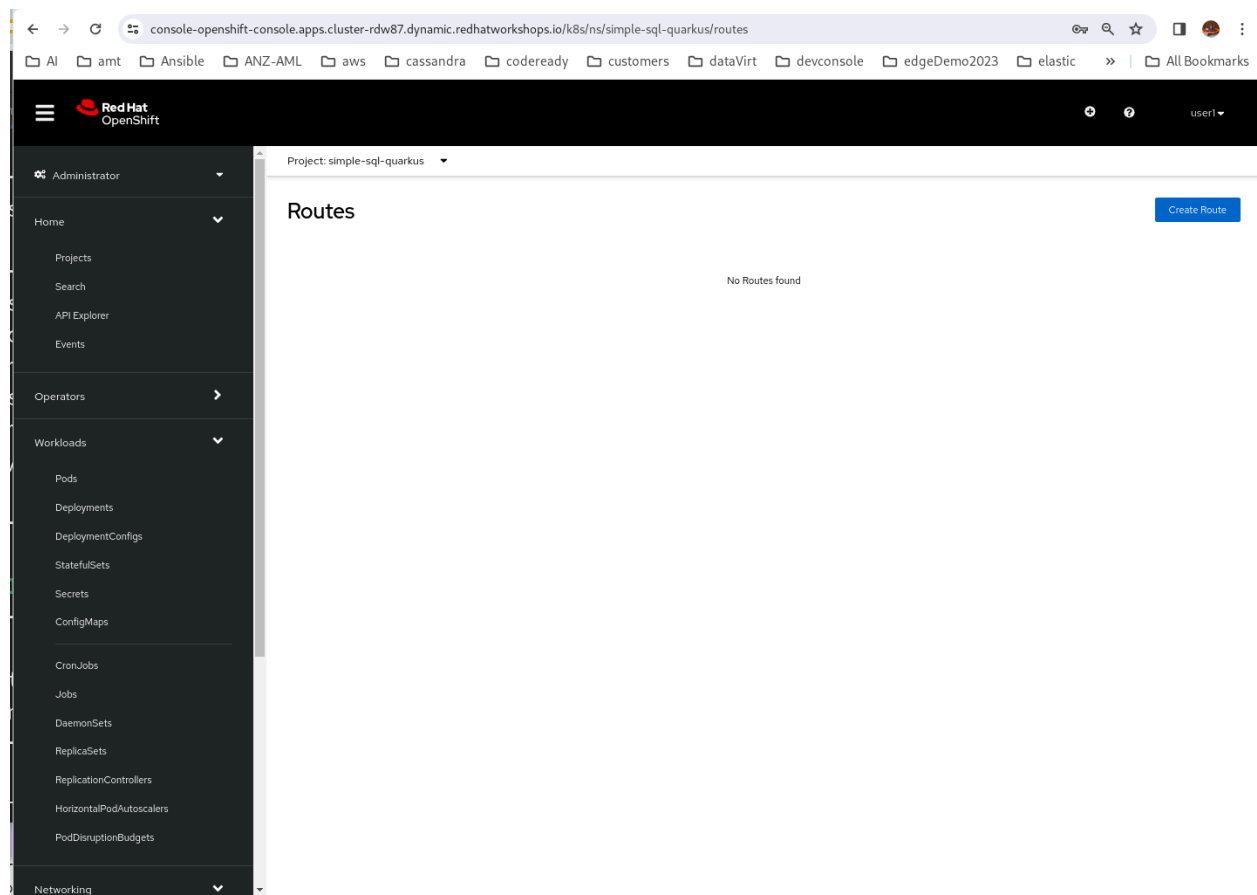
```
oc login -u yourUserName OpenShiftApiServerURL
```

```
oc project simple-sql-quarkus
```

```
./mvnw install -Dquarkus.kubernetes.deploy=true -DskipTests
```

Log in the the OpenShift console:

Create a route for the service which has been created by the mvnw command above.



Click on 'Create Route'.

← → ↻ console-openshift-console.apps.cluster-rdw87.dynamic.redhatworkshops.io/k8s/ns/simple-sql-quarkus/routes/~new 🔍 ☆ 📱 🧑

📁 AI 📁 amt 📁 Ansible 📁 ANZ-AML 📁 aws 📁 cassandra 📁 codeready 📁 customers 📁 dataVirt 📁 devconsole 📁 edgeDemo2023 📁 elastic » 📁 All Bookmarks

☰ Red Hat OpenShift

⚙️ Administrator

Home

Projects

Search

API Explorer

Events

Operators

Workloads

Pods

Deployments

DeploymentConfigs

StatefulSets

Secrets

ConfigMaps

CronJobs

Jobs

DaemonSets

ReplicaSets

ReplicationControllers

HorizontalPodAutoscalers

PodDisruptionBudgets

Networking

Project: simple-sql-quarkus

## Create Route

Routing is a way to make your application publicly visible.

Configure via: ☒ Form view ☐ YAML view

**Name \***

A unique name for the Route within the project.

**Hostname**

Public hostname for the Route. If not specified, a hostname is generated.

**Path**

Path that the router watches to route traffic to the service.

**Service \***

simple-sql-quarkus

Service to route to.

**Target port \***

80 → 8080 (TCP)

Target port for traffic.

**Security**

☒ Secure Route

Routes can be secured using several TLS termination types for serving certificates.

