

Cloud Native Training Lab Environment Setup V2 Day2

System Requirements:

OS

1. Windows 10 or above
2. Mac OS 10.15 or above

Hardware requirement

1. CPU i5 or above
2. RAM 8GB or above
3. Hard Disk 80GB free or above

Software List

1. JDK 17 or above
2. Apache Maven 3.8.3 or above
3. IDE (Visual Studio Code)
4. [Pre-Lab] Quarkus Sample Project Hello World
5. (Optional) VMware Player or Virtual Box (for Kubernetes Installation) for Ubuntu 18.04.5 (x2 VMs)
6. (Optional) Script for install Kubernetes Cluster(1 Master Node, 1 Workder Node) on Ubuntu 18.04.5
ubuntu-18.04.5-live-server-amd64

Ubuntu 18.04.5 ISO Download

<https://old-releases.ubuntu.com/releases/18.04.5/ubuntu-18.04.5-live-server-amd64.iso>

Lab Source Git

Lab0-3

<https://github.com/lftsang/cloudnative-quarkus-lab>

Lab4

<https://github.com/lftsang/cloudnative-quarkus-jwt-lab>

Prepared by :

Felix Tsang

felix@linux.com

Lab List

Table of Contents

Environment Setup..... **2**

1) JDK Installation	2
2) Maven Installation	3
3) IDE (Visual Studio Code)	3
Lab0 Quarkus Sample Project Hello World.....	5
LAB1 URL Parameter	8
LAB2 Response with JSON Obj.....	9
LAB3 HTTP Post Consume JSON Obj.....	14
LAB4 JWT	15
Challenge yourself.....	20
Lab 5 Install Kubernetes Cluster (1 Master , 1 Worker)	20
Lab 6 Create Quay.io registry Account	21
Lab 7 Install podman/docker client and build the container image with quarkus extension	24
Lab 8 Deploy the app on kubernetes cluster	26
Appendix.....	27

Environment Setup

1) JDK Installation

For Mac (Detailed Procedures ref “Appendix I JDK Installation”)

Install JDK 17 or above by dmg installer

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

java -version

java version "17.0.1" 2021-10-19 LTS

Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)

Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

For Windows (Detailed Procedures ref “Appendix I JDK Installation”)

Install JDK 17

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Set Environment Variable

java -version or above

java version "17.0.1" 2021-10-19 LTS

Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)

Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

2) Maven Installation

Install Maven (Detailed Procedures ref “Appendix II Maven Installation”)

The installation process of Apache Maven is quite simple, we just need to extract the Maven’s zip file and set up maven environment variables.

For MAC , Install brew

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"  
brew install maven  
mvn -v  
Apache Maven 3.8.3 (ff8e977a158738155dc465c6a97ffaf31982d739)  
Maven home: /usr/local/Cellar/maven/3.8.3/libexec  
Java version: 17.0.1, vendor: Oracle Corporation, runtime:  
/Library/Java/JavaVirtualMachines/jdk-17.0.1.jdk/Contents/Home  
Default locale: en_HK, platform encoding: UTF-8  
OS name: "mac os x", version: "10.15.7", arch: "x86_64", family: "mac"
```

