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

1) JDK Installation

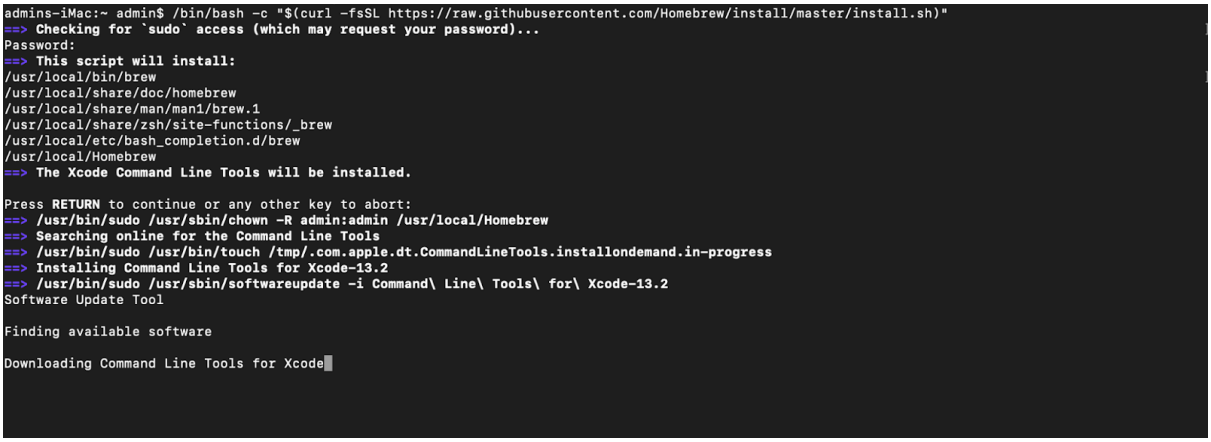
For Mac (Detailed Procedures ref “Appendix I JDK Installation”)
Install JDK 17 or above by dmg installer
https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html
java -version
java version "17.0.1" 2021-10-19 LTS
Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)
Java HotSpot(TM) 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"

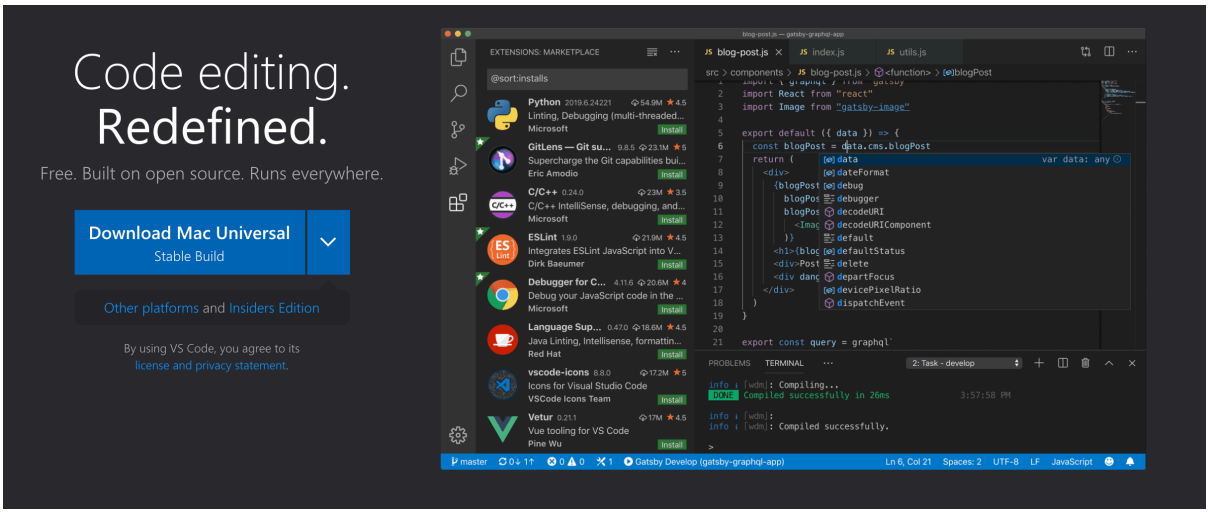


For Windows
Maven needs JDK to be installed before configuration.
1.Finish JDK setup and set up JAVA_HOME **Environment Variable in the previous steps**
2.Download Maven Zip and unzip to your working folder
3.**Add *MAVEN_HOME* Environment Variable**
Reference Steps on Appendix Appendix I

3) IDE (Visual Studio Code)

