Maarten Smeets

Rosanna Denis

**Workshop Graal VM & Quarkus**

# Introduction

The purpose of the workshop is to illustrate some of the powerful features of GraalVM in combination with Quarkus. ‘A Kubernetes Native Java stack tailored for GraalVM & OpenJDK HotSpot, crafted from the best of breed Java libraries and standards’. The workshop will focus on using GraalVM to generate native images, running JVM and native images in Docker containers and deploying to Kubernetes.

This workshop consists of several steps.

* Preparations  
  Setting up your environment. Import or generate the workshop VM
* Your first Quarkus application  
  Importing a Quarkus application in Eclipse and running it in development mode  
  Illustrates the live reload functionality
* Native image generation  
  Generate a native image using a locally installed GraalVM and from a prepared Docker container  
  Illustrates start-up time and memory usage differences between JIT and AOT
* Docker  
  Run Quarkus applications (native and JVM) in a Docker container
* Quarkus on Kubernetes (bonus)  
  Deploy a Quarkus native image Docker container to Minikube  
  Generate Kubernetes resources and use them to deploy to Minikube

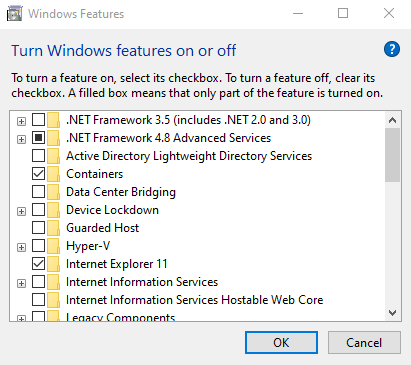
The documents and sources can be found on <https://github.com/AMIS-Services/sig-graalvm-quarkus-15082019>

# Preparations

## Install VirtualBox

It is possible to do the setup of the environment, install GraalVM, Maven, Docker Desktop and the IDE in Windows. However GraalVM on Windows is experimental and there is a (high) probability you won’t be able to get it to generate native images. In order to avoid disappointment, a Linux VM has been prepared in which things do work. This also gives everyone the same environment.

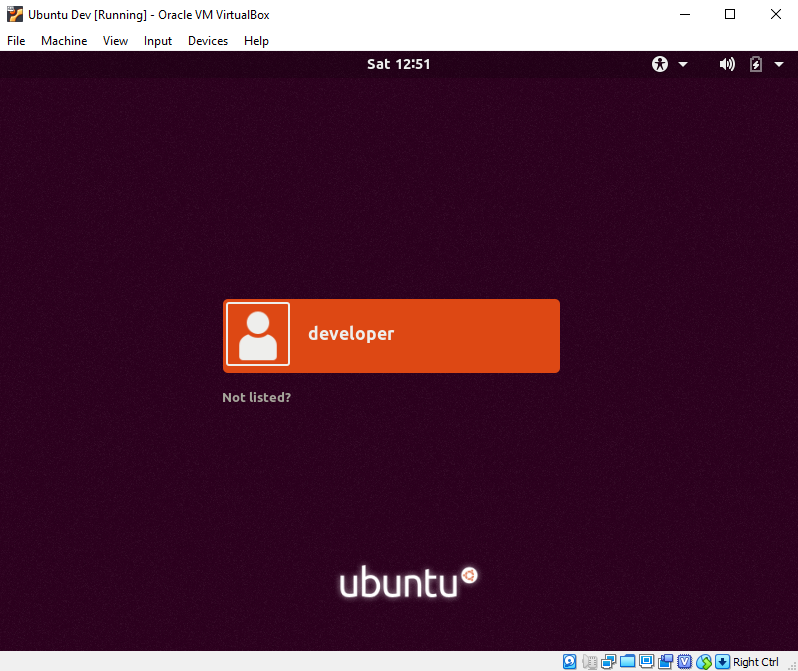
Make sure hardware virtualization support is enabled in the BIOS and make sure Hyper-V is disabled.



Download and install VirtualBox. If you already have Virtualbox installed and are importing the VM, make sure you have at least version 6.0.10 installed.