```
admins-iMac:~ admin$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"  
==> Checking for 'sudo' access (which may request your password)...  
Password:  
==> This script will install:  
/usr/local/bin/brew  
/usr/local/share/doc/homebrew  
/usr/local/share/man/man1/brew.1  
/usr/local/share/zsh/site-functions/_brew  
/usr/local/etc/bash_completion.d/brew  
/usr/local/Homebrew  
==> The Xcode Command Line Tools will be installed.  
Press RETURN to continue or any other key to abort:  
==> /usr/bin/sudo /usr/sbin/chown -R admin:admin /usr/local/Homebrew  
==> Searching online for the Command Line Tools  
==> /usr/bin/sudo /usr/bin/touch /tmp/.com.apple.dt.CommandLineTools.installondemand.in-progress  
==> Installing Command Line Tools for Xcode-13.2  
==> /usr/bin/sudo /usr/sbin/softwareupdate -i Command\ Line\ Tools\ for\ Xcode-13.2  
Software Update Tool  
Finding available software  
Downloading Command Line Tools for Xcode■
```

For Windows

Maven needs JDK to be installed before configuration.

1.Finish JDK setup and set up JAVA_HOME **Environment Variable in the previous steps**

2.Download Maven Zip and unzip to your working folder

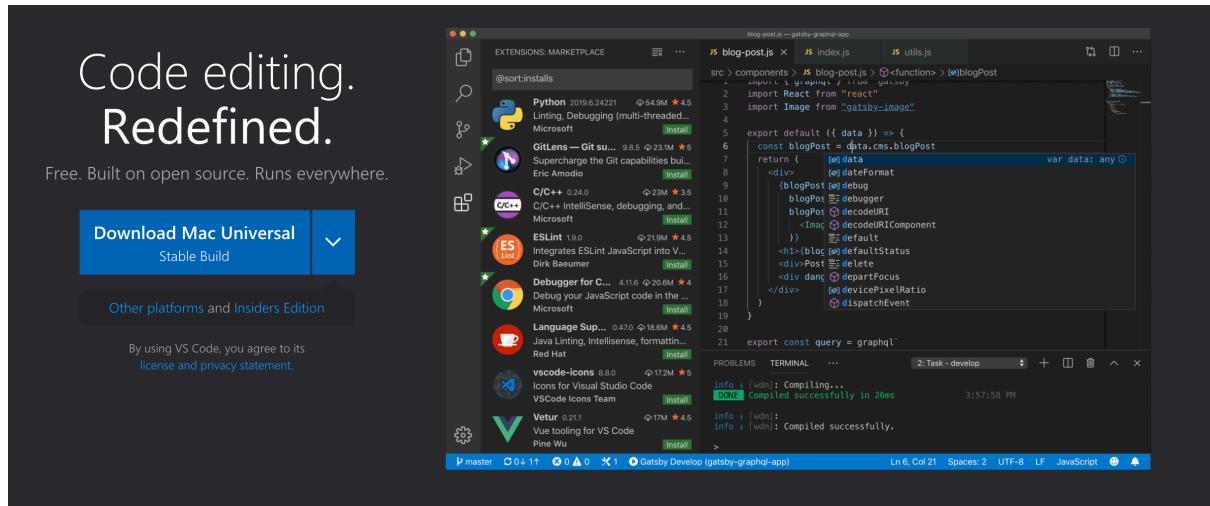
3.Add **MAVEN_HOME Environment Variable**

Reference Steps on Appendix Appendix I

3) IDE (Visual Studio Code)

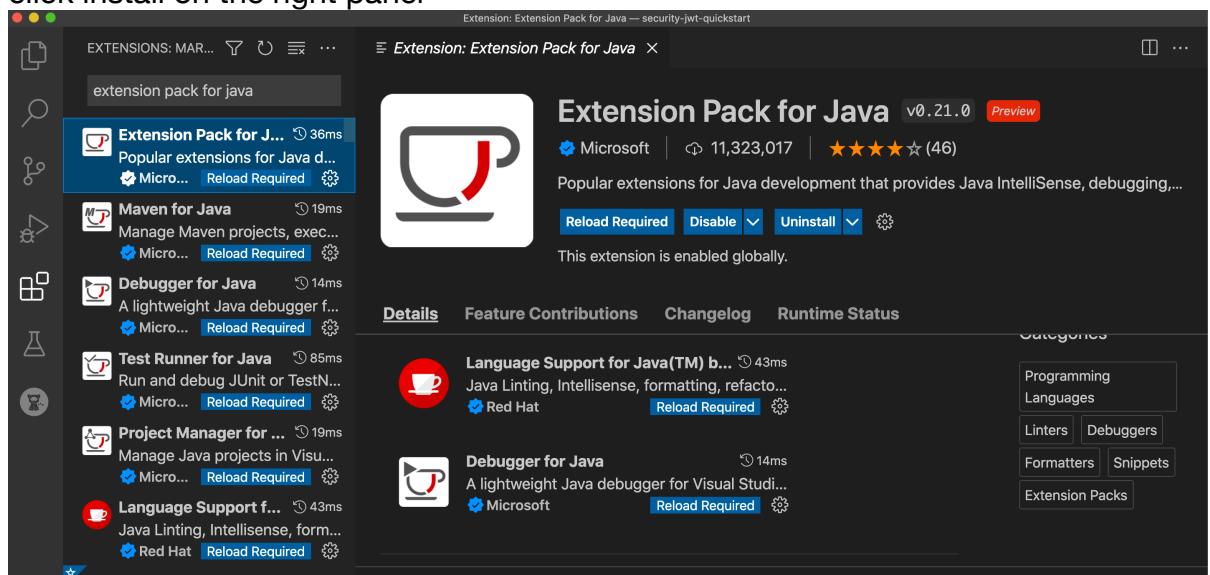
Download and complete the installation

<https://code.visualstudio.com/>



Extension Pack for Java v0.18.6 or above

Navigate to the menu on left hand side and search “extension pack for java” , then click install on the right panel



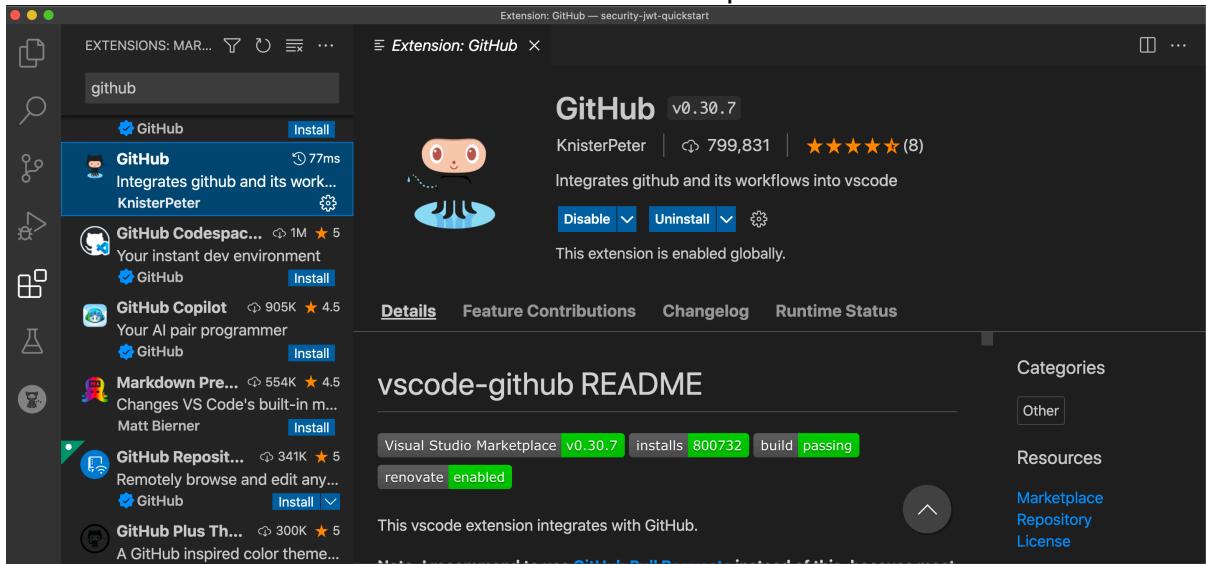
Install Quarkus Tools for Visual Studio Code

Again, to search “Quarkus” under extension and install the quarkus extation as shown below.



Install Github for Visual Studio Code

search “Quarkus” under extension and install the quarkus extation as shown below.



Lab0 Quarkus Sample Project Hello World

Download Link:
<https://code.quarkus.io/>

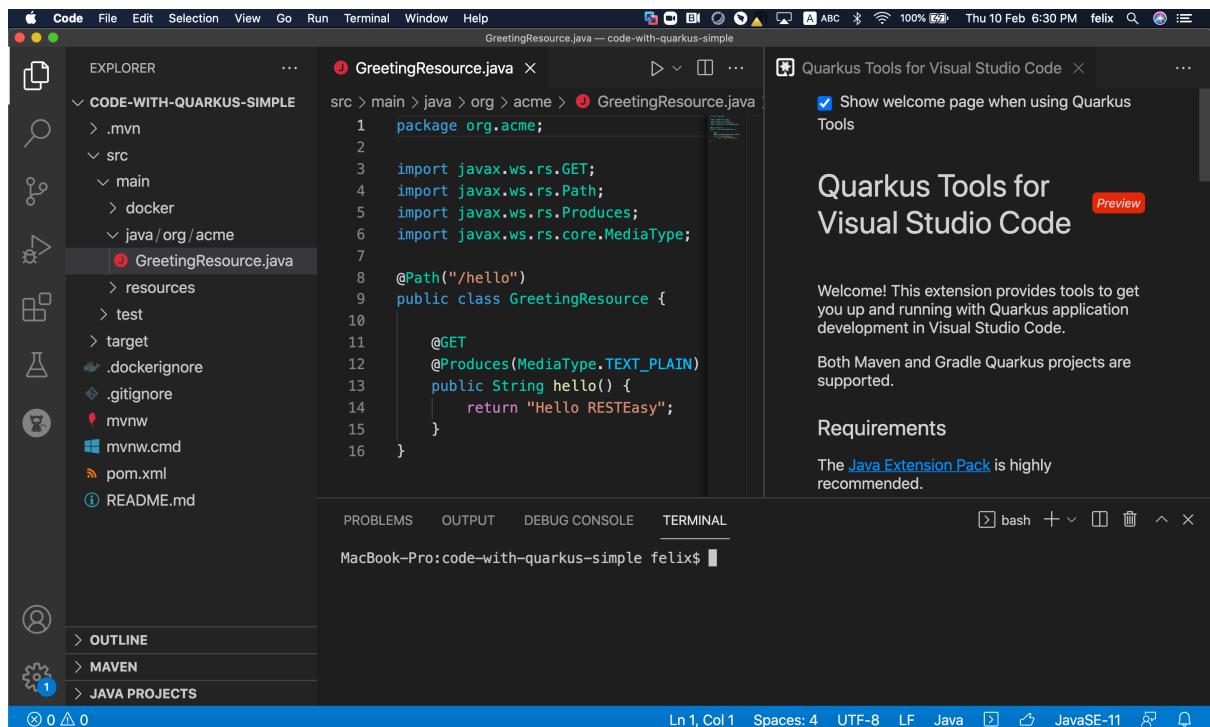
download zip and unzip to working folder