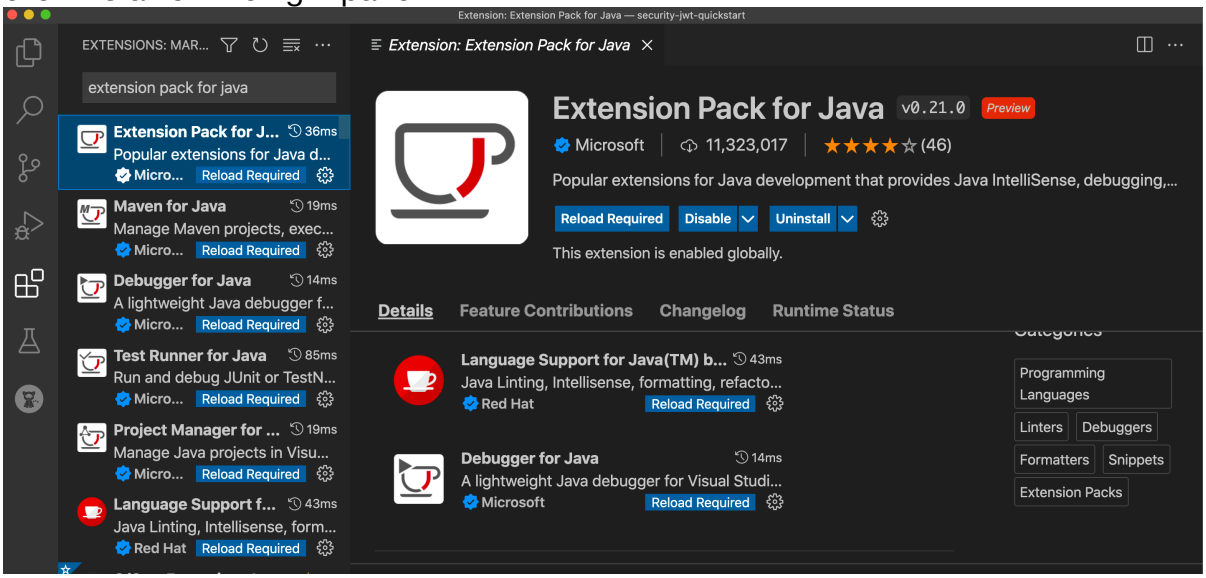
Download and complete the installation

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

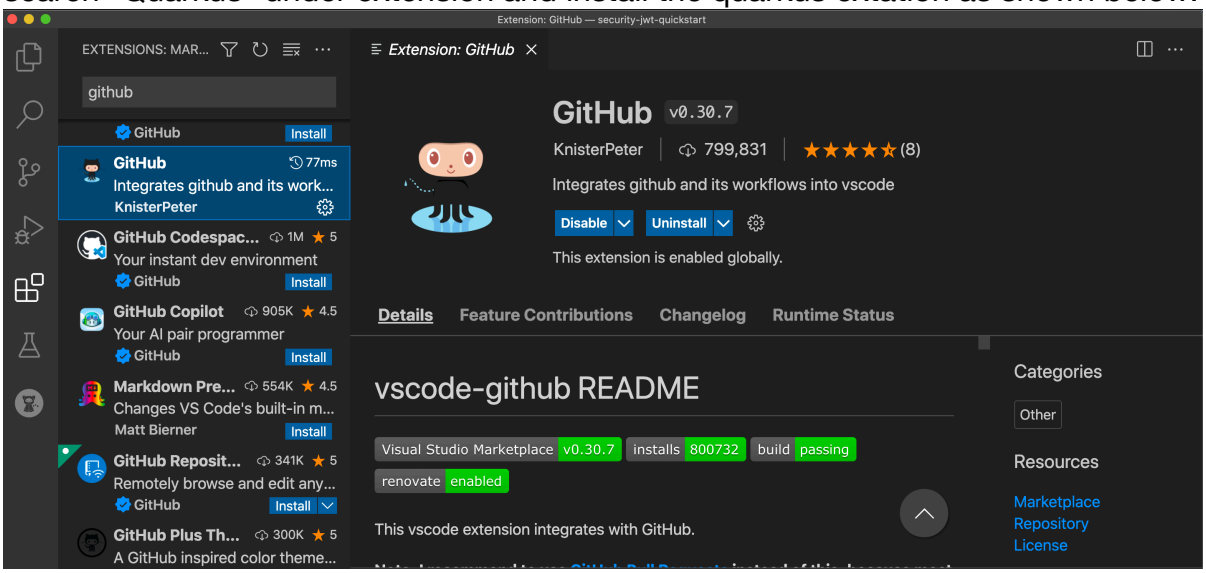


Install Quarkus Tools for Visual Studio Code

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

Install Github for Visual Studio Code

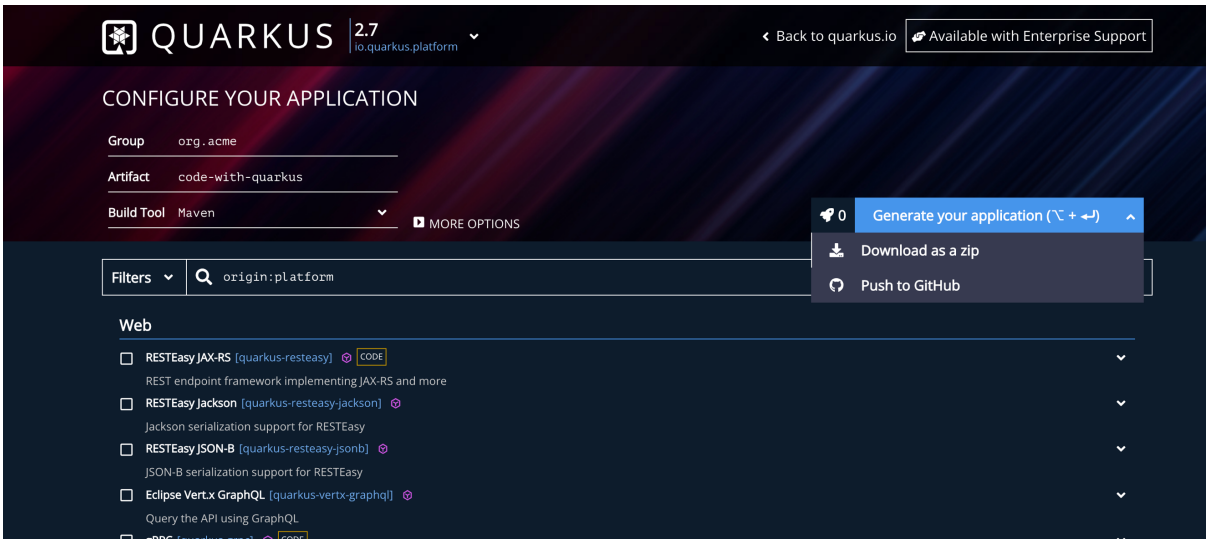
search “Quarkus” under extension and install the quarkus extation as shown below.



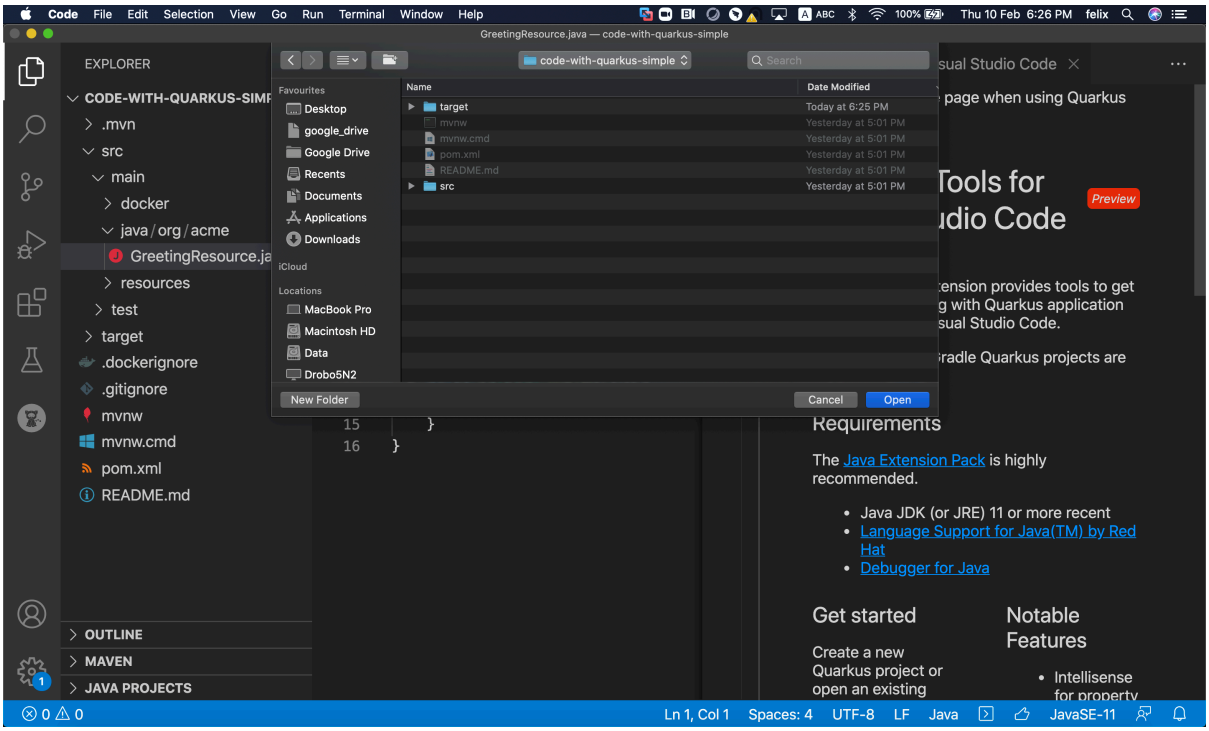
Lab0 Quarkus Sample Project Hello World

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

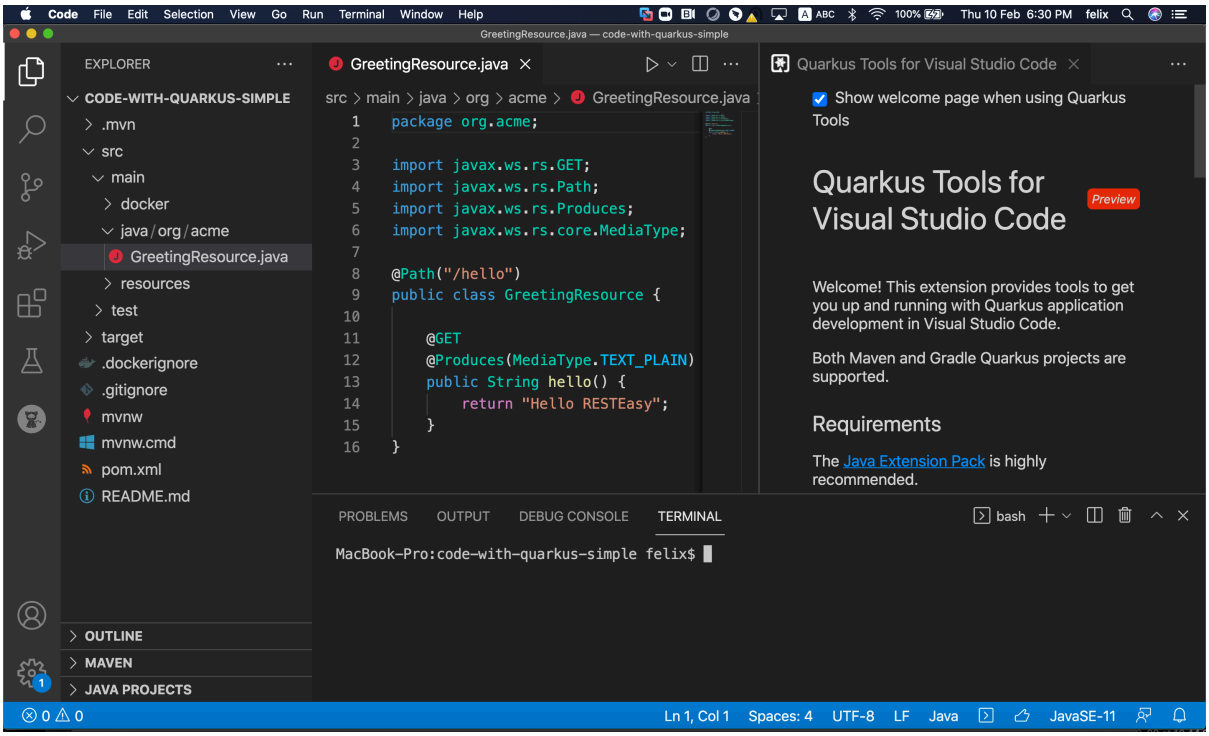
download zip and unzip to working folder