VirtualBox: <https://www.virtualbox.org/wiki/Downloads>

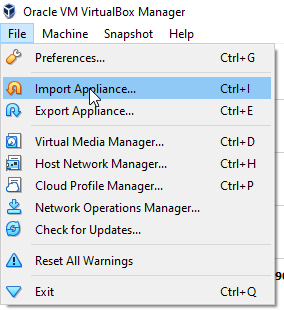
There are two ways to get started with the VM: download it and import in into VirtualBox or build it yourself using Vagrant. The second takes more time. Choose one of the options described below. After you have completed this, you can login using user developer password Welcome01



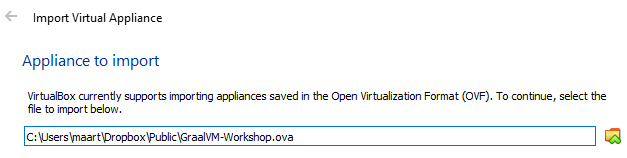
## Download and import the VM in VirtualBox

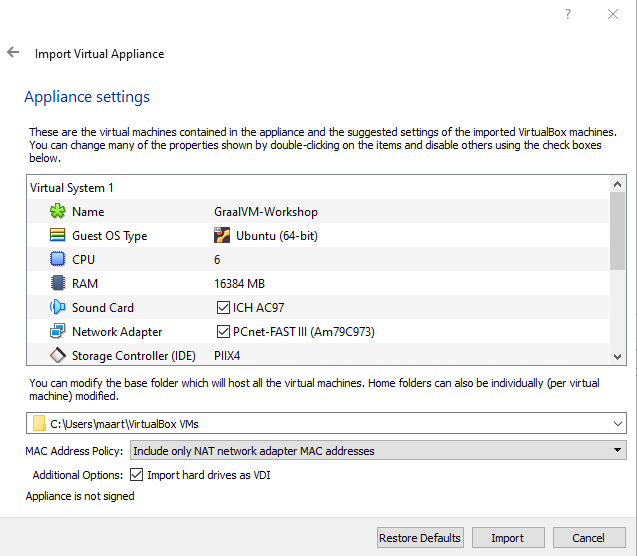
Download the appliance from <https://www.dropbox.com/s/vrecq53mdvi95kc/GraalVM-Workshop.ova?dl=0>

Start VirtualBox and import the appliance.



Select the downloaded OVA file.





Make sure to select the correct MAC address policy else your VM will not have internet access. Also check the available CPUs and RAM of your laptop and adjust the settings accordingly.

Now you can start the imported VM. DO take a look at the provision.sh script in the next section since it shows how to install GraalVM and set it as the default JVM.

## Build the VM yourself using Vagrant

If you have downloaded and imported the VM in the previous step, you should skip this step!

Download and install Vagrant: <https://www.vagrantup.com/>. If you already have Vagrant installed, make sure you have at least version 2.2.5.

Create an empty directory and put the following two files in it.

Vagrantfile   
<https://raw.githubusercontent.com/AMIS-Services/sig-graalvm-quarkus-15082019/master/provision/Vagrantfile>

provision.sh

<https://raw.githubusercontent.com/AMIS-Services/sig-graalvm-quarkus-15082019/master/provision/provision.sh>

The Vagrantfile might have to be updated slightly to reflect the available resources on your laptop (CPUs, memory, disk). The provision.sh script shows the required steps to install GraalVM, Docker, Eclipse, Minikube, Kubectl. As you can see in the provision.sh script, you can use GraalVM like any other JVM.

Start a command prompt in the directory and do:

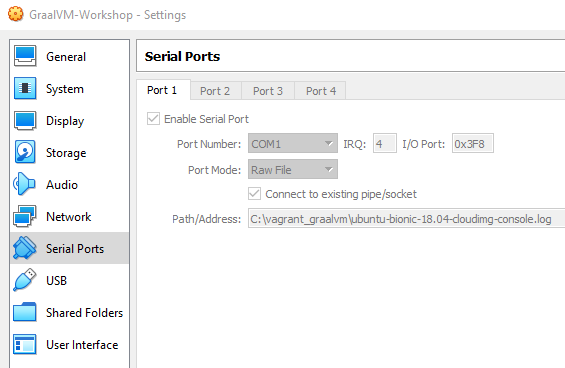
vagrant up

Now you can start the VM

## Troubleshooting

### Serial port

Should the imported or generated VM hang during boot, you might have to configure the Serial port. Make sure a serial port is available and it can send output to a file;



### Internet

Some of the steps require internet availability. One of the issues you might encounter is that the MAC address is hardcoded in the VM in the file /etc/netplan/50-cloud-init.yaml. The MAC might change due to importing the VM. The MAC address listed there should be the same as the output of: cat /sys/class/net/enp0s3/address