The screenshot shows the Quarkus Platform interface. At the top, it displays "QUARKUS | 2.7 io.quarkus.platform". Below that, a header bar includes "Back to quarkus.io", "Available with Enterprise Support", "Group: org.acme", "Artifact: code-with-quarkus", "Build Tool: Maven", and a "Filters" dropdown set to "origin:platform". On the right, there are buttons for "Generate your application (Esc + ⌘ + ⌂)", "Download as a zip", and "Push to GitHub". A search bar at the bottom left contains the query "origin:platform". The main content area is titled "CONFIGURE YOUR APPLICATION" and lists various Java libraries and tools under the "Web" category, such as RESTEasy JAX-RS, RESTEasy Jackson, RESTEasy JSON-B, Eclipse Vert.x GraphQL, and others.

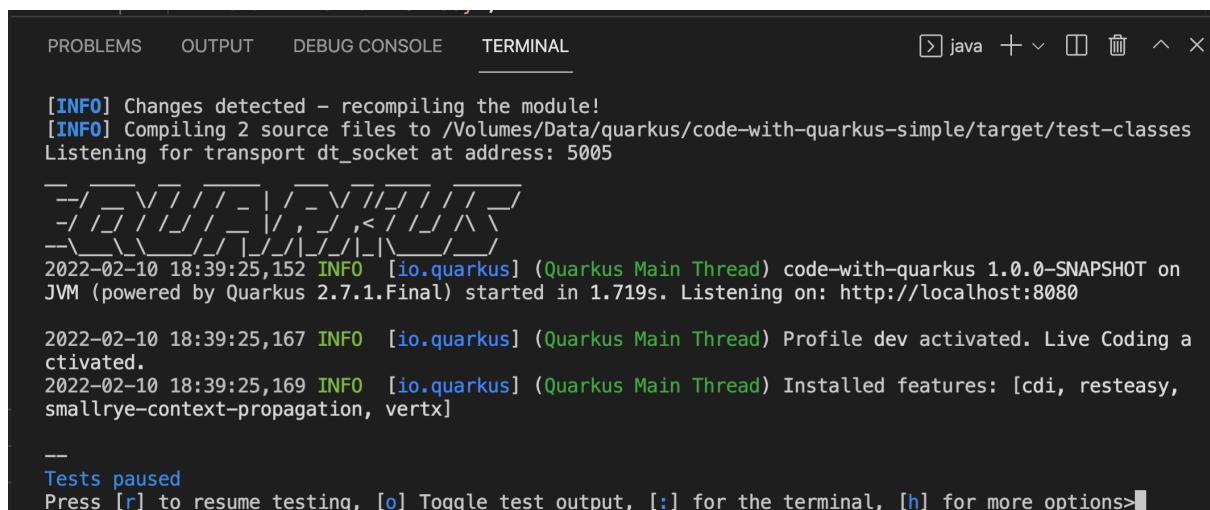
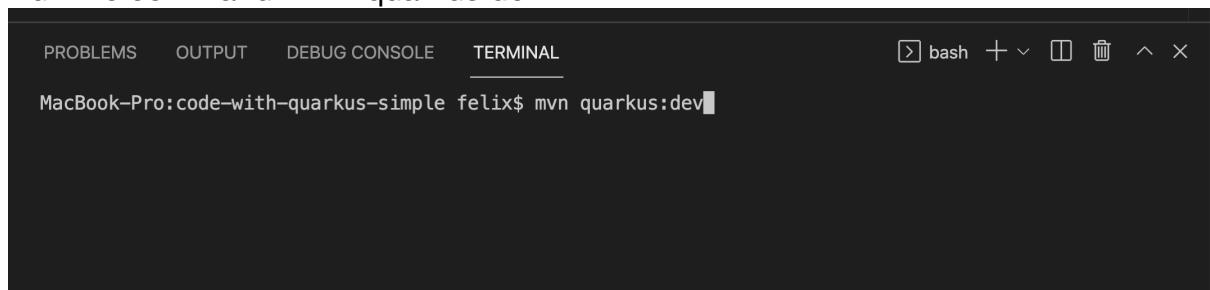
Open visual Studio Code IDE and open the folder

The screenshot shows the Visual Studio Code (VS Code) interface. The title bar indicates the file "GreetingResource.java" is open in the "code-with-quarkus-simple" workspace. The left sidebar shows a file tree with a project structure: "CODE-WITH-QUARKUS-SIMPL", ".mvn", "src" (containing "main", "docker", and "java/org/acme"), and files like ".dockerignore", ".gitignore", "mvnw", "mvnw.cmd", "pom.xml", and "README.md". The center pane displays the code for "GreetingResource.java". The right sidebar features a "Tools for Quarkus" extension preview, which provides information about getting started with Quarkus in VS Code, including requirements (Java JDK 11 or later), recommended extensions (Language Support for Java by Red Hat, Debugger for Java), and notable features (IntelliSense for properties). The status bar at the bottom shows "Ln 1, Col 1", "Spaces: 4", "UTF-8", "LF", "Java", and other icons.

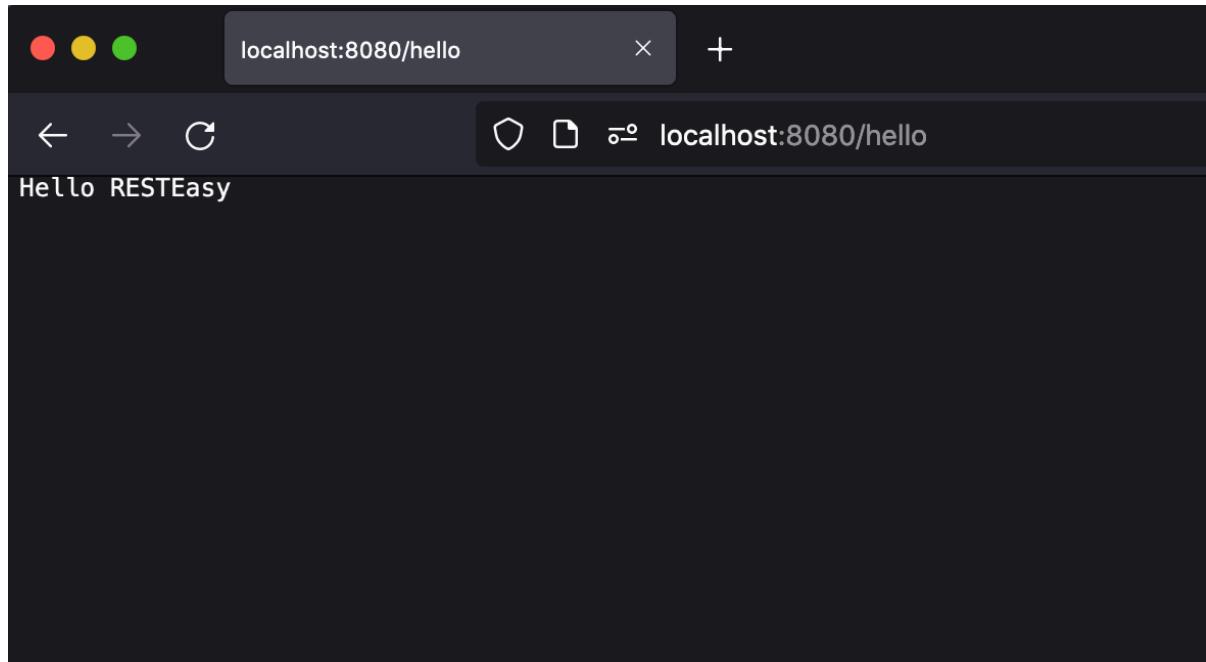
Open Terminal under the menu bar.



Run the command “mvn quarkus:dev”



Test with browser : <http://localhost:8080/hello>



LAB1 URL Parameter

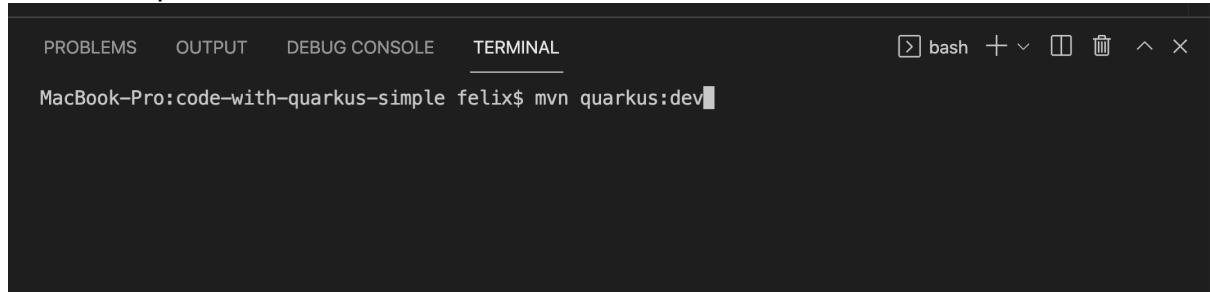
Create an other class named “HttpGetSample”

UsePathParam mapper to retrieve the URL parameter

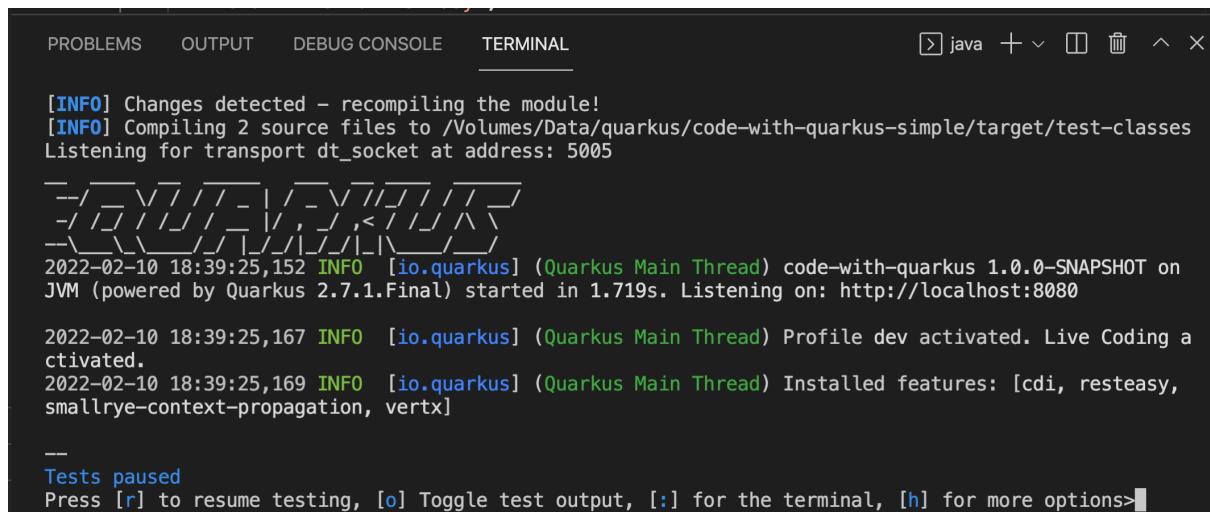
The screenshot shows an IDE interface with two tabs open: 'GreetingResource.java' and 'HttpGetSample.java'. The 'HttpGetSample.java' tab is active, displaying the following Java code:

```
1 package org.acme;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.PathParam;
6
7 @Path("/sayhello")
8 public class HttpGetSample {
9
10     @GET
11     @Path("/{name}")
12     http://localhost:8080/sayhello/{name}
13     public String sayHello(@PathParam("name") String name) {
14         return "Hello " + name + " !";
15     }
}
```

Run mvn quarkus:dev

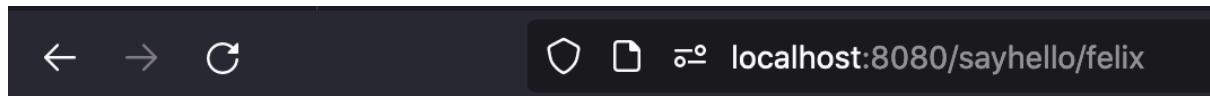


```
MacBook-Pro:code-with-quarkus-simple felix$ mvn quarkus:dev
```



```
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to /Volumes/Data/quarkus/code-with-quarkus-simple/target/test-classes
Listening for transport dt_socket at address: 5005
--/--\v/\_/_/_|/_\v/_/_/_/_/
--/_/_/_/_/_/_\_,_,</_/_/\_\
--\_\_\_/_/_/_/_/_/_/_/_/_/_\_
2022-02-10 18:39:25,152 INFO [io.quarkus] (Quarkus Main Thread) code-with-quarkus 1.0.0-SNAPSHOT on
JVM (powered by Quarkus 2.7.1.Final) started in 1.719s. Listening on: http://localhost:8080
2022-02-10 18:39:25,167 INFO [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding a
ctivated.
2022-02-10 18:39:25,169 INFO [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, resteasy,
smallrye-context-propagation, vertx]
--
Tests paused
Press [r] to resume testing, [o] Toggle test output, [:] for the terminal, [h] for more options>
```

Test from Brower locally

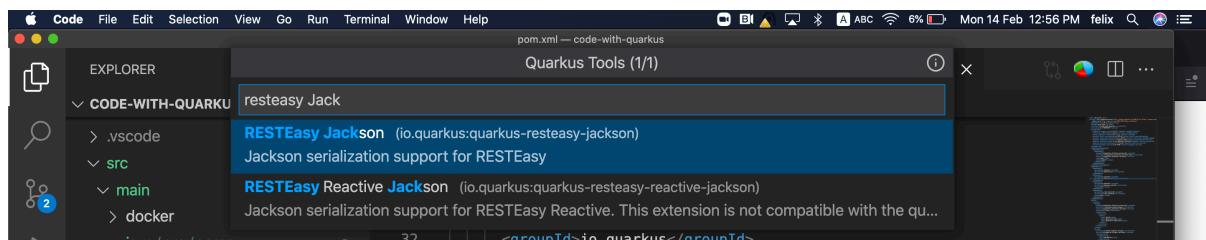
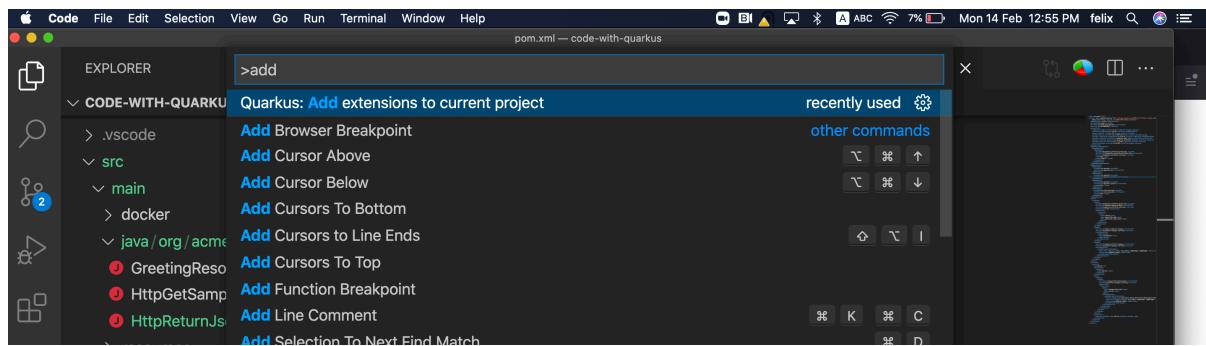


Hello felix !

LAB2 Response with JSON Obj

Importing the RESTEasy/JAX-RS and [Jackson](#) extensions

Click “View” > “Command Palette”, type “add” and choose “Quarkus: Add extensions to current project”



and press enter to import the dependency.

Or

(Manual method) add the dependency

```

> test
> target
.dockerignore
.gitignore
mvnw
mvnw.cmd
pom.xml

```

54 </dependency>
35 <dependency>
36 <groupId>io.quarkus</groupId>
37 <artifactId>quarkus-resteasy</artifactId>
38 </dependency>
39 <dependency>
40 <groupId>io.quarkus</groupId>
41 <artifactId>quarkus-resteasy-jackson</artifactId>
42 </dependency>

open pom.xml and add the dependency

```

<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
</dependency>

```

Create a class named “Lab2HttpReturnJsonObj” and a Student class for the example of converting the Obj to JSON response.

Student.java

```

package org.acme;

public class Student {
    private String name;
    private String gender;
    private String year;

    public Student(String name, String gender , String year) {
        this.name = name;
        this.gender = gender;
    }
}

```

```

        this.year = year;
    }

    public String getGender() {
        return gender;
    }
    public String getYear() {
        return year;
    }
    public void setYear(String year) {
        this.year = year;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}

```

Lab2HttpReturnJsonObj.java