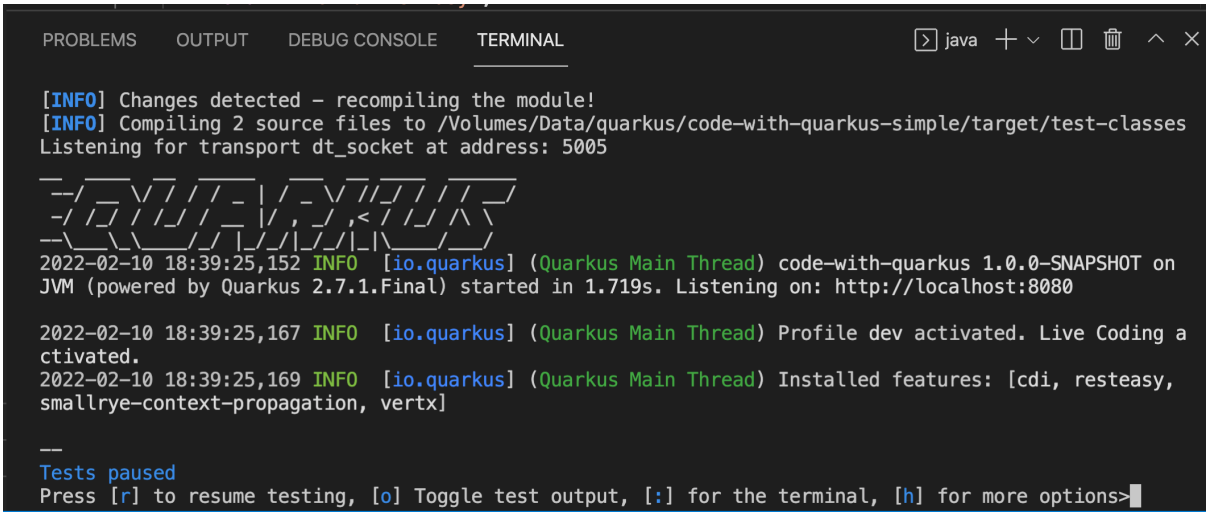
Open visual Studio Code IDE and open the folder



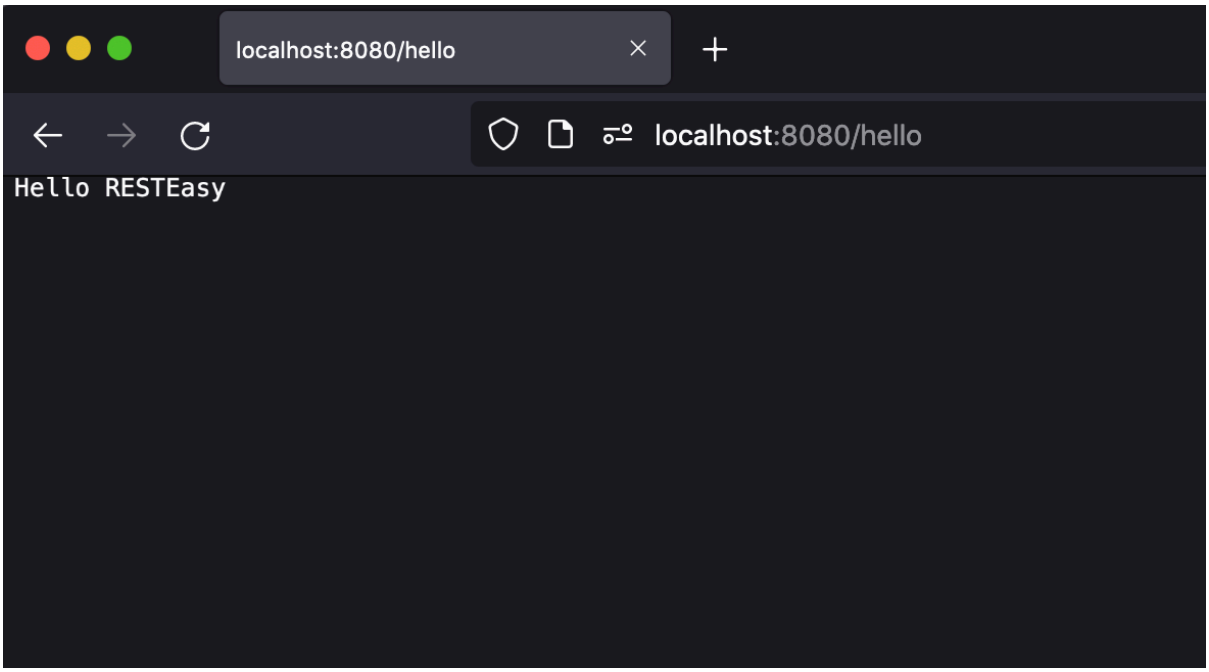
Open Terminal under the menu bar.



Run the command “mvn quarkus:dev”



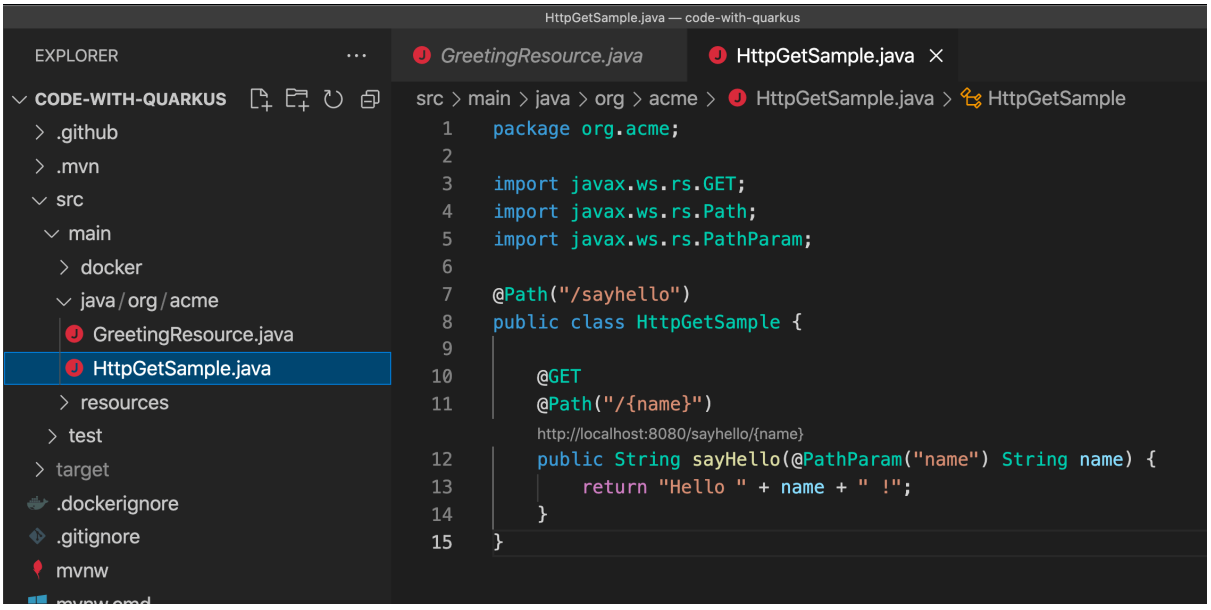
Test with browser : http://localhost:8080/hello