**TLS termination \***

Edge

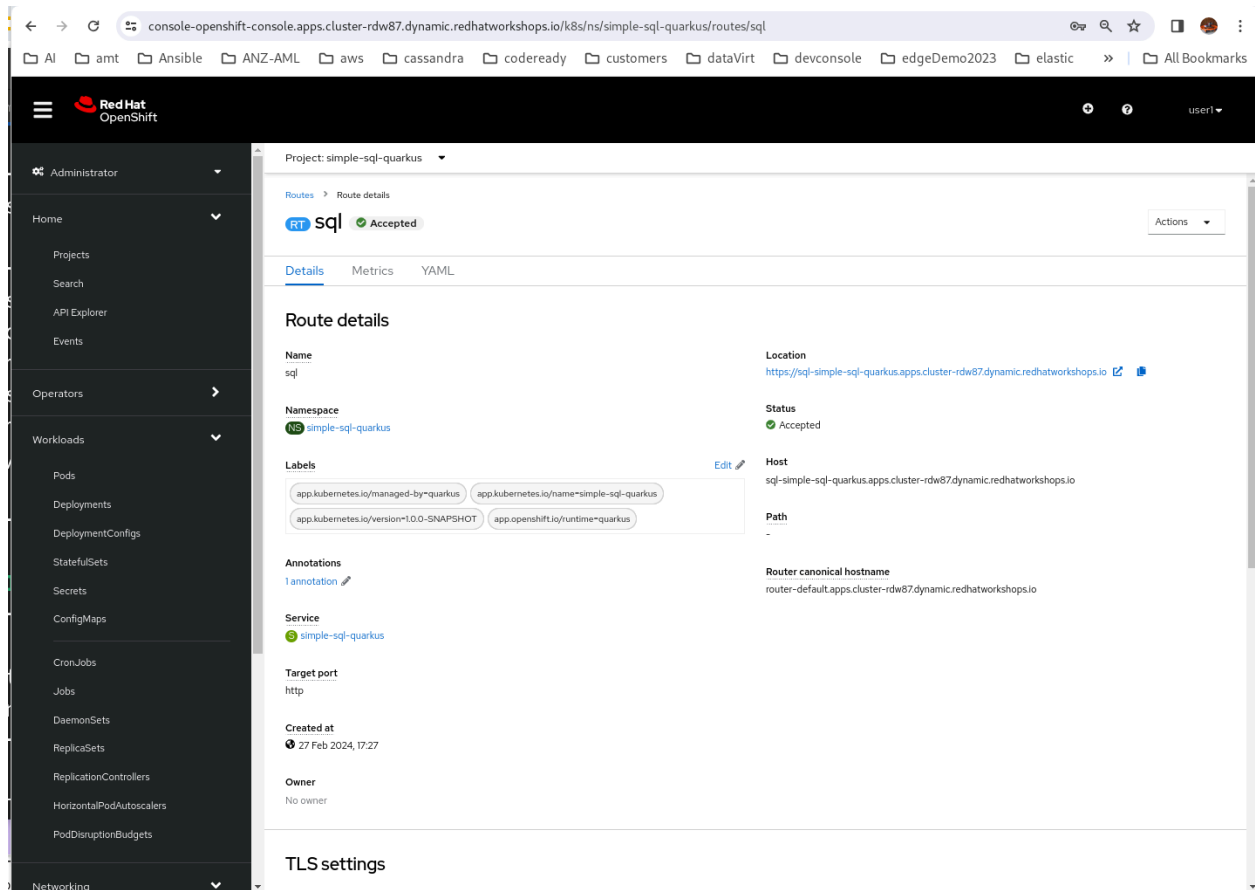
**Insecure traffic**

Select insecure traffic type

Create Cancel

Enter the name and select from the drop down boxes as shown above and click 'Create'





The route is shown at the 'Location'. Use it to interact with your application.

Testing:

Note: Your URL will be different running on a different OpenShift environment.

Unset

`curl`

`https://rest-simple-sql-quarkus.apps.cluster-rdw87.dynamic.redhatworkshops.io/fruits 2>/dev/null | jq`

`curl -X POST`

`https://rest-simple-sql-quarkus.apps.cluster-rdw87.dynamic.redhatworkshops.io/fruits \`

`-H 'Content-Type: application/json' \`

`-d '{"name": "Orange", "description": "Popular fruit"}' 2>/dev/null | jq`

## 1.3 An Application Using EIPs

This application demonstrates the use of Content-based Routing, WireTap EIPs using additional Camel components(jsonpath, qson, SEDA) and a Processor. This app is an enhancement to the simple-rest-project.

You POST a fruit with name and description to the Rest API. If the description is one of {"Winter fruit", "Tropical fruit", "Super fruit"}, it takes the path to log the fruit preceded by the description.

If the description is unknown, the route uses a wiretap (an EIP) to send the request asynchronously to the "seda:correction" endpoint. So far, all routing eg, direct:submit, was done synchronously.

The "seda:correction" endpoint uses a LookupProcessor, with some predefined fruits, to lookup the fruit passed to it should contain what description. If the lookup is successful, it will update the fruit description and send it back to the "direct:submit" endpoint for re-processing otherwise, it logs the message: " SEDA ignoring fruit with unknown description: ...."

You can generate another project at <https://code.quarkus.redhat.com/>, or you can make a copy of the previous project 'simple-rest-quarkus' and start from there.

Change the directory name of your copy to "simple-cbr-quarkus"

Open that folder using your VSCode IDE

Open the pom.xml file

Change the line from:

```
Unset
<artifactId>simple-rest-quarkus</artifactId>
```

To

```
Unset
<artifactId>simple-cbr-quarkus</artifactId>
```

Open the simple-cbr-quarkus/src/main/java/com/redhat/Routes.java and replace its content with the following:

Java

```
package com.redhat;

import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.Set;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.rest.RestBindingMode;
import org.apache.camel.component.jackson.JsonDataFormat;
import org.apache.camel.model.dataformat.JsonLibrary;

public class Routes extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/submitFruit")

            .post()
            .type(Fruit.class)
            .to("direct:submit");

        from("direct:submit")
            .marshal().json(JsonLibrary.Gson)
            .choice()
                .when().jsonpath("$.description == 'Winter fruit'")
                    .log("Winter fruit: ${body}")
                .when().jsonpath("$.description == 'Tropical fruit'")
                    .log("Tropical fruit: ${body}")
                .when().jsonpath("$.description == 'Super fruit'")
                    .log("Super fruit: ${body}")
                .otherwise()
                    .wireTap("seda:correction")
                    .log("Unknown fruit: ${body}");

        from("seda:correction")
            .choice()
                .when().jsonpath("$.description != 'Unknown fruit'")
                    .unmarshal().json(JsonLibrary.Jackson, Fruit.class)
                    .process("lookupProcessor")
                    .log("SEDA resubmitting modified unknown fruit: ${body}")
                    .to("direct:submit")
                .otherwise()
```

```

        .log("SEDA ignoring fruit with unknown description: ${body}");
    }
}

```

Create `simple-cbr-quarkus/src/main/java/com/redhat/LookupProcessor.java` and paste in the content shown below.

```

Java
package com.redhat;

import java.util.Map;
import java.util.HashMap;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.Message;

@ApplicationScoped
@Named
public class LookupProcessor implements Processor {

    private final String WINTER_FRUIT = "Winter fruit";
    private final String TROPICAL_FRUIT = "Tropical fruit";
    private final String SUPER_FRUIT = "Super fruit";
    private final String UNKNOWN_FRUIT = "Unknown fruit";

    private final Map<String, String> descMap = new HashMap<>();

    public LookupProcessor() {

        descMap.put("Apple", WINTER_FRUIT);
        descMap.put("Kiwi", WINTER_FRUIT);
        descMap.put("Pear", WINTER_FRUIT);
        descMap.put("Orange", WINTER_FRUIT);
        descMap.put("Pineapple", TROPICAL_FRUIT);
        descMap.put("Durian", TROPICAL_FRUIT);
        descMap.put("Lychee", TROPICAL_FRUIT);
        descMap.put("Mango", TROPICAL_FRUIT);
        descMap.put("Avocado", SUPER_FRUIT);
        descMap.put("Plum", SUPER_FRUIT);
    }
}

```

```

        descMap.put("Strawberry", SUPER_FRUIT);
        descMap.put("Peaches", SUPER_FRUIT);
    }

    public boolean lookup(String fruit) {
        if (descMap.containsKey(fruit)) {
            return true;
        }
        return false;
    }

    public String lookupDescription(String fruit) {
        if (lookup(fruit)) {
            return descMap.get(fruit);
        }
        return UNKNOWN_FRUIT;
    }

    @Override
    public void process(Exchange exchange) {
        Message msg = exchange.getMessage();
        Fruit fruit = msg.getBody(Fruit.class);
        fruit.setDescription(lookupDescription(fruit.getName()));
        System.out.println("message: " + fruit.getName() + " is a " +
            fruit.getDescription() );
    }
}

```

Add the following Quarkus extensions:

```

Unset
./mvnw quarkus:add-extension -Dextensions="jsonpath"
./mvnw quarkus:add-extension -Dextensions="gson"
./mvnw quarkus:add-extension -Dextensions="camel-quarkus-seda"

```

Run the app in DEV mode:

Unset

```
./mvnw clean compile quarkus:dev
```

The app knows about 3 fruit descriptions: 'Winter fruit', 'Tropical fruit' and 'Super fruit' only.

Interact with the app:

For fruit description for the 3 known one, the message will be logged prefixed with the description in the console:

Unset

```
curl -X POST http://localhost:8080/submitFruit \
-H 'Content-Type: application/json' \
-d '{"name": "Orange", "description": "Winter fruit"}'
```

```
2024-02-27 10:51:19,780 INFO [route1] (vert.x-worker-thread-1) Winter fruit:
{"name":"Orange","description":"Winter fruit"}
```

For unknown or missing description, the message will be route asynchronously to "seda:correction". It uses a Camel processor to look up the proper description of known fruits. If the lookup is successful, it routes it back to 'direct:submit' for processing again.

Unset

```
curl -X POST http://localhost:8080/submitFruit \
-H 'Content-Type: application/json' \
-d '{"name": "Orange", "description": "Popular fruit"}'
```

```
2024-02-27 10:59:01,848 INFO [route1] (vert.x-worker-thread-1) Unknown fruit:
{"name":"Orange","description":"Popular fruit"}
```

```
message: Orange is a Winter fruit <-----gets wiretapped and sent to
seda:correction. The LookupProcessor looks up orange's description and outputs
it here.
```

```
2024-02-27 10:59:01,862 INFO [route2] (Camel (camel-1) thread #1 -
seda://correction) SEDA resubmitting modified unknown fruit:
com.redhat.Fruit@8d43126d <----- the message was updated with the proper
description provided by the LookupProcessor and routed back to direct:submit
2024-02-27 10:59:01,863 INFO [route1] (Camel (camel-1) thread #1 -
seda://correction) Winter fruit: {"name":"Orange","description":"Winter fruit"}
```

```
<----- The message is logged normally as the fruit contains a proper
description
```

Same as above, but this time the LookupProcessor fails to look up the fruit's description:

```
Unset
curl -X POST http://localhost:8080/submitFruit \
  -H 'Content-Type: application/json' \
  -d '{"name": "Jackfood", "description": "Unusual fruit"}'

2024-02-27 10:59:01,862 INFO [route2] (Camel (camel-1) thread #1 -
seda://correction) SEDA resubmitting modified unknown fruit:
com.redhat.Fruit@8d43126d
2024-02-27 10:59:01,863 INFO [route1] (Camel (camel-1) thread #1 -
seda://correction) Winter fruit: {"name":"Orange","description":"Winter fruit"}
2024-02-27 11:09:25,987 INFO [route1] (vert.x-worker-thread-1) Unknown fruit:
{"name":"Jackfood","description":"Unusual fruit"}

message: Jackfood is a Unknown fruit<-----gets wiretapped and sent to
seda:correction. The LookupProcessor fails to look up Jackfood's description
and outputs it here.
2024-02-27 11:09:25,988 INFO [route2] (Camel (camel-1) thread #1 -
seda://correction) SEDA ignoring fruit with unknown description:
com.redhat.Fruit@c15710fc<-----no more rerouting
```