```

package org.acme;

import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/getStudent")
@Produces(MediaType.APPLICATION_JSON)
@ApplicationScoped
public class Lab2HttpReturnJsonObj {
    @GET
    public Student sayHello() {
        Student s = new Student("M", "Felix T", "year 12");
        return s;
    }
}

```

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure under **CODE-WITH-QUARKUS**, including .github, .mvn, .vscode, src (with main and docker), java/org/acme (containing Lab0GreetingResource.java, Lab1HttpGetSample.java, Lab2HttpReturnJsonObj.java, and Lab2HttpReturnJsonObjList.java), and Student.java.
- EDITOR**: Displays the **Student.java** code:

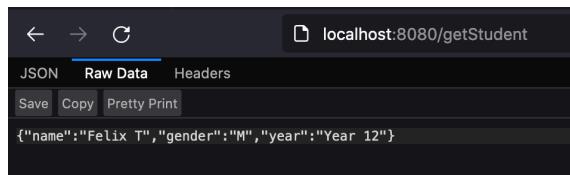
```
Student.java — code-with-quarkus
src > main > java > org > acme > Student.java
1 package org.acme;
2
3 public class Student {
4     private String name;
5     private String gender;
6     private String year;
7
8     public Student(String name, String gender , String year) {
9         this.name = name;
10        this.gender = gender;
11        this.year = year;
12    }
13
14    public String getGender() {
15        return gender;
16    }
17    public String getYear() {
18        return year;
19    }
20    public void setYear(String year) {
21        this.year = year;
22    }
23    public String getName() {
24        return name;
25    }
26    public void setName(String name) {
27        this.name = name;
28    }
29    public void setGender(String gender) {
30        this.gender = gender;
31    }
32}
33}
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure under **CODE-WITH-QUARKUS**, including .github, .mvn, .vscode, src (with main and docker), java/org/acme (containing Lab0GreetingResource.java, Lab1HttpGetSample.java, Lab2HttpReturnJsonObj.java, and Lab2HttpReturnJsonObjList.java), and Student.java.
- EDITOR**: Displays the **Lab2HttpReturnJsonObj.java** code:

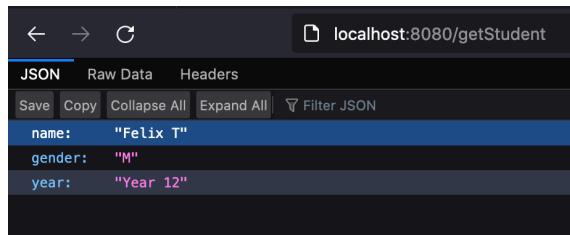
```
Lab2HttpReturnJsonObj.java — code-with-quarkus
src > main > java > org > acme > Lab2HttpReturnJsonObj.java
1 package org.acme;
2
3 import javax.enterprise.context.ApplicationScoped;
4 import javax.ws.rs.GET;
5 import javax.ws.rs.Path;
6 import javax.ws.rs.Produces;
7 import javax.ws.rs.core.MediaType;
8
9 @Path("/getStudent")
10 @Produces(MediaType.APPLICATION_JSON)
11 @ApplicationScoped
12 public class Lab2HttpReturnJsonObj {
13     @GET
14     http://localhost:8080/getStudent
15     public Student sayHello() {
16         Student s = new Student("M","Felix T","year 12");
17         return s;
18     }
19 }
```

Test the JSON response



A screenshot of a browser developer tools JSON tab titled "localhost:8080/getStudent". The "Raw Data" tab is selected. The JSON response is:

```
{"name": "Felix T", "gender": "M", "year": "Year 12"}
```



A screenshot of a browser developer tools JSON tab titled "localhost:8080/getStudent". The "Raw Data" tab is selected. The JSON response is expanded:

```
name: "Felix T"  
gender: "M"  
year: "Year 12"
```

LAB3 HTTP Post Consume JSON Obj

Create “Lab3HttpPostAndConsumeJsonObj.java” and add the consume annotation

The screenshot shows a code editor with two panes. The left pane displays the project structure:

```
CODE-WITH-QUARKUS-LAB
├── .mvn
└── src
    └── main
        ├── docker
        └── java/org/acme
            ├── GreetingResource.java
            ├── Lab0GreetingResource.java
            ├── Lab1HttpGetSample.java
            ├── Lab2HttpReturnJsonObj.java
            ├── Lab2HttpReturnJsonObjList.java
            ├── Lab3HttpPostAndConsumeJsonObj.java
            └── Student.java
        └── resources
    └── test
    └── target
└── .dockerignore
```

The right pane shows the code for `Lab3HttpPostAndConsumeJsonObj.java`:

```
import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/addStudent")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@ApplicationScoped
public class Lab3HttpPostAndConsumeJsonObj {

    @POST
    public List<Student> addStudents(List<Student> student) {
        return student;
    }
}
```

Eg : POST Student Obj and convert to Java Student Obj

The screenshot shows a Postman request configuration:

- Method: POST
- URL: `http://localhost:8080/addStudent`
- Body tab selected
- JSON selected under the dropdown
- Body content:

```
[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]
```

The response view shows the JSON output:

```
[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]
```

```
curl --location --request POST 'http://localhost:8080/addStudent' \
--header 'Content-Type: application/json' \
--data-raw '[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]'
```

LAB4 JWT

Add extention

```
smallrye-jwt
smallrye-jwt-build
resteas
resteasy-jackson
```

```
./mvnw quarkus:add-extension -Dextensions="resteas,resteeasy-jackson,smallrye-jwt,smallrye-jwt-build"
```

pom.xml

```
....  
<dependency>  
    <groupId>io.smallrye</groupId>  
    <artifactId>smallrye-jwt-build</artifactId>  
</dependency>  
  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-resteasy-jackson</artifactId>  
</dependency>  
  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-smallrye-jwt</artifactId>  
</dependency>  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-resteasy</artifactId>  
</dependency>  
....
```

Open the `src/main/java/org/acme/security/jwt/TokenSecuredResource.java` file and see the following content:

Path `/secured/permit-all` , to display the SecurityContext info.

Path `/secured/helloadmin`, to allow the role “admin” to access the function

TokenSecuredResource.java

```
package org.acme.security.jwt;  
  
import javax.annotation.security.PermitAll;  
import javax.annotation.security.RolesAllowed;  
import javax.enterprise.context.RequestScoped;  
import javax.inject.Inject;  
import javax.ws.rs.GET;  
import javax.ws.rs.InternalServerErrorException;  
import javax.ws.rs.POST;  
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.Context;  
import javax.ws.rs.core.MediaType;  
import javax.ws.rs.core.SecurityContext;  
  
import org.eclipse.microprofile.jwt.JsonWebToken;
```

```

@Path("/secured")
@RequestScoped
public class TokenSecuredResource {

    @Inject
    JsonWebToken jwt;

    @GET
    @Path("permit-all")
    @PermitAll
    @Produces(MediaType.TEXT_PLAIN)
    public String hello(@Context SecurityContext ctx) {
        return getResponseString(ctx);
    }

    @GET
    @Path("roles-allowed")
    @RolesAllowed({ "User", "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloRolesAllowed(@Context SecurityContext ctx) {
        return getResponseString(ctx) + ", birthdate: " + jwt.getClaim("birthdate").toString() + ", exp:" +
        jwt.getClaim("exp").toString();
    }

    private String getResponseString(SecurityContext ctx) {
        String name;
        if (ctx.getUserPrincipal() == null) {
            name = "anonymous";
        } else if (!ctx.getUserPrincipal().getName().equals(jwt.getName())) {
            throw new InternalServerErrorException("Principal and JsonWebToken names do not match");
        } else {
            name = ctx.getUserPrincipal().getName();
        }
        return String.format("hello + %s," +
            " + isHttps: %s," +
            " + authScheme: %s," +
            " + hasJWT: %s",
            name, ctx.isSecure(), ctx.getAuthenticationScheme(), hasJwt());
    }

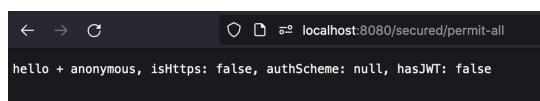
    private boolean hasJwt() {
        return jwt.getClaimNames() != null;
    }

    @GET
    @POST
    @Path("helloworld")
    @RolesAllowed({ "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloworld(@Context SecurityContext ctx) {
        return "I am admin. " + getResponseString(ctx) + ", birthdate: " + jwt.getClaim("birthdate").toString() + ", exp:" +
        jwt.getClaim("exp").toString();
    }

}

```

Run mvn quarkus:dev to test
<http://localhost:8080/secured/permit-all>



POSTMAN

<http://localhost:8080/secured/helloadmin>

>> 401 Unauthorized

Untitled Request

The screenshot shows the Postman interface with a request to `http://localhost:8080/secured/helloadmin`. The 'Params' tab is selected, showing a single entry: 'Key' (Value) and 'Value' (Description). The 'Body' tab is selected, showing a JSON response with a single key '1'. The 'Headers' tab shows two entries: 'Status: 401 Unauthorized' and 'Time: 14 ms'. A tooltip for '401 Unauthorized' provides a detailed explanation: 'Similar to 403 Forbidden, but specifically for use when authentication is possible but has failed or not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.'