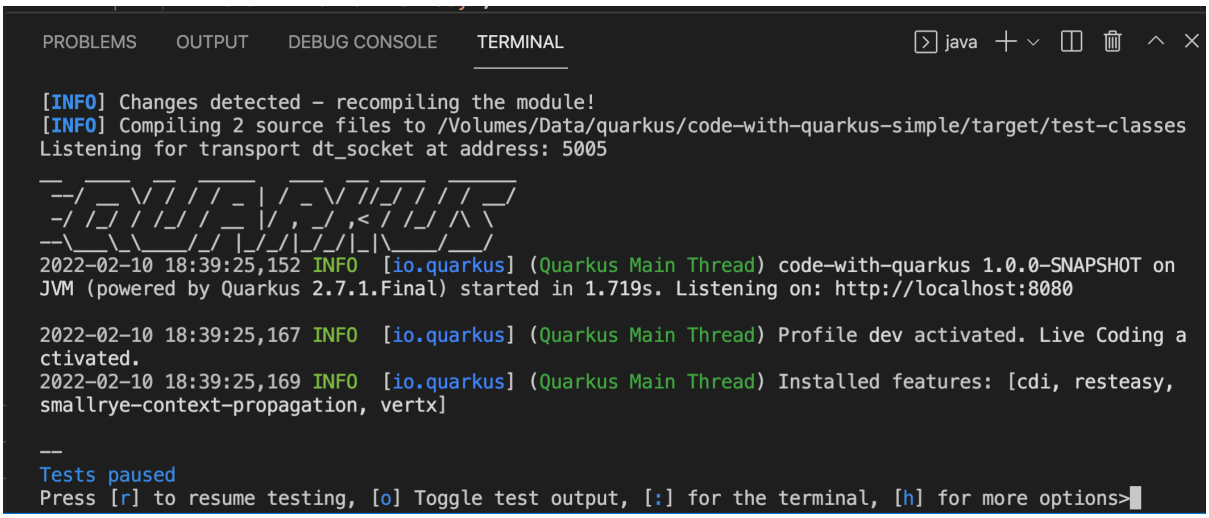
LAB1 URL Parameter

Create an other class named “HttpGetSample”

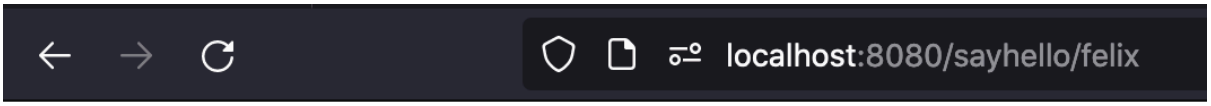
Use PathParam mapper to retrieve the URL parameter



Run mvn quarkus:dev



Test from Brower locally

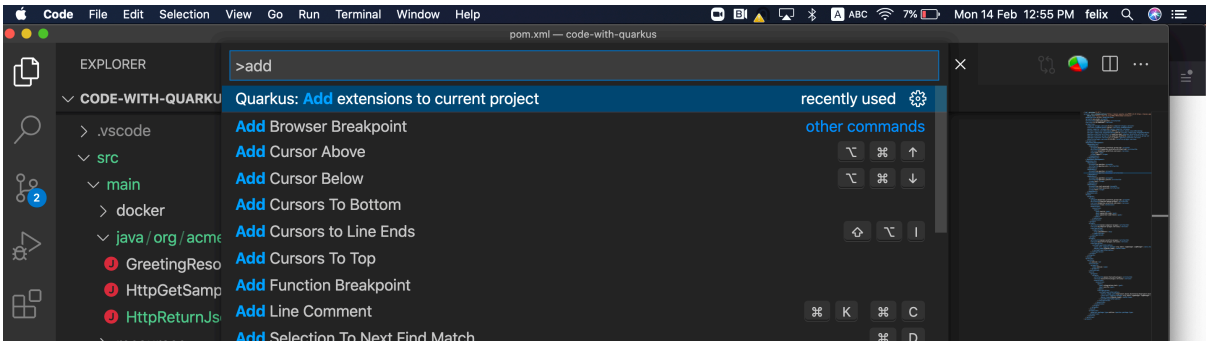


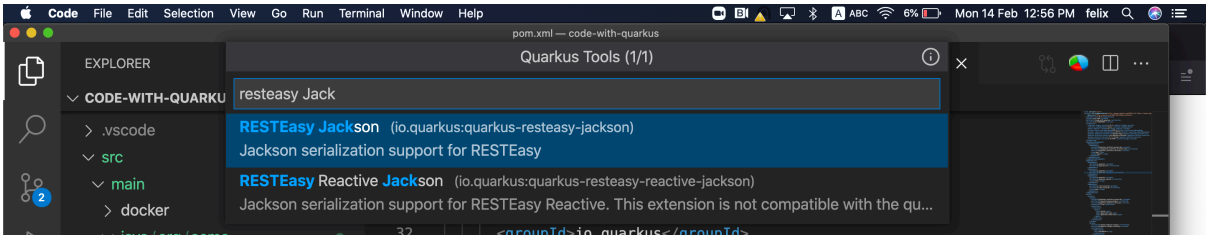
Hello felix !

LAB2 Response with JSON Obj

Importing the RESTEasy/JAX-RS and Jackson extensions

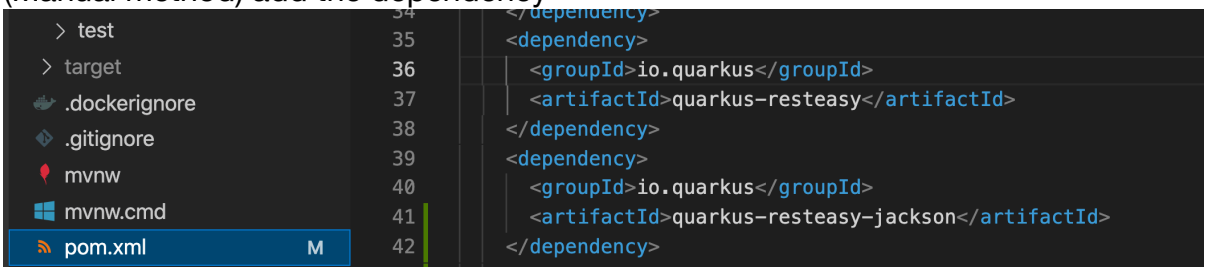
Click “View” > “Command Palette” , type “add” and choose “Quarkus: Add extensions to current project”





and press enter to import the dependency.

Or
(Manual method) add the dependency



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.