In order to fix this, execute the following:

sudo sed -E -i "s/([[:xdigit:]]{1,2}:){5}[[:xdigit:]]{1,2}/`cat /sys/class/net/enp0s3/address`/" /etc/netplan/50-cloud-init.yaml

sudo netplan apply

# Your first Quarkus application

Start a terminal, go to your home folder and clone the provided Git repository



cd ~

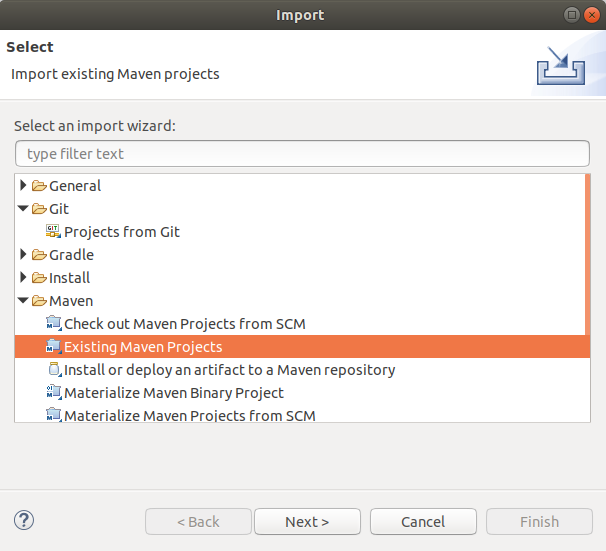
git clone https://github.com/AMIS-Services/sig-graalvm-quarkus-15082019.git

This includes the quarkus-getting-started project which can be created by following the following tutorial: <https://quarkus.io/guides/getting-started-guide>. Read this tutorial and understand what has been prepared. We’ll use it as a starting point.