Open the `src/main/java/org/acme/security/jwt/GenerateToken.java` file and see the following content:

GenerateToken.java

```
package org.acme.security.jwt;

import java.util.Arrays;
import java.util.HashSet;

import javax.enterprise.context.RequestScoped;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import io.smallrye.jwt.build.Jwt;
import org.acme.security.UserLoginBean;
import org.eclipse.microprofile.jwt.Claims;

@Path("/authen")
@Produces(MediaType.APPLICATION_JSON)
@RequestScoped
public class GenerateToken {
    /**
     * Generate JWT token
     */
    public static void main(String[] args) {
        String token =
            Jwt.issuer("https://felixtsang.com/issuer")
                .upn("felix@linux.com")
```

```

// .preferredUserName("felix")
    .groups(new HashSet<>(Arrays.asList("User", "Admin")))
    .claim(Claims.birthdate.name(), "2001-07-13")
    // .claim(Claims.exp.name(), "2022-07-13")
    .sign();
System.out.println(token);
}

public static String getJwtToken() {
String token =
Jwt.issuer("https://felixtsang.com/issuer")
.upn("felix@linux.com")
.preferredUserName("felix")
.claim("name", "nathan")
.groups(new HashSet<>(Arrays.asList("User", "Admin")))
.claim(Claims.birthdate.name(), "2001-07-13")
.claim("userid", "123")
.sign();
return token;
}

@POST
@Produces(MediaType.APPLICATION_JSON)
@Path("login")
public UserLoginBean login() {

UserLoginBean userLoginBean = new UserLoginBean();
userLoginBean.setJwtToken(getJwtToken());

return userLoginBean;
// return getJwtToken();
}

@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("logindemo")
public UserLoginBean loginDemo() {

UserLoginBean userLoginBean = new UserLoginBean();
userLoginBean.setJwtToken(getJwtToken());
userLoginBean.setUserId("1");
userLoginBean.setUserName("Felix");

return userLoginBean;
// return getJwtToken();
}
}

```

Generate the private.key to decryt the jwt token and public to encrypt the jwt

Generating Keys with OpenSSL

It is also possible to generate a public and private key pair using the OpenSSL command line tool.

openssl commands for generating keys

```
openssl genrsa -out rsaPrivateKey.pem 2048
openssl rsa -pubout -in rsaPrivateKey.pem -out publicKey.pem
```

An additional step is needed for generating the private key for converting it into the PKCS#8 format.

openssl command for converting private key

```
openssl pkcs8 -topk8 -nocrypt -inform pem -in rsaPrivateKey.pem -outform pem -out
privateKey.pem
```

You can use the generated pair of keys insted of the keys used in this quickstart.

Or generate the key pairs online if no openssl command installed

[https://cryptotoools.net/rsagen](https://cryptotools.net/rsagen)

application.properties

```
mp.jwt.verify.publickey.location=publicKey.pem  
mp.jwt.verify.issuer=https://felixsang.com/issuer  
quarkus.native.resources.includes=publicKey.pem  
smallrye.jwt.sign.key.location=privateKey.pem  
quarkus.http.cors=true  
quarkus.http.cors.origins=http://localhost,http://127.0.0.1  
quarkus.http.cors.method=GET,POST
```

/src/main/resources/

application.properties

privateKey.pem

publicKey.pem

rsaPrivateKey.pem

1. generate the jwt token

localhost:8080//authen/logindemo

The screenshot shows a Postman interface with the URL `localhost:8080//authen/logindemo` in the search bar. The response is displayed in JSON format:

```
jwtToken: "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2ZlbGl49XQnDtYwkhsvuwCurDHcupl7wN2hTuZLlUQ_YLY7T3dunvm0OuTmbiFZa-89ErvyNc  
userId: "1"  
userName: "Felix"
```

copy the jwtToken and use the postman to verify

under the postman

1. select POST method
2. `http://localhost:8080/secured/helloadmin`
3. Authorizatoin > Type > “Bearer Token” > patse the token on right hand side

The screenshot shows a Postman interface with a POST request to `http://localhost:8080/secured/helloadmin`. The **Authorization** tab is selected, displaying a **Bearer Token** parameter with the value `eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lzczVlcIisInVwbii6ImZlbGl4QGxpbnV4LmNvbSIsInByZWZlcnJlZF91c2VybmtSI6ImZlbGl4IiwbmFtZSI6Im5hdGhbiiIsImdyb3VwcyI6WyJVc2VyIiwiQWRtaW4iXSviYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ00TIyNzgwLCJqdGki0iI5MmVhZDc4Yi04MzMltQyZGYtODNiOC04MDJlMmM3NmJkYTcif0.t3RWRF7y6wwt--9XQnDtYwkhsuwCurDHcupl7wN2hTuZLlUQ_YLY7T3dunvm00uTmbiFZa-89ErvyNcE_vfSwalz6mwJs4gToKE7zQNwg-N_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNETqd_R_14Feo_n8NV98-j3A3SjNdIj7zY6xo4_rw7cpkoAkalCSZAtcJDbizT-2ssCzSPc6BeNFTm4GM2zA3DPB7E0PucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-lSiZ_rCDa5X9lkeeWf2Uyjg''`. The response status is 200 OK with a size of 204 B.

raw curl like the following:

```
curl --location --request POST 'http://localhost:8080/secured/helloadmin' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3Mi0iJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lzczVlcIisInVwbii6ImZlbGl4QGxpbnV4LmNvbSIsInByZWZlcnJlZF91c2VybmtSI6ImZlbGl4IiwbmFtZSI6Im5hdGhbiiIsImdyb3VwcyI6WyJVc2VyIiwiQWRtaW4iXSviYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ00TIyNzgwLCJqdGki0iI5MmVhZDc4Yi04MzMltQyZGYtODNiOC04MDJlMmM3NmJkYTcif0.t3RWRF7y6wwt--9XQnDtYwkhsuwCurDHcupl7wN2hTuZLlUQ_YLY7T3dunvm00uTmbiFZa-89ErvyNcE_vfSwalz6mwJs4gToKE7zQNwg-N_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNETqd_R_14Feo_n8NV98-j3A3SjNdIj7zY6xo4_rw7cpkoAkalCSZAtcJDbizT-2ssCzSPc6BeNFTm4GM2zA3DPB7E0PucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-lSiZ_rCDa5X9lkeeWf2Uyjg'''
```

Challenge yourself

Lab 5 Install Kubernetes Cluster (1 Master , 1 Worker)

Install Kubernetes

Ubuntu 18.04.5 ISO Download

<https://old-releases.ubuntu.com/releases/18.04.5/ubuntu-18.04.5-live-server-amd64.iso>

1. Prepare two Ubuntu VM, 1 for master node , 1 for worker node
2. Download the k8smaster.sh and k8sworker.sh from github
https://github.com/lftsang/cloudnative/tree/main/k8s_installation_script

eg :

On Master node

```
 wget https://raw.githubusercontent.com/lftsang/cloudnative/main/k8s_installation_script/k8smaster.sh
```

On Worker node

```
 wget https://raw.githubusercontent.com/lftsang/cloudnative/main/k8s_installation_script/k8sworker.sh
```

3. Under master node , run the k8smaster.sh by root.
4. Under worker node , run the k8sworker.sh by root.

```
root@k8sworkertemp:~# ./k8sworker.sh
```

5. Run “kubeadm token create --print-join-command” to get the join cluster command , and run on worker node

```
root@k8smastertemp:~# kubeadm token create --print-join-command  
kubeadm join 192.168.205.172:6443 --token c6w74t.4q0rf79n9o0mo02s --discovery-token-ca-cert-hash sha256:7bc665462e22c1c9009982bdb2c199ef1928da324c8606fc84cf630320f46fc
```

6. Run the join command generated from step 5 .

```
root@k8sworkertemp:~# kubeadm join 192.168.205.172:6443 --token c6w74t.4q0rf79n9o0mo02s --discovery-token-ca-cert-hash sha256:  
7bc665462e22c1c9009982bdb2c199ef1928da324c8606fc84cf630320f46fc
```

7. Run “kubectl get node” under master to verify the node status

```
root@k8smastertemp:~# kubectl get nodes  
NAME     STATUS   ROLES      AGE   VERSION  
k8smastertemp   Ready    control-plane,master   42m   v1.21.1  
k8sworkertemp   Ready    <none>    67s   v1.21.1
```

8. Run “kubectl run nginx --image=nginx” to create a pod for testing the workload creation

```
root@k8smastertemp:~# kubectl get pod  
NAME    READY   STATUS    RESTARTS   AGE  
nginx  1/1     Running   0          36m
```

9. dfdfdf

Lab 6 Create Quay.io registry Account

Quay.io provides a container image registry service. In this lab , we will build our app and push to this quay.io image registry.