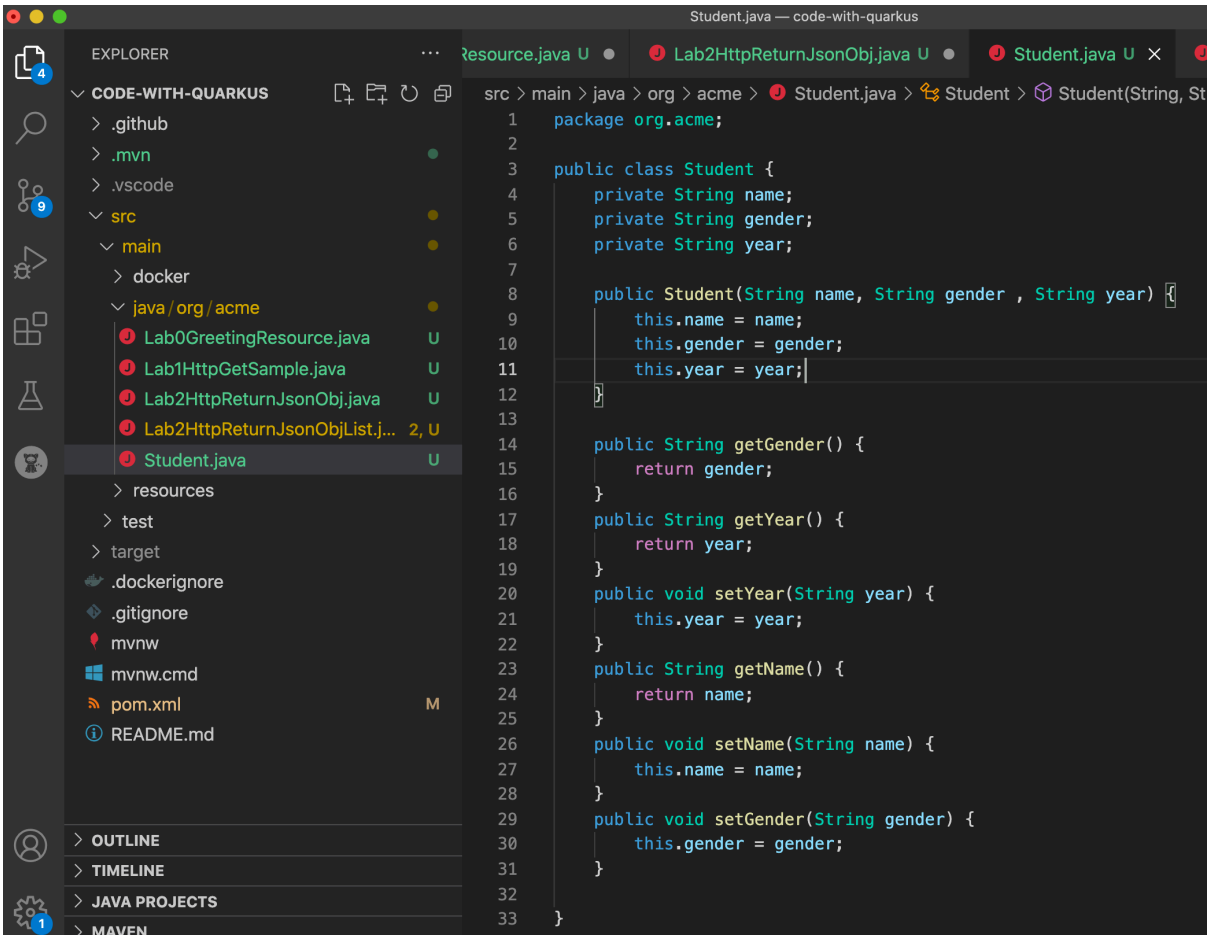
Student.java

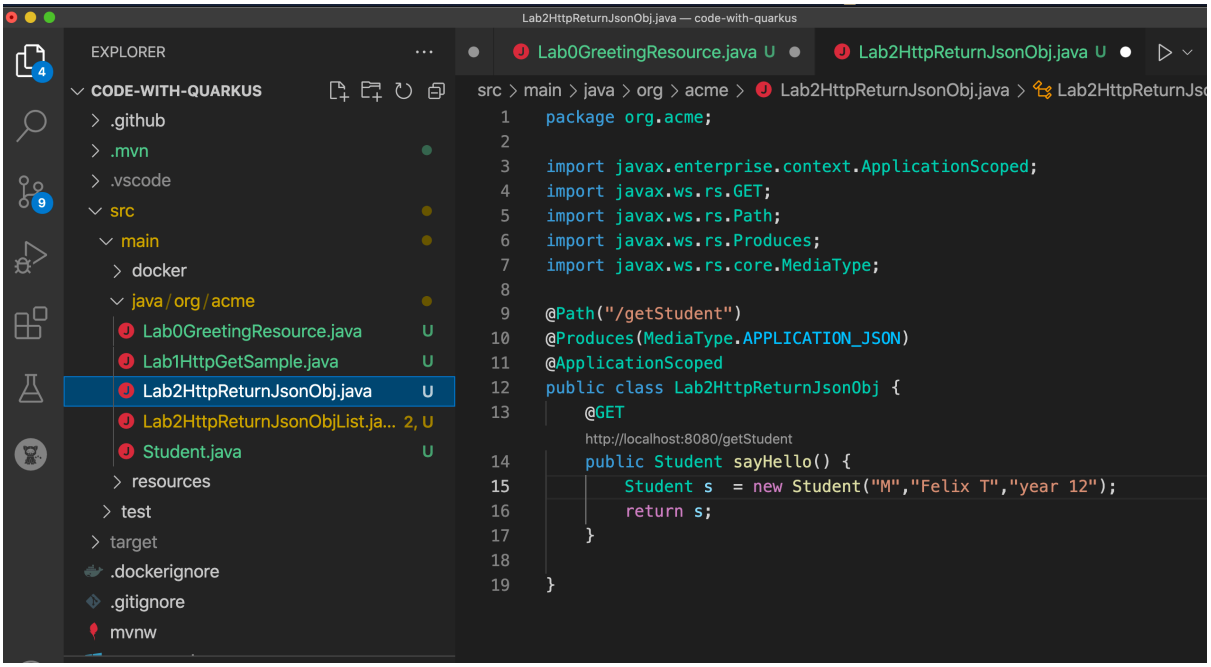
```
package org.acme;

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;
    }

}
```

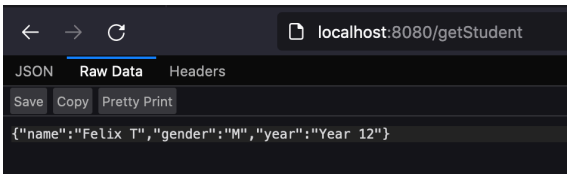
Lab2HttpReturnJsonObj.java

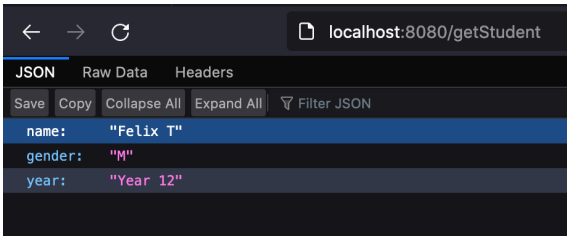
```
package org.acme;

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;
    }

}
```





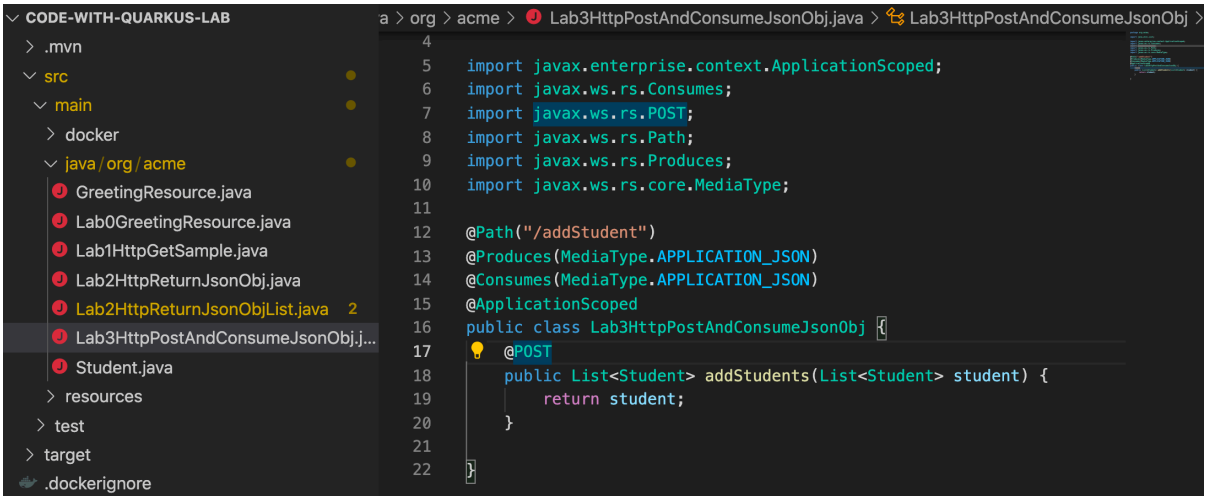
Test the JSON response


localhost:8080/getStudent
JSON
Raw Data
Headers
Save
Copy
Pretty Print
{"name":"Felix T","gender":"M","year":"Year 12"}