## **Deploying to OpenShift**

### ***Deploy the Application***

Install extension:

```
Unset
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-openshift"
```

Add to pom.xml

Unset

```
<maven.compiler.source>17</maven.compiler.source>  
<maven.compiler.target>17</maven.compiler.target>
```

To avoid certificate problems, add the following entry to the application.properties file:

Unset

```
quarkus.kubernetes-client.trust-certs=true
```

And execute the following command:

Unset

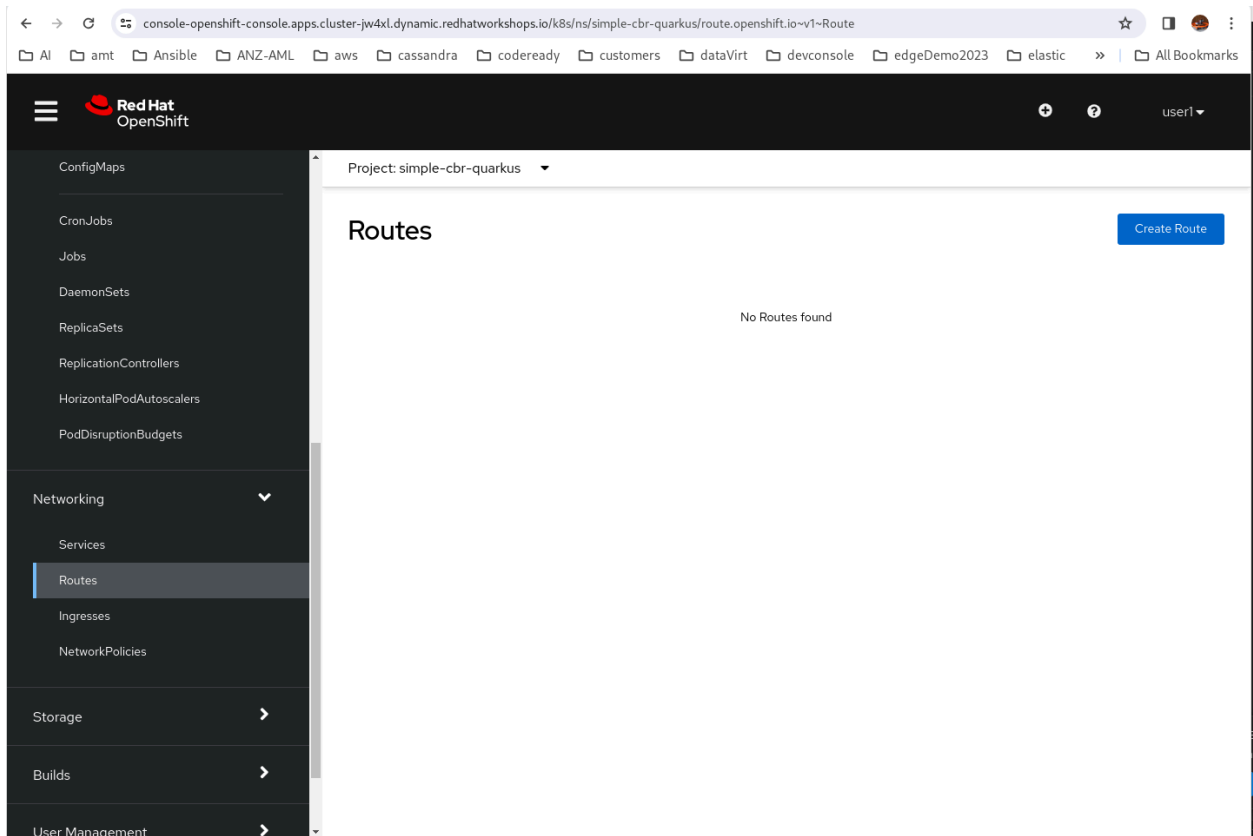
```
oc login -u yourUserName OpenShiftApiServerURL  
  
oc new-project simple-cbr-quarkus  
  
./mvnw clean install -Dquarkus.kubernetes.deploy=true -DskipTests
```

Log in the the OpenShift console:

Select the Administrator tab and then Networking->Routes.

Create a route for the service which has been created by the mvnw command above.





Click on 'Create Route'.

console-openshift-console.apps.cluster-jw4xl.dynamic.redhatworkshops.io/k8s/ns/simple-cbr-quarkus/routes/~new/form

AI amt Ansible ANZ-AML aws cassandra codeready customers dataVirt devconsole edgeDemo2023 elastic All Bookmarks