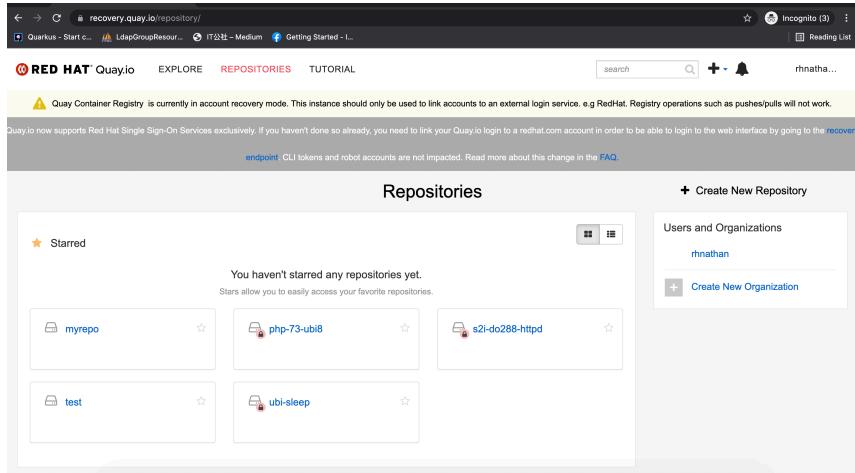
1. Registe a RedHat ID first

https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/auth?response_type=code&client_id=https%3A%2Fwww.redhat.com%2Fwapps%2Fugc-oidc&redirect_uri=https%3A%2F%2Fwww.redhat.com%2Fwapps%2Fugc%2Fsso%2Flogin&state=0ee9d56f-91d8-

2. Sign in with RedHat SSO created in step 1.

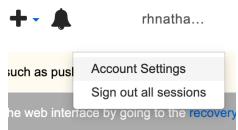
<https://quay.io/signin/>

If you already has a quay.io , please login with the following link
<https://recovery.quay.io/>



3. Obtain the access token

Go to “Account Settings” under the profile icon on top right corner



Click “Generate Encrypted Password” to get the access token

Username & Encrypted Password:	
rhnathan	Yxg0HGGCJMTh4Qa9hLIXofnRgvob8L2y1sUWGGGN5s5BAgCxxxxxxxxxxxxxx

copy it and save in your local machine.

4. Create New Organization

Name “**cmhklab{\$}cloudnativelab**” + your prefix

cmhklab{\$}cloudnativelab

eg ; cmhklab1cloudnativelab

eg ; cmhklab2cloudnativelab

eg ; cmhklab3cloudnativelab

Quay Container Registry is currently in account recovery mode. This instance should only be used to link accounts to an external login service.

Quay.io now supports Red Hat Single Sign-On Services exclusively. If you haven't done so already, you need to link your Quay.io login to a redhat.com account in order to be able to login to the web interface by going to the [recovery endpoint](#). CLI tokens and robot accounts impacted. Read more about this change in the [FAQ](#).

← Repositories Create New Organization

Organization Name
cmhklab500cloudnativelab

Organization Email
cmhklab500cloudnativelab@testlabdemo1243.com

I'm not a robot

Create Organization

Create

Quay Container Registry is currently in account recovery mode. This instance should only be used to link accounts to an external login service. e.g RedHat. Registry operations impacted. Read more about this change in the [FAQ](#).

cmhklab500cloudnativelab + Create New Repository

0 - 0 of 0 Filter Repositories...

This namespace doesn't have any visible repositories.

Named “tutorial-app” and select “Public” Access

Quay Container Registry is currently in account recovery mode. This instance should only be used to link accounts to an external login service. e.g RedHat. Registry operations impacted. Read more about this change in the [FAQ](#).

← Repositories Create New Repository

cmhklab500cloudnativelab / tutorial-app

Click to set repository description

Public
Anyone can see and pull from this repository. You choose who can push.

Private
You choose who can see, pull and push from/to this repository.

(Empty repository)
 Initialize from a Dockerfile
 Link to a Custom Git Repository Push

Create Public Repository

Lab 7 Install podman/docker client and build the container image with quarkus extension

What is Podman

Podman is a daemonless, open source, Linux native tool designed to make it easy to find, run, build, share and deploy applications using Open Containers Initiative (OCI) Containers and Container Images. Podman provides a command line interface (CLI) familiar to anyone who has used the Docker Container Engine. Most users can simply alias Docker to Podman (alias docker=podman) without any problems. Similar to other common Container Engines (Docker, CRI-O, containerd), Podman relies on an OCI compliant Container Runtime (runc, crun, runv, etc) to interface with the operating system and create the running containers. This makes the running containers created by Podman nearly indistinguishable from those created by any other common container engine.

<https://docs.podman.io/en/latest/>

Install “podman” / “docker” client on the local machine.

try something new about podman ☺

Podman is a tool for running Linux containers. You can do this from a MacOS desktop as long as you have access to a linux box either running inside of a VM on the host, or available via the network. Podman includes a command, podman machine that automatically manages VM's.

Eg : Podman installation on Mac by brew

<https://podman.io/getting-started/installation>

```
able alias docker = podman  
vi ~/.bash_profile  
alias docker=podman
```

Eg: Docker installation on Windows

<https://docs.docker.com/desktop/windows/install/>

```
MacBook-Pro:- Felix$ podman --version  
podman version 3.4.4  
MacBook-Pro:- Felix$
```

```
init and start the podman service  
alias docker=podman  
docker machine init  
docker machine start  
docker login quay.io
```

```

MacBook-Pro:~ felix$ docker login quay.io
Username: rhnathan
Password: with a t
Login Succeeded felix.tsh@gmail.com
MacBook-Pro:~ felix$ 

```

5. Install Quarkus Kubernetes Extension

We use JIB to build the image , which not like docker need the daemon traditionally.

Command :

```
./mvnw quarkus:add-extension -Dextensions="resteasy,kubernetes,jib"
```

Or edit the pom.xml

```

<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-kubernetes</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-container-image-jib</artifactId>
</dependency>

```

```

pom.xml
45   <dependency>
46     <groupId>io.quarkus</groupId>
47     <artifactId>quarkus-resteasy</artifactId>
48   </dependency>
49   <dependency>
50     <groupId>io.quarkus</groupId>
51     <artifactId>quarkus-resteasy-jackson</artifactId>
52   </dependency>
53   <dependency>
54     <groupId>io.quarkus</groupId>
55     <artifactId>quarkus-kubernetes</artifactId>
56   </dependency>
57   <dependency>
58     <groupId>io.quarkus</groupId>
59     <artifactId>quarkus-container-image-jib</artifactId>
60   </dependency>
61   <dependency>
62     <groupId>io.quarkus</groupId>
63     <artifactId>quarkus-junit5</artifactId>
64     <scope>test</scope>
65   </dependency>

```

edit the “application.properties” under resources folder. Setting up the kubernetes parameters and registry information.