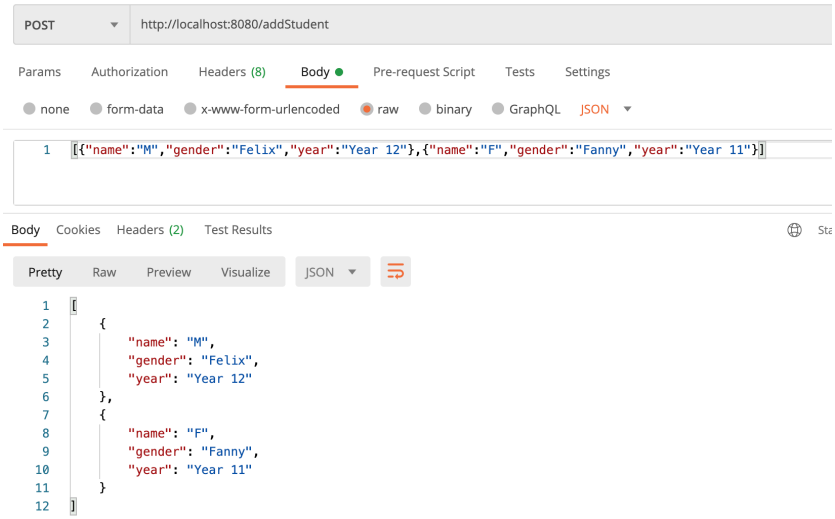

localhost:8080/getStudent
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
name: "Felix T"
gender: "M"
year: "Year 12"

LAB3 HTTP Post Consume JSON Obj

Create “Lab3HttpPostAndConsumeJsonObj.java” and add the consume annotation



Eg : POST Student Obj and convert to Java Student Obj




```
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`
`resteasy`
`resteasy-jackson`

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

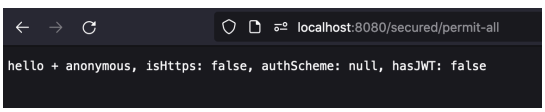
Path /secured/helloadmin, to allow the role “admin” to access the function

`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;
```

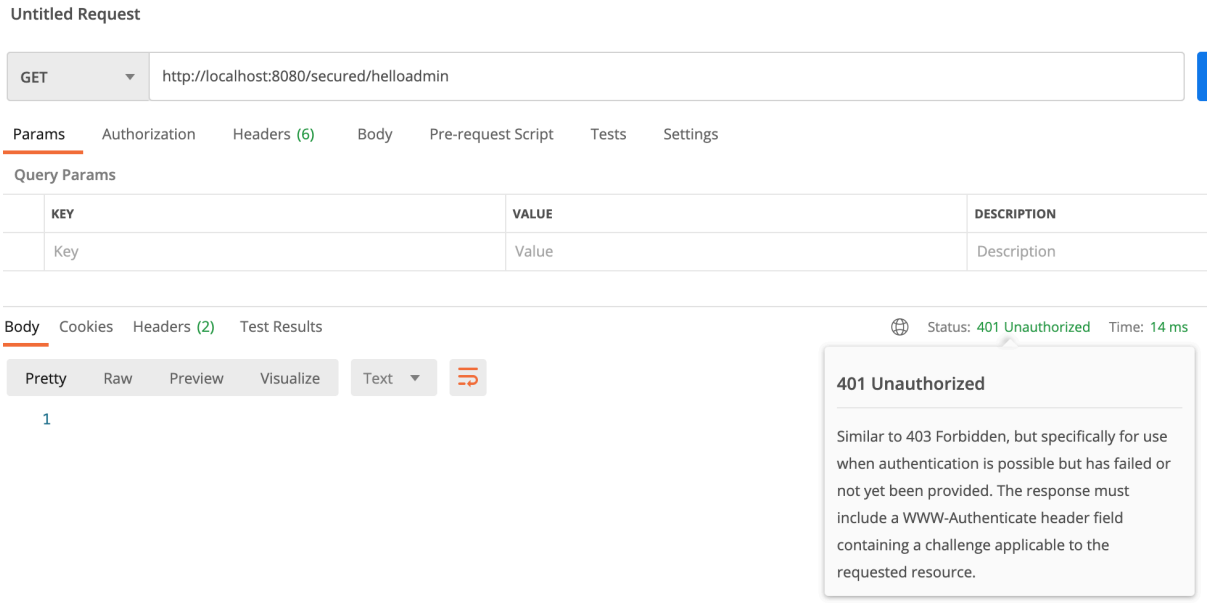
```
@Path("/secured")
@RequestScoped
public class TokenSecuredResource {

    @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("helloadmin")
    @RolesAllowed({ "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloadmin(@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



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://cryptotools.net/rsagen

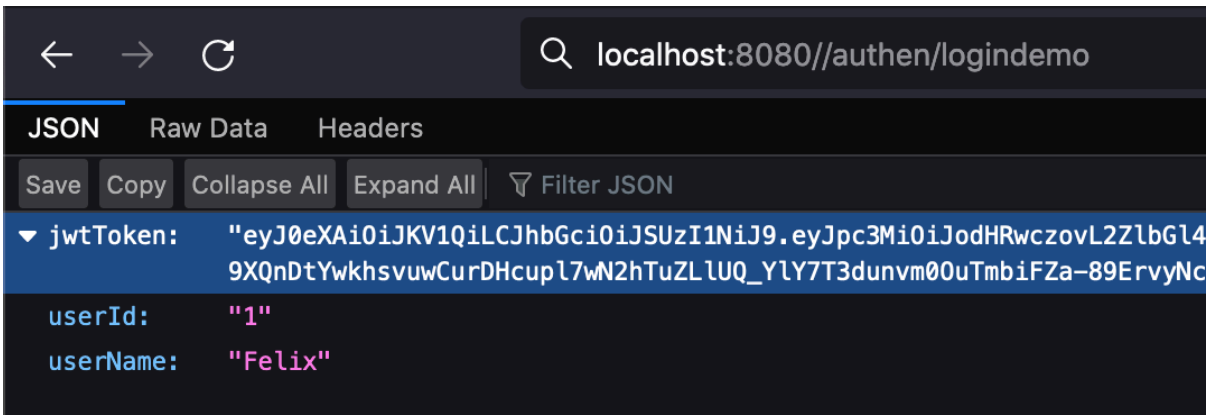
application.properties

```
mp.jwt.verify.publickey.location=publicKey.pem
mp.jwt.verify.issuer=https://felixtsang.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/

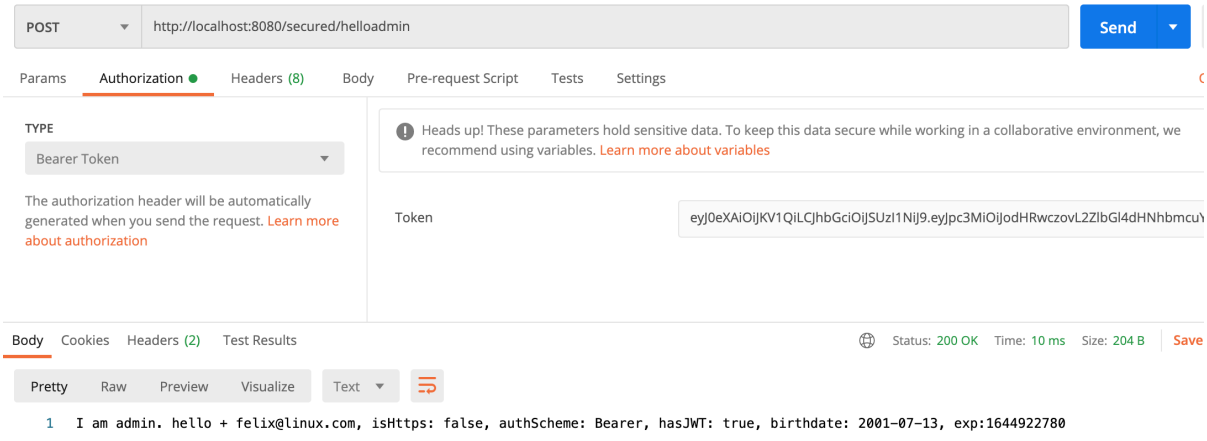
application.properties
privateKey.pem
publicKey.pem
rsaPrivateKey.pem