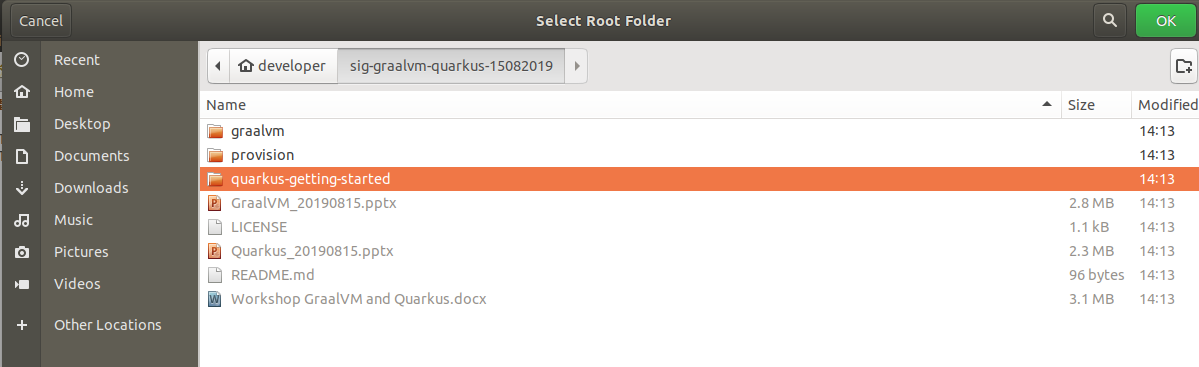
Start Eclipse

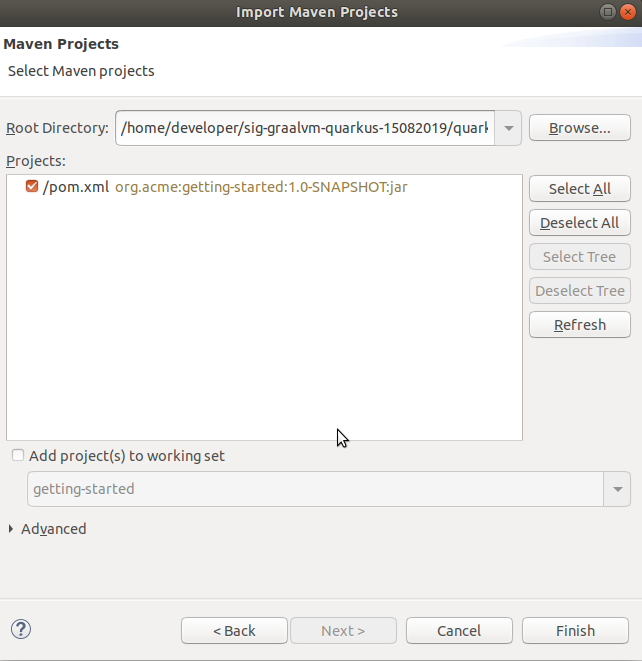


File, Import, Existing Maven projects

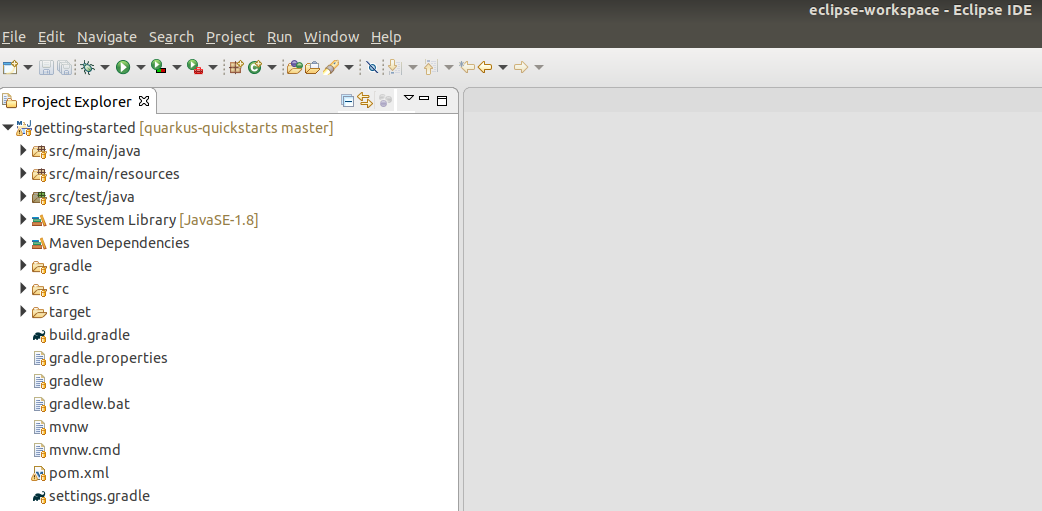


Select the quarkus-getting-started folder

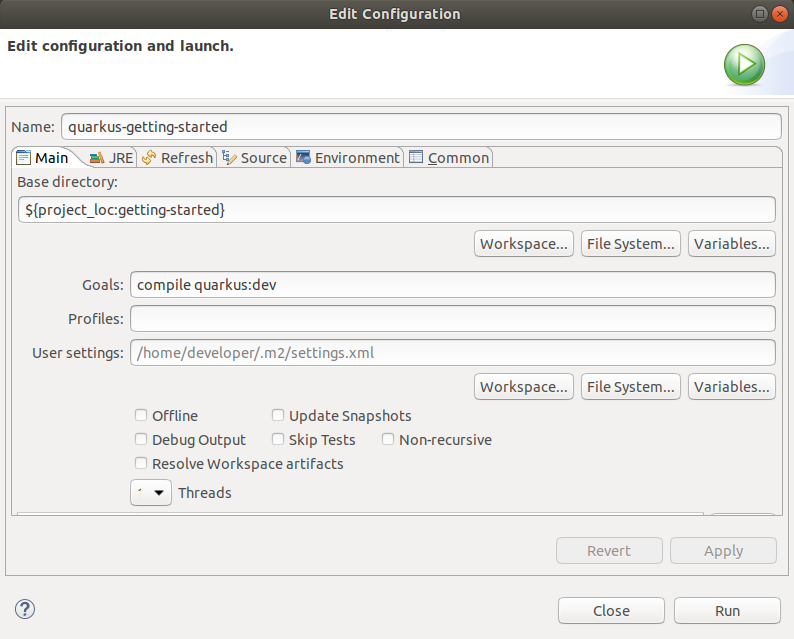




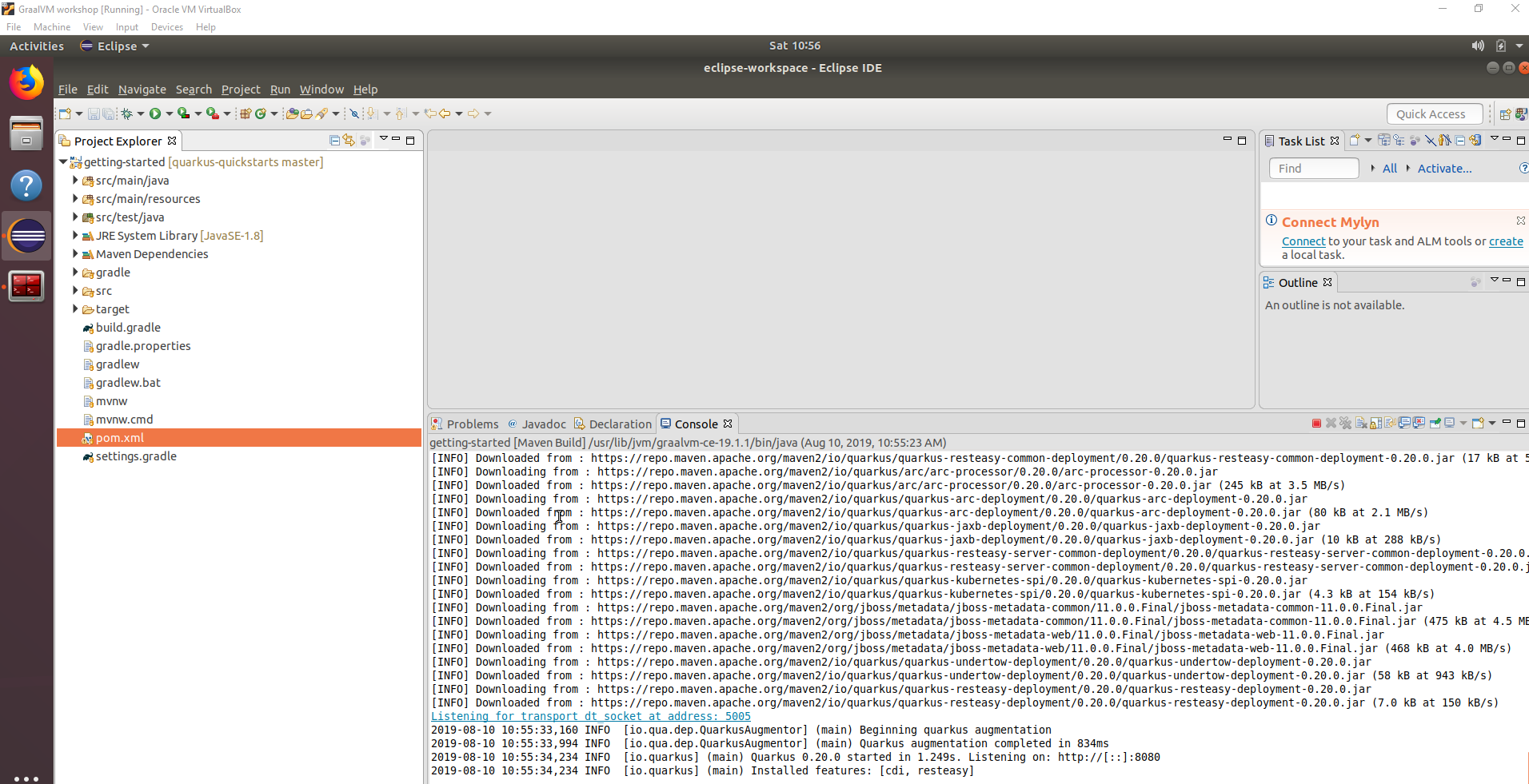