Red Hat OpenShift user1

Project: simple-cbr-quarkus

Configure via: ☒ Form view ☐ YAML view

**Name \***  
cbr  
A unique name for the Route within the project.

**Hostname**  
www.example.com  
Public hostname for the Route. If not specified, a hostname is generated.

**Path**  
/  
Path that the router watches to route traffic to the service.

**Service \***  
simple-cbr-quarkus  
Service to route to.

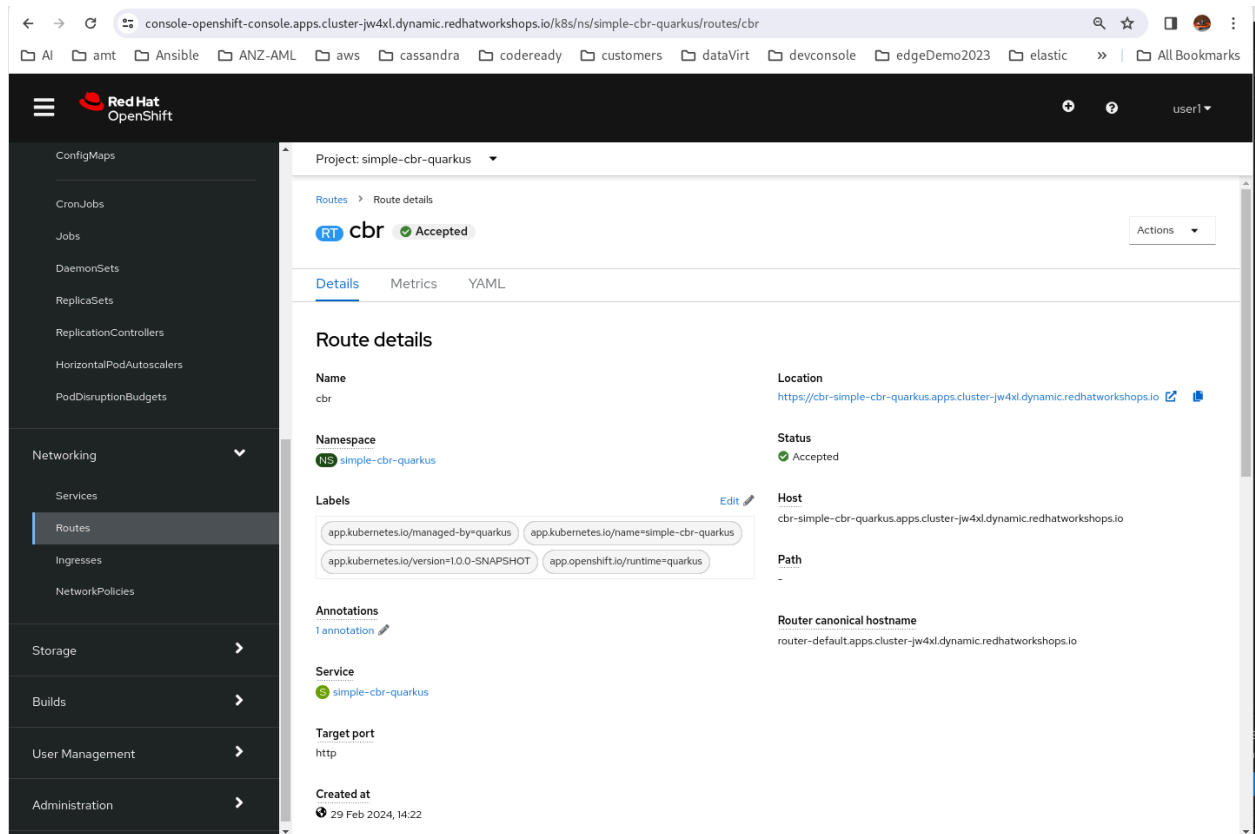
**Target port \***  
80 → 8080 (TCP)  
Target port for traffic.

**Security**  
☒ Secure Route  
Routes can be secured using several TLS termination types for serving certificates.

**TLS termination \***  
Edge

Create Cancel

Enter the name and select from the drop down boxes as shown above and click 'Create'



The route is shown at the 'Location'. Use it to interact with your application.

### Testing:

Similar to what you did when testing locally. Only the URL is different. Your URL will be different when running in a different OpenShift environment.

Unset

```
curl -X POST
https://cbr-simple-cbr-quarkus.apps.cluster-jw4xl.dynamic.redhatworkshops.io/submitFruit \
-H 'Content-Type: application/json' \
-d '{"name": "Orange", "description": "Popular fruit"}' 2>/dev/null | jq
```

and

```
curl -X POST
https://cbr-simple-cbr-quarkus.apps.cluster-jw4xl.dynamic.redhatworkshops.io/submitFruit \
-H 'Content-Type: application/json' \
-d '{"name": "Orange", "description": "Winter fruit"}' 2>/dev/null | jq
```

The results are in the pod's log file which you can access as follows:

Login to the OpenShift Console.

Change to the Administrator tab. Click Workloads->Pods and select the simple-cbr-quarkus project near the top.

The screenshot shows the OpenShift Console interface. The left sidebar contains navigation options: Administrator, Home, Projects, Search, API Explorer, Events, Operators, Workloads, and a list of workload types (Deployments, DeploymentConfigs, StatefulSets, Secrets, ConfigMaps, CronJobs, Jobs, DaemonSets, ReplicaSets). The 'Workloads' section is expanded, and 'Pods' is selected. The main panel displays the 'Pods' page for the 'simple-cbr-quarkus' project. It includes a 'Create Pod' button and a table of pods.

Name	Status	Ready	Restarts	Owner	Memo...	CPU	Creat...
simple-cbr-quarkus-1-2dwk6	Running	1/1	0	simple-cbr-quarkus-1	183.6 MiB	0.001 cores	29 Feb 2024, 14:20
simple-cbr-quarkus-1-build	Completed	0/1	0	simple-cbr-quarkus-1	-	-	29 Feb 2024, 14:19
simple-cbr-quarkus-1-deploy	Completed	0/1	0	simple-cbr-quarkus-1	-	-	29 Feb 2024, 14:20

Click on the running simple-cbr-quarkus-... pod.

console-openshift-console.apps.cluster-jw4xl.dynamic.redhatworkshops.io/k8s/ns/simple-cbr-quarkus/pods/simple-cbr-quarkus-1-2dwk6

AI amt Ansible ANZ-AML aws cassandra codeready customers dataVirt devconsole edgeDemo2023 elastic All Bookmarks