1. generate the jwt token
   localhost:8080//authen/logindemo



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



raw curl like the following:

```
curl --location --request POST 'http://localhost:8080/secured/helloadmin' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lz
c3VlciIsInVwbiI6ImZlbGl4QGxpbnV4LmNvbSIsInByZWZlcnJlZF91c2VybmFtZSI6ImZlbGl4Iiwib
mFtZSI6Im5hdGhhbiIsImdyb3VwcyI6WyJVc2VyIiwiQWRtaW4iXSwiYmlydGhkYXRlIjoiMjAwMS0wNy
0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ0OTIyNzgwLCJqdGkiOiI
5MmVhZDc4Yi04MzRmLTQyZGYtODNiOC04MDJlMmM3NmJkYTcifQ.t3RWRF7y6wwt--
9XQnDtYwkhsvuwCurDHcupl7wN2hTuZLlUQ_YlY7T3dunvm0OuTmbiFZa-
89ErvyNcE_vfSwalz6mwJs4gToKE7zQNWg-N_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-
5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNEtqd_R_14Feo_n8NV98-
j3A3SjNdlj7zY6xo4_rw7cpkoAkalCSZAtcJDbizT-
2ssCzSPc6BeNFTm4GM2zA3DPB7EOPucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-
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.4q0rf79n9o0ma02s --discovery-token-ca-cert-hash sha256:7bc665462e22c1c900998
2bdb2c199ef1928da324c8606fc84cf630320f46fca
root@k8smastertemp:~#
```

6. Run the join command generated from step 5 .

```
root@k8sworkertemp:~# kubeadm join 192.168.205.172:6443 --token c6w74t.4q0rf79n9o0ma02s --discovery-token-ca-cert-hash sha256:
7bc665462e22c1c9009982bdb2c199ef1928da324c8606fc84cf630320f46fca
```

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

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

9. dfdfdf

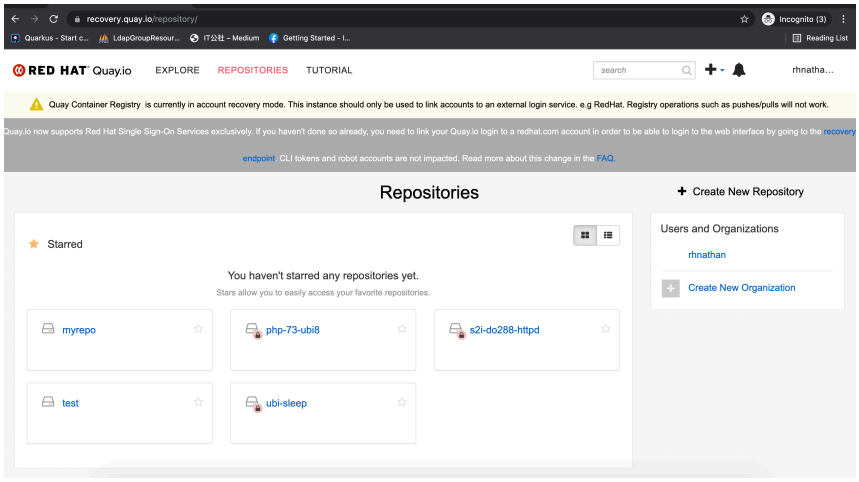
Lab 6 Create Quay.io registry Account

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

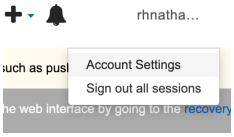
1. **Registe a RedHat ID first**
   https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/auth?response_type=code&client_id=https%3A%2F%2Fwww.redhat.com%2Fwapps%2Fugc-oidc&redirect_uri=https%3A%2F%2Fwww.redhat.com%2Fwapps%2Fugc%2Fsso%2Flogin&state=0ee9d56f-91d8-

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



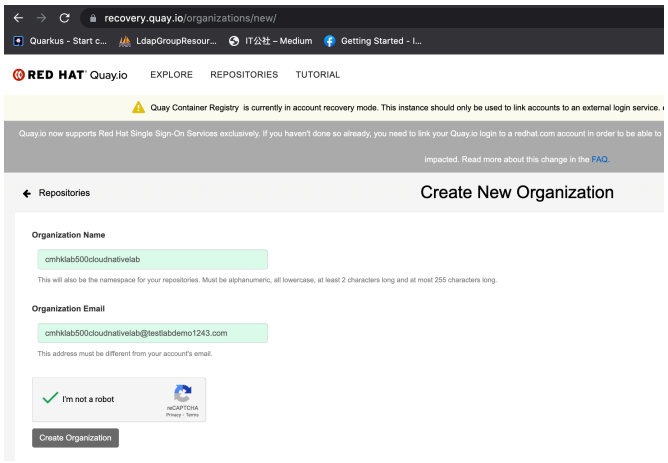
Click “Generate Encrypted Password” to get the access token

copy it and save in your local machine.

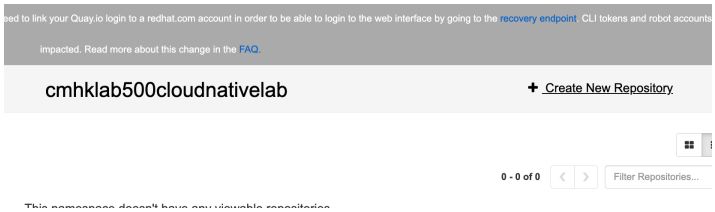
4. **Create New Organization**
   Name “cloudnativelab” + your prefix

   cmhklab{$}cloudnativelab

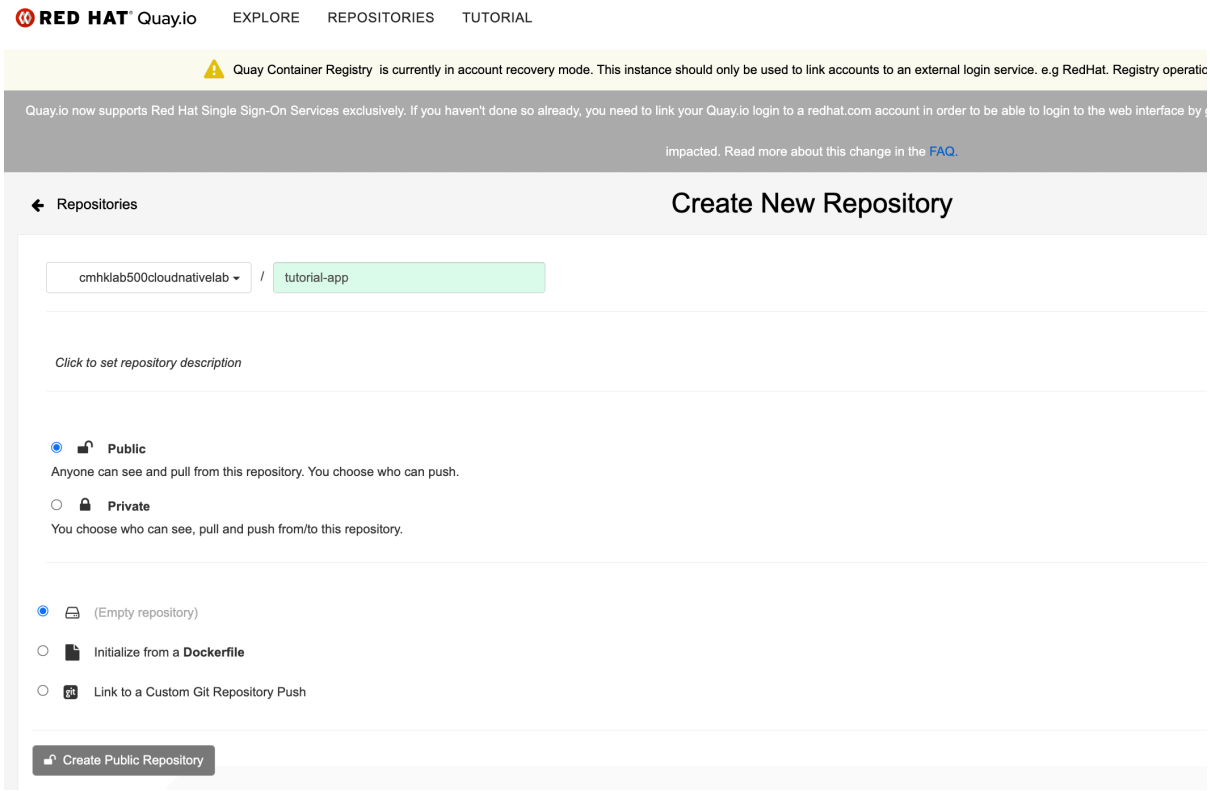
   eg ; cmhklab1cloudnativelab
   eg ; cmhklab2cloudnativelab
   eg ; cmhklab3cloudnativelab