Open the Project explorer (Window, Show view, Project explorer) or use the Package explorer



Right click the pom.xml, Run As, Maven build …



Click Run and confirm the application is running:



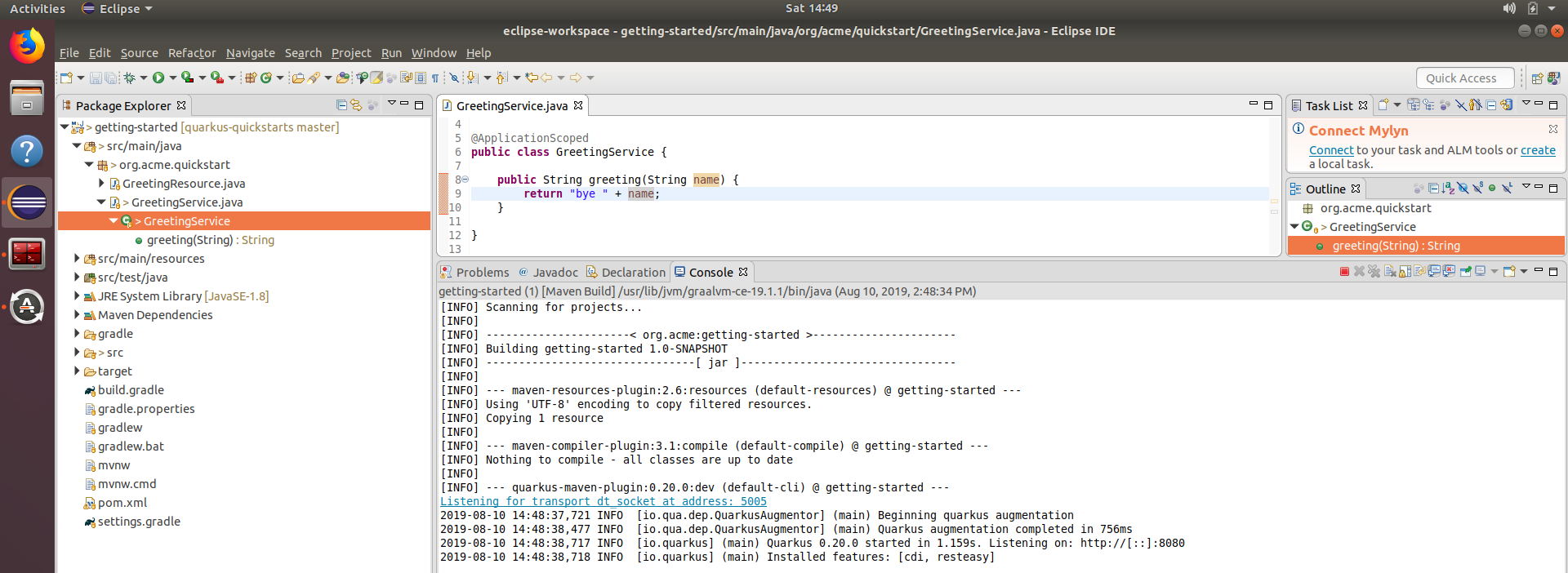
In a terminal give the following command:

curl http://localhost:8080/hello

Confirm it returns hello

## Live reload

Update the GreetingService.java and save



Notice nothing peculiar happens next to the file being saved.

Now fire off a new request

curl http://localhost:8080/hello

and check the response. Check the Console in Eclipse. What has happened?

Notice that when you do a mvn package, 2 JAR files get generated.

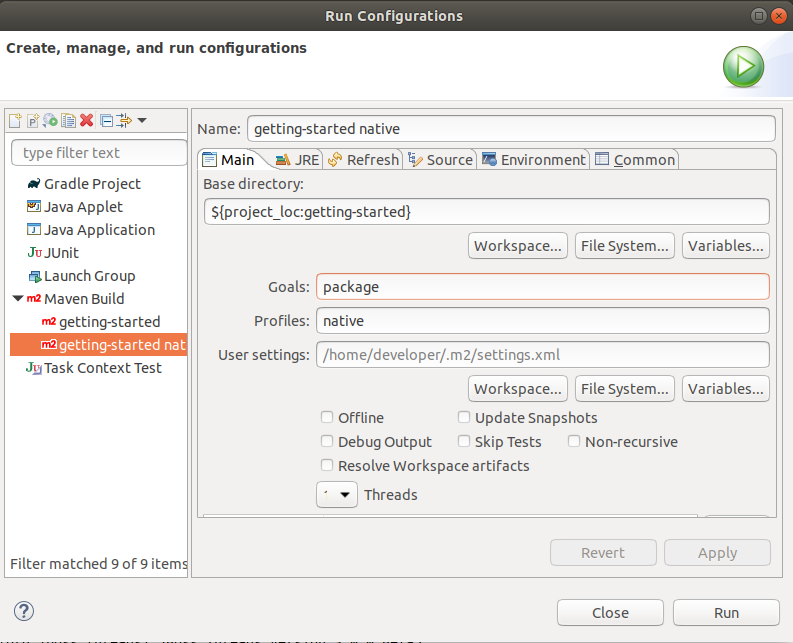


The runner has dependencies in the target/lib folder. This is the file which can be run with java -jar. The smaller JAR file contains the actual application logic. Both are not suitable to be compiled to a native image since they are not a single ‘fat jar’ containing everything which is required.

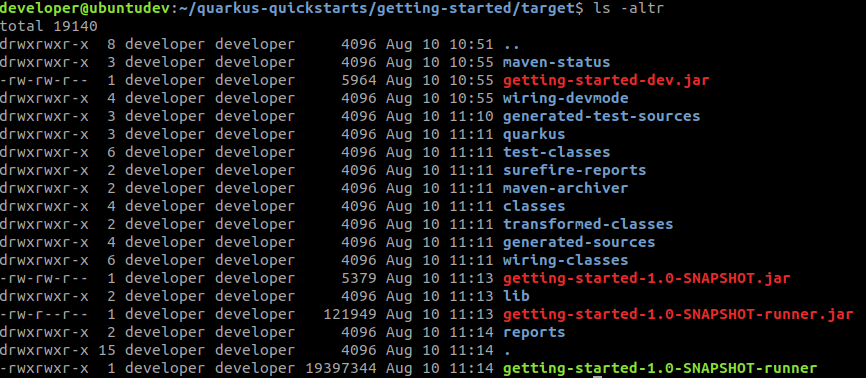
Restore the GreetingService to returning hello again or update the tests in order to avoid errors during the next steps.