Red Hat OpenShift user1

Administrator

Home

Projects

Search

API Explorer

Events

Operators

Workloads

Pods

Deployments

DeploymentConfigs

StatefulSets

Secrets

ConfigMaps

CronJobs

Jobs

DaemonSets

ReplicaSets

Project: simple-cbr-quarkus

Pods Pod details

simple-cbr-quarkus-1-2dwk6 Running Actions

Details Metrics YAML Environment Logs Events Terminal

### Pod details

<b>Name</b> simple-cbr-quarkus-1-2dwk6	<b>Status</b> Running
<b>Namespace</b> simple-cbr-quarkus	<b>Restart policy</b> Always restart
<b>Labels</b> <a href="#">app.kubernetes.io/managed-by=quarkus</a> <a href="#">app.kubernetes.io/name=simple-cbr-quarkus</a> <a href="#">app.kubernetes.io/version=1.0.0-SNAPSHOT</a> <a href="#">app.openshift.io/runtime=quarkus</a> <a href="#">deployment=simple-cbr-quarkus-1</a> <a href="#">deploymentconfig=simple-cbr-quarkus</a>	<b>Active deadline seconds</b> Not configured
<b>Node selector</b> No selector	<b>Pod IP</b> 10.132.2.27
<b>Tolerations</b> <a href="#">2 tolerations</a>	<b>Host IP</b> 10.10.10.22
<b>Annotations</b> <a href="#">9 annotations</a>	<b>Node</b> worker-cluster-jw4xl-3
<b>Created at</b>	<b>Image pull secret</b> default-dockercfg-g6w9l
	<b>PodDisruptionBudget</b> No PodDisruptionBudget

Click on Logs and you will see the log.



## 2.0 Red Hat build of Apache Camel for Spring Boot

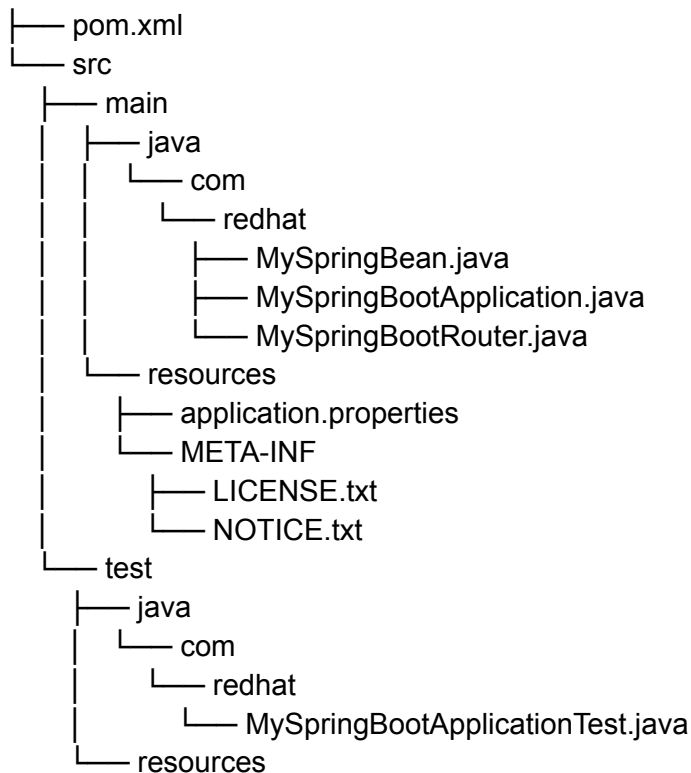
### 2.1 Your First Spring Boot Apache Camel Application

Generating a Camel for Spring Boot application skeleton using Maven.

From a terminal, run the command:

```
mvn archetype:generate \
-DarchetypeGroupId=org.apache.camel.archetypes \
-DarchetypeArtifactId=camel-archetype-spring-boot \
-DarchetypeVersion=4.0.0.redhat-00045 \
-DgroupId=com.redhat \
-DartifactId=simple-eip-springboot \
-Dversion=1.0-SNAPSHOT \
-DinteractiveMode=false
```

The generated projects has the following directory structure:



From VSCode, open the simple-eip folder and examine the source code ie, java and resources.

From a terminal window, execute the command:

Unset

```
mvn clean spring-boot:run
```

The above command may take some time to download the dependencies.

The result look like:

Unset

...

```
2024-02-22T07:26:25.344+11:00 INFO 47089 --- [      main]
o.a.c.impl.engine.AbstractCamelContext : Routes startup (started:1)
2024-02-22T07:26:25.344+11:00 INFO 47089 --- [      main]
o.a.c.impl.engine.AbstractCamelContext : Started hello (timer://hello)
2024-02-22T07:26:25.344+11:00 INFO 47089 --- [      main]
o.a.c.impl.engine.AbstractCamelContext : Apache Camel 4.0.0.redhat-00036
(MyCamel) started in 15ms (build:0ms init:0ms start:15ms)
2024-02-22T07:26:25.348+11:00 INFO 47089 --- [      main]
com.redhat.MySpringBootApplication : Started MySpringBootApplication in 2.976
seconds (process running for 3.399)
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

It should be clear by now that the application has nothing to do with EIP, yet. We just want to see that the generated simple Apache Camel application is working.

We are now converting the application to do the following:

- This app is very similar to the Quarkus 1.3 app. The difference is that it uses a timer to generate a random Fruit object and passes it through the routes. The output will be similar to that of Quarkus app 1.3. This is to show you how similar the source code are when using Camel in Quarkus and Spring Boot..
- Deploy your application to OpenShift



## 2.2 An Application Using EIPs

This application demonstrates the use of Content-based Routing, WireTap EIPs using additional Camel components(jsonpath, qson, SEDA, Bean) and a Processor.

It is very similar to the simple-cbr-quarkus project. The difference is that it uses a timer and a Bean (MySpringBean) to generate a random fruit and submit it to the “direct:submit” endpoint every 10 seconds instead of relying on a user to call a Rest API. Some of the generated fruit will have a description and/or name not known to the MySpringBean ending up exercising all the defined routes eventually.

We are going to change the ‘simple-eip’ project.

If you have the feeling that you have seen the code before, that is correct. You have seen most of the code in projects described in this document.

Open that folder using your VSCode IDE

Add the following dependencies to the pom.xml file:

Note that unlike Quarkus, you have to manually add the dependencies to the pom.xml file.

Unset

...

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jsonpath-starter</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-seda-starter</artifactId>
</dependency>
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-jackson-starter</artifactId>
  </dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-gson</artifactId>
  <version>4.0.0</version>
</dependency>
```

```
<dependency>
  <groupId>jakarta.enterprise</groupId>
  <artifactId>jakarta.enterprise.cdi-api</artifactId>
  <version>3.0.0</version>
</dependency>
...
```