Create



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



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

anble 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:
Login Succeeded!
MacBook-Pro:~ felix$
```

5. Install Quarkus Kubernetes Extension

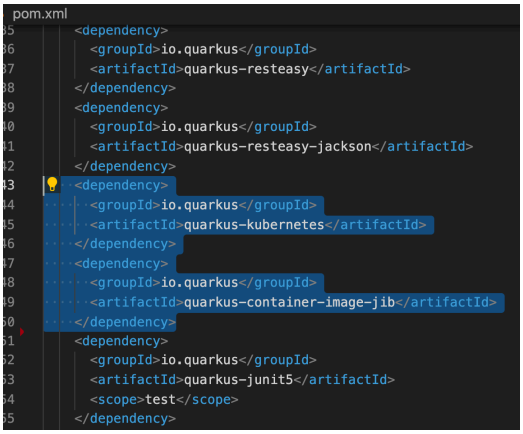
We use JIB to build the image , which not like docker need the daemon traditionally.

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>
```

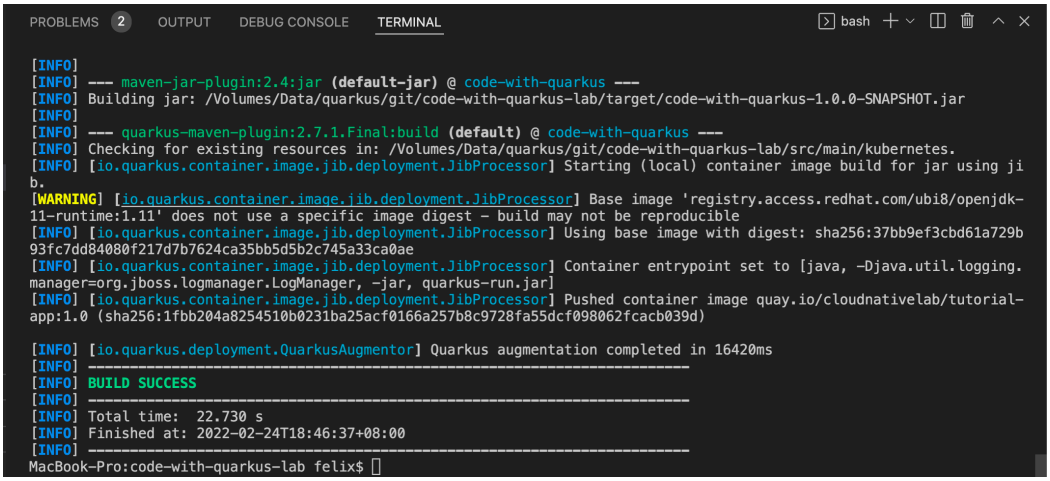


edit the “application.properties” under resources folder.Setting up the kubernetes parameters and registry information.

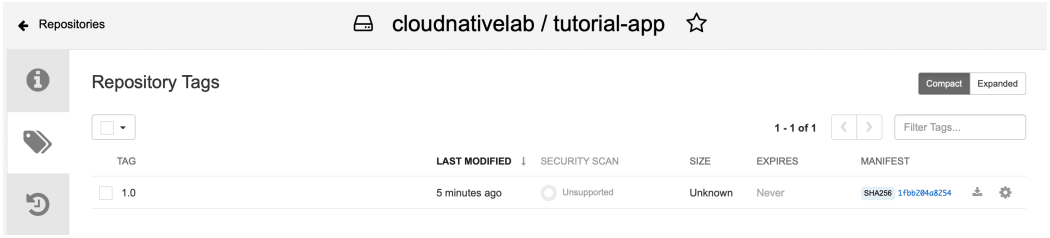
```
# Configuration file
# key = value
# tutorial https://redhat-developer-demos.github.io/quarkus-tutorial/quarkus-tutorial/kubernetes.html
quarkus.container-image.registry=quay.io
#group = organization name
quarkus.container-image.group=cloudnativelab
quarkus.container-image.name=tutorial-app
quarkus.container-image.tag=1.0
#quarkus.kubernetes.service-type=load-balancer
quarkus.kubernetes.service-type=node-port
quarkus.container-image.username= rhnathan
quarkus.container-image.password=WxF4nj12AUSZLxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

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

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

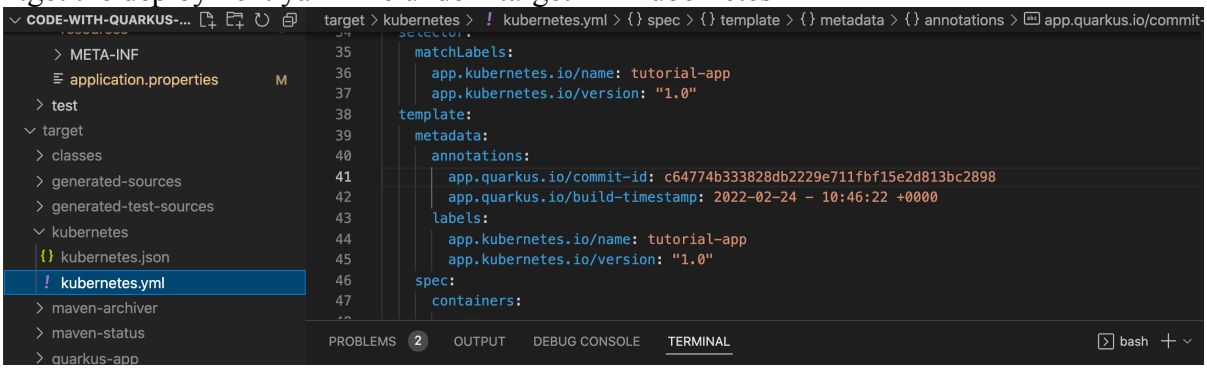


And login back to quay.io and see the image was created under the organization

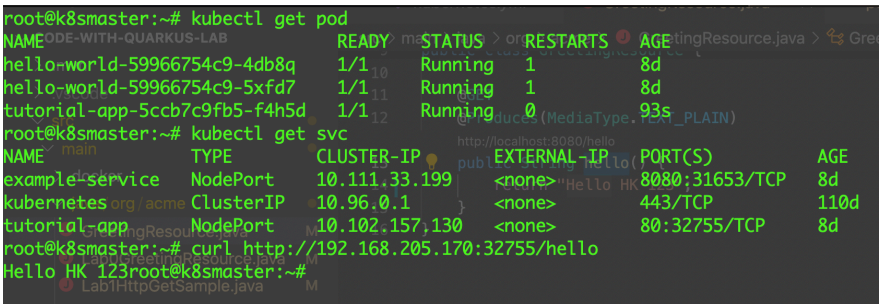


Lab 8 Deploy the app on kubernetes cluster

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



copy the kubernetes.yml to k8smater and run
kubectl apply –f kubernetes.yml



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/