# Native image generation

Create a new Maven run configuration



Confirm the native image has been created



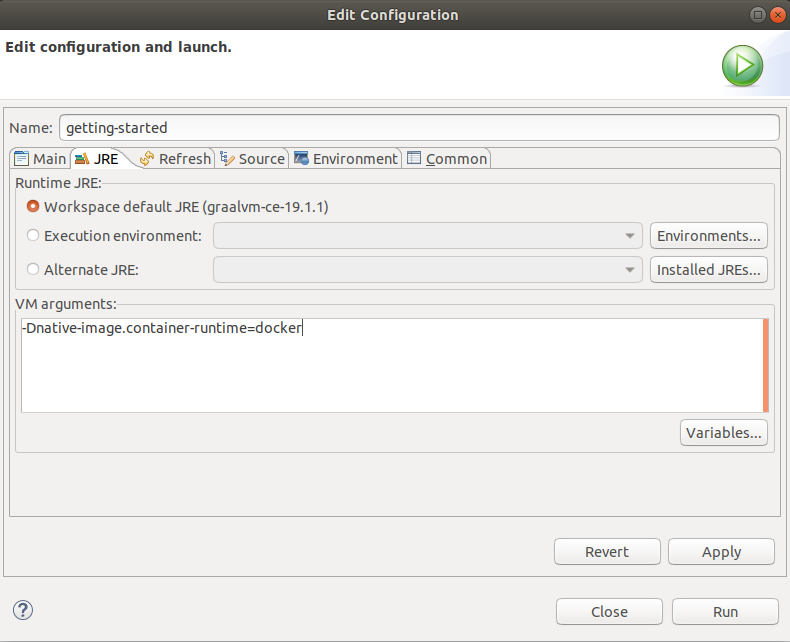
Execute it



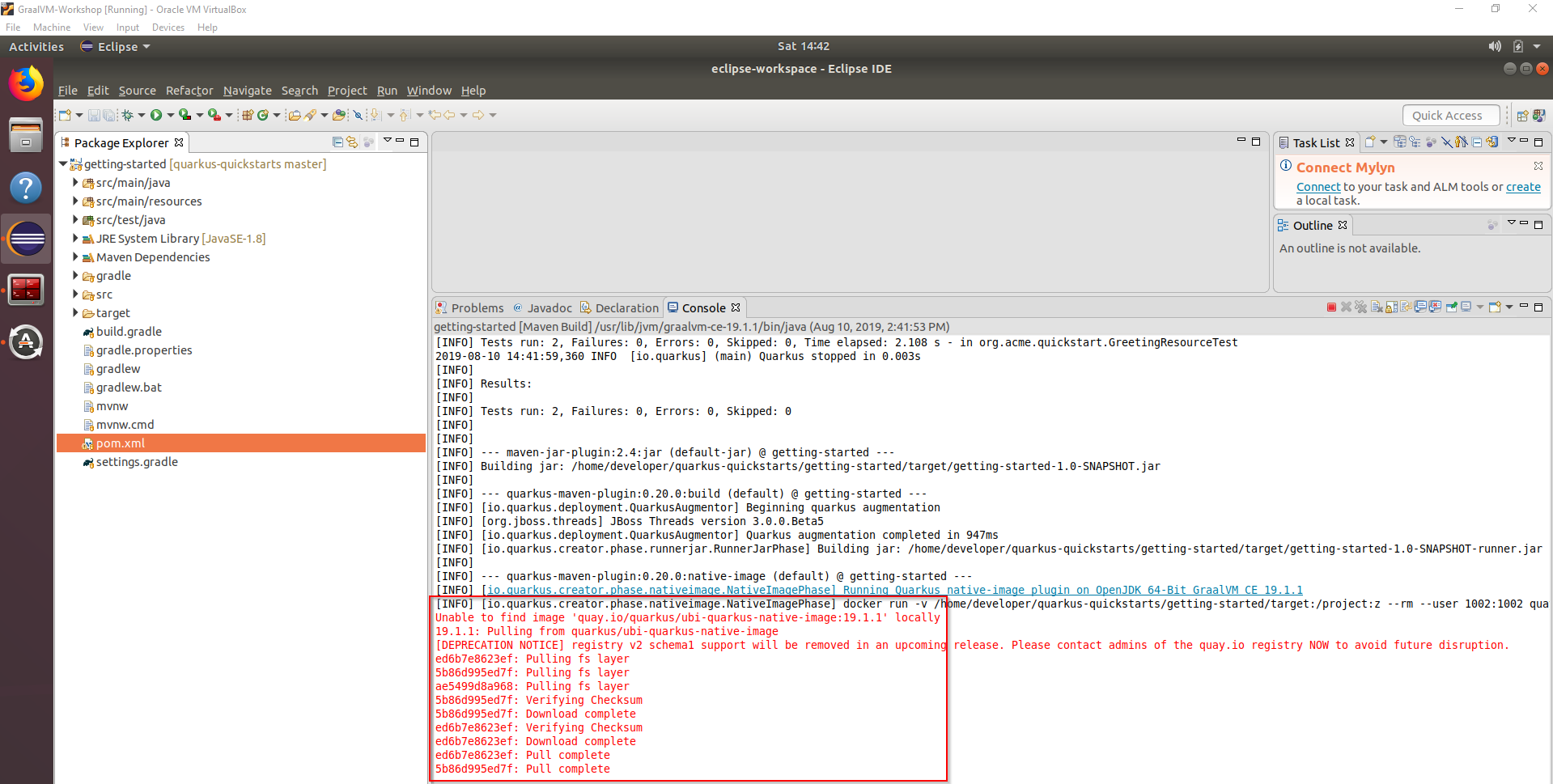
Compare the start-up time of the native executable with the start-up time of the JAR. If you do not have the JAR running anymore, rerun it. Otherwise check the screenshots in this workshop.