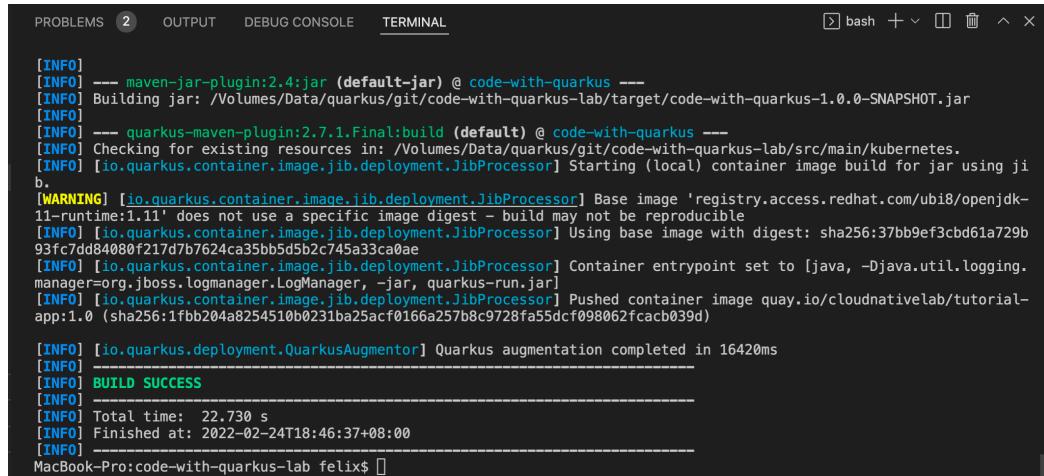
```

# Configuration file
# key = value
# tutorial https://redhat-developer-demos.github.io/quarkus-tutorial/quarkus-tutorial/kubernetes.html
quarkus.container-image.registry=quay.io
#group = organization name
quarkus.container-image.group=cloudnativelab
quarkus.container-image.name=tutorial-app
quarkus.container-image.tag=1.0
#quarkus.kubernetes.service-type=load-balancer
quarkus.kubernetes.service-type=node-port
quarkus.container-image.username=rhnathan
quarkus.container-image.password=WxF4nj12AUSZLxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

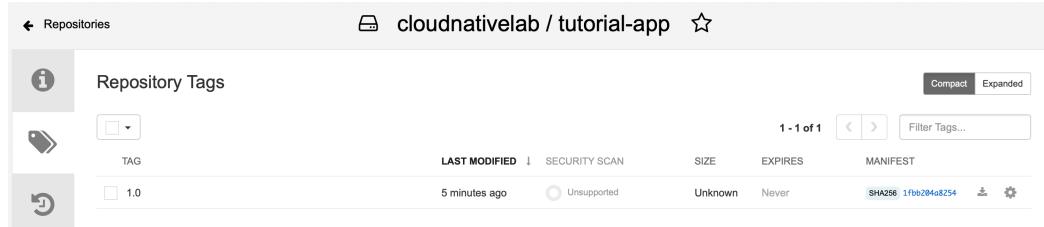
Run maven build command to build image and push to quay.io

```
mvn clean package -DskipTests -Dquarkus.container-image.push=true
```



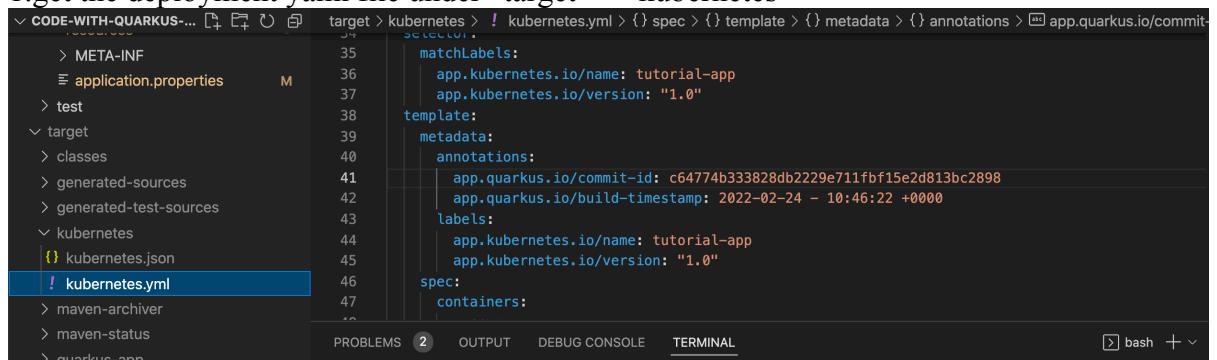
```
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ code-with-quarkus ---
[INFO] Building jar: /Volumes/Data/quarkus/git/code-with-quarkus-lab/target/code-with-quarkus-1.0.0-SNAPSHOT.jar
[INFO] --- quarkus-maven-plugin:2.7.1.Final:build (default) @ code-with-quarkus ---
[INFO] Checking for existing resources in: /Volumes/Data/quarkus/git/code-with-quarkus-lab/src/main/kubernetes.
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Starting (local) container image build for jar using jib.
[WARNING] [io.quarkus.container.image.jib.deployment.JibProcessor] Base image 'registry.access.redhat.com/ubi8/openjdk-11-runtime:1.11' does not use a specific image digest - build may not be reproducible
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Using base image with digest: sha256:37bb9ef3cbd61a729b93fc7dd84080f217d7b7624ca35bb5d5b2c745a33ca0ae
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Container entrypoint set to [java, -Djava.util.logging.manager=org.jboss.logmanager.LogManager, -jar, quarkus-run.jar]
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Pushed container image quay.io/cloudnativelab/tutorial-app:1.0 (sha256:1ffb204a8254510b0231ba25acf0166a257b8c9728ra55dcf0980062fcac039d)
[INFO] [io.quarkus.deployment.QuarkusAugmentor] Quarkus augmentation completed in 16420ms
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 22.730 s
[INFO] Finished at: 2022-02-24T18:46:37+08:00
[INFO] -----
MacBook-Pro:code-with-quarkus-lab felix$
```

And login back to quay.io and see the image was created under the organization



Lab 8 Deploy the app on kubernetes cluster

1.get the deployment yaml file under “target” > “kubernetes”



```
target > kubernetes > ! kubernetes.yaml > {} spec > {} template > {} metadata > {} annotations > app.quarkus.io/commit-id: c64774b333828db2229e711fbf15e2d813bc2898
      > matchLabels:
        app.kubernetes.io/name: tutorial-app
        app.kubernetes.io/version: "1.0"
      template:
        metadata:
          annotations:
            app.quarkus.io/commit-id: c64774b333828db2229e711fbf15e2d813bc2898
            app.quarkus.io/build-timestamp: 2022-02-24 - 10:46:22 +0000
          labels:
            app.kubernetes.io/name: tutorial-app
            app.kubernetes.io/version: "1.0"
        spec:
          containers:
```

copy the kubernetes.yaml to k8smater and run
kubectl apply -f kubernetes.yaml

```

root@k8smaster:~# kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
hello-world-59966754c9-4db8q   1/1    Running   10        8d
hello-world-59966754c9-5xfd7   1/1    Running   11        8d
tutorial-app-5ccb7c9fb5-f4h5d   1/1    Running   0         93s
root@k8smaster:~# kubectl get svc
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
example-service   NodePort   10.111.33.199   <none>       8080:31653/TCP   8d
kubernetes   ClusterIP  10.96.0.1    <none>       443/TCP    110d
tutorial-app   NodePort   10.102.157.130  <none>       80:32755/TCP   8d
root@k8smaster:~# curl http://192.168.205.170:32755/hello
Hello HK 123root@k8smaster:~#

```

Appendix

Appendix I JDK Installation

For Mac

1.The detailed procedures of install JDK

<https://devwithus.com/install-java-jdk#mac>

For Windows OS

2.The detailed procedures of install JDK

<https://devwithus.com/install-java-windows-10/>

Appendix II Maven Installation

For Mac

1.The detailed procedures of **How to Install Maven on Mac**

<https://devwithus.com/install-maven-mac/>

For Windows OS

2.The detailed procedures of **How to Install Maven on windows**

<https://devwithus.com/install-maven-windows/>