Add file `simple-eip-springboot/src/main/java/com/redhat/Fruit.java`  
Paste in the content:

```
Java
package com.redhat;

import java.util.Objects;

public class Fruit {
    private String name;
    private String description;

    public Fruit() {
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

```

@Override
public boolean equals(Object obj) {
    if (!(obj instanceof Fruit)) {
        return false;
    }

    Fruit other = (Fruit) obj;

    return Objects.equals(other.name, this.name);
}

@Override
public int hashCode() {
    return Objects.hash(this.name);
}
}

```

Add file `simple-eip-springboot/src/main/java/com/redhat/LookupProcessor.java`  
 Paste in the content:

```

Java
package com.redhat;

import java.util.Map;
import java.util.HashMap;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.Message;

@ApplicationScoped
@Named
public class LookupProcessor implements Processor {

    private final String WINTER_FRUIT = "Winter fruit";
    private final String TROPICAL_FRUIT = "Tropical fruit";
    private final String SUPER_FRUIT = "Super fruit";
    private final String UNKNOWN_FRUIT = "Unknown fruit";
}

```

```

private final Map<String, String> descMap = new HashMap<>();

public LookupProcessor() {

    descMap.put("Apple", WINTER_FRUIT);
    descMap.put("Kiwi", WINTER_FRUIT);
    descMap.put("Pear", WINTER_FRUIT);
    descMap.put("Orange", WINTER_FRUIT);
    descMap.put("Pineapple", TROPICAL_FRUIT);
    descMap.put("Durian", TROPICAL_FRUIT);
    descMap.put("Lychee", TROPICAL_FRUIT);
    descMap.put("Mango", TROPICAL_FRUIT);
    descMap.put("Avocado", SUPER_FRUIT);
    descMap.put("Plum", SUPER_FRUIT);
    descMap.put("Strawberry", SUPER_FRUIT);
    descMap.put("Peaches", SUPER_FRUIT);

}

public boolean lookup(String fruit) {
    if (descMap.containsKey(fruit)) {
        return true;
    }
    return false;
}

public String lookupDescription(String fruit) {
    if (lookup(fruit)) {
        return descMap.get(fruit);
    }
    return UNKNOWN_FRUIT;
}

@Override
public void process(Exchange exchange) {
    Message msg = exchange.getMessage();
    Fruit fruit = msg.getBody(Fruit.class);
    fruit.setDescription(lookupDescription(fruit.getName()));
    System.out.println("message: " + fruit.getName() + " is a " +
fruit.getDescription() );
}
}

```

Open file simple-eip-springboot/src/main/java/com/redhat/MySpringBean.java  
Replace the content with:

```
Java
package com.redhat;

import org.springframework.stereotype.Component;

@Component("myBean")
public class MySpringBean {

    static final String[] fruitName = new String[] {
        "Apple", "Banana",
        "Kiwi", "Grape",
        "Pear", "Strawberry",
        "Orange", "Melon",
        "Pineapple", "Blueberry",
        "Durian", "Mandarin",
        "Lychee", "Peach",
        "Mango", "Raspberry",
        "Avocado", "Blackberry",
        "Plum", "Lemon",
        "Strawberry", "Line",
        "Peaches", "Grapefruit"
    };

    static final String[] fruitDescription = new String[] {
        "Winter fruit", "Tropical fruit",
        "Super fruit", "Popular fruit",
        "Summer fruit", "Autumn fruit",
        "Spring fruit", "Seasonal fruit"
    };

    public Fruit getFruit() {
        int nameIndex = (int) (Math.random() * fruitName.length);
        int descIndex = (int) (Math.random() * fruitDescription.length);
        return new Fruit(fruitName[nameIndex], fruitDescription[descIndex]);
    }
}
```

Open file simple-eip-springboot/src/main/java/com/redhat/MySpringBootRouter.java  
Replace the content with:

Java

```
package com.redhat;

import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;
import org.apache.camel.component.jackson.JsonDataFormat;
import org.apache.camel.model.dataformat.JsonLibrary;

@Component
public class MySpringBootRouter extends RouteBuilder {

    @Override
    public void configure() {
        from("timer:generate?period={{timer.period}}")
            .transform().method("myBean", "getFruit")
            .to("direct:submit");

        from("direct:submit")
            .marshal().json(JsonLibrary.Gson)
            .choice()
                .when().jsonpath("${?(@.description == 'Winter fruit']}")
                    .log("Winter fruit: ${body}")
                .when().jsonpath("${?(@.description == 'Tropical fruit')}")
                    .log("Tropical fruit: ${body}")
                .when().jsonpath("${?(@.description == 'Super fruit')}")
                    .log("Super fruit: ${body}")
                .otherwise()
                    .wireTap("seda:correction")
                    .log("Unknown fruit: ${body}");

        from("seda:correction")
            .choice()
                .when().jsonpath("${?(@.description != 'Unknown fruit')}")
                    .unmarshal().json(JsonLibrary.Jackson, Fruit.class)
                    .process("lookupProcessor")
                    .log("SEDA resubmitting modified unknown fruit: ${body}")
                    .to("direct:submit")
                .otherwise()
                    .log("SEDA ignoring fruit with unknown description: ${body}");
    }
}
```

Open the file `simple-eipl-quarkus/src/main/resources/application.properties`  
Replace the entry:

```
Unset
# how often to trigger the timer
timer.period = 2000
```

With

```
Unset
# how often to trigger the timer
timer.period = 10000
```

And run the following command in a terminal:

```
Unset
mvn clean spring-boot:run
```

You will see in the console a randomly generated fruit with name and description go through the routes every 10 seconds. You can interpret the log entries as described for the output of Quarkus app 1.3.

### **Deploying to OpenShift**

cd to your project.

And execute the following command:

```
Unset
oc login -u yourUserName OpenShiftApiServerURL

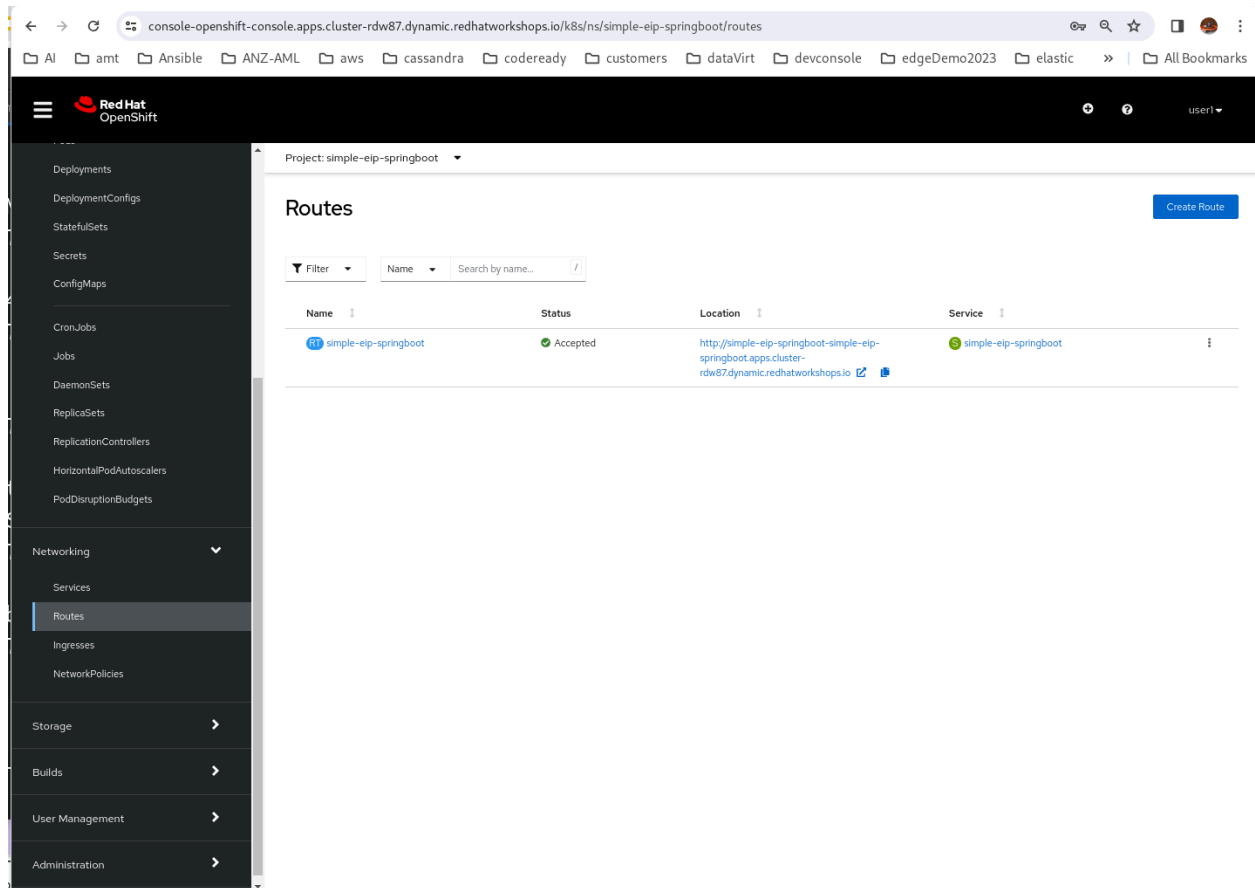
oc new-project simple-eip-springboot

mvn -Popenshift oc:deploy -DskipTests
```

Log in the the OpenShift console:

Select the Administrator tap and then Networking->Routes

Select the project simple-eip-springboot



The route has already been created by the deployment but we do not need the route for this app as the result can be observed in the logs.

### Testing:

Here is another way of looking at the log from the command line instead of from the OpenShift Console.

Assuming you are still in the OpenShift simple-eip-springboot project, find out the pod name of your app.

```
Unset
oc get pod

or

oc get pod -n simple-eip-springboot
```

And you should see something like this:



Unset

NAME	READY	STATUS	RESTARTS	AGE
simple-eip-springboot-1-deploy	0/1	Completed	0	10m
simple-eip-springboot-1-tnx82	1/1	Running	0	10m
simple-eip-springboot-s2i-1-build	0/1	Completed	0	11m

Identify the running pod. In this case, it is simple-eip-springboot-1-tnx82. In your case it will have a different name.

Access the log as follows from a terminal:

Unset

```
oc logs -f simple-eip-springboot-1-tnx82
```

And you will see the similar log entries as you saw on the Spring Boot console when you run the app locally using 'mvn clean spring-boot:run'.

The -f option stands for follow meaning it will continually update the screen with the latest log entries.

Make sure you understand what you are seeing as described previously.

And that's it folks!