## Build the native image without locally installed GraalVM

Add the following JVM argument and rerun the build



What is happening here?



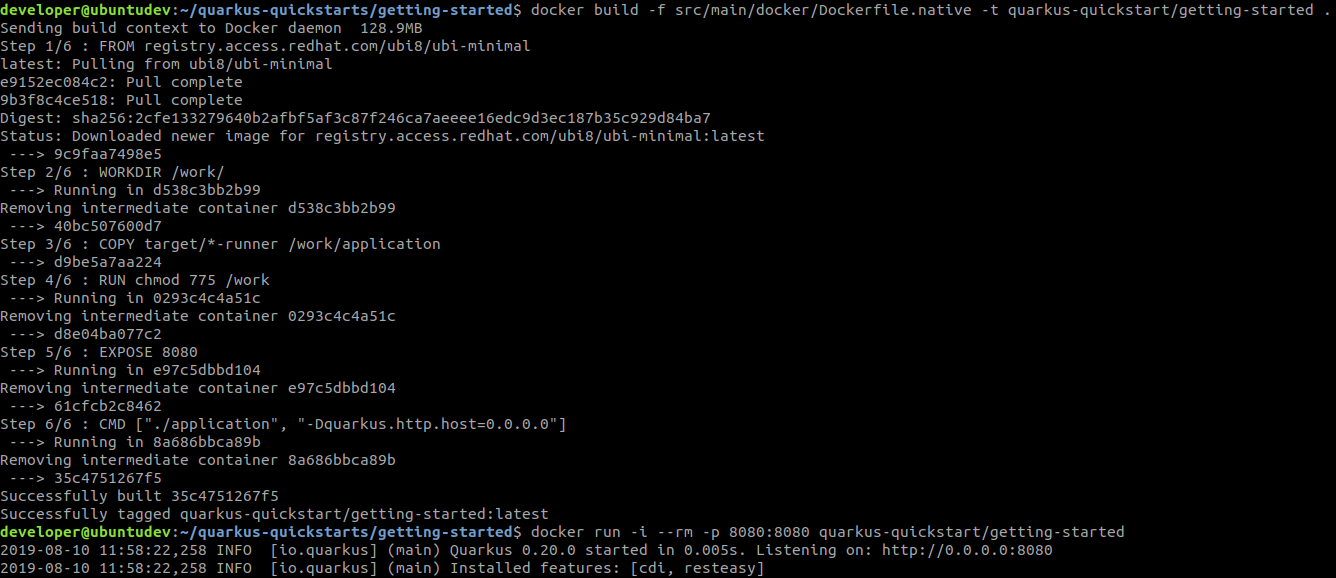
# Docker

Build and run the native image in a Docker container

cd /home/developer/sig-graalvm-quarkus-15082019/quarkus-getting-started

docker build -f src/main/docker/Dockerfile.native -t quarkus-quickstart/getting-started .

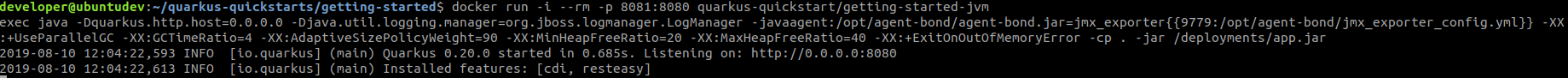
docker run -i --rm -p 8080:8080 quarkus-quickstart/getting-started



Build and run the JVM image in a Docker container

docker build -f src/main/docker/Dockerfile.jvm -t quarkus-quickstart/getting-started-jvm .

docker run -i --rm -p 8081:8080 quarkus-quickstart/getting-started-jvm



Compare the memory usage of the two containers

docker stats



## Cleanup

docker stop $(docker ps -a -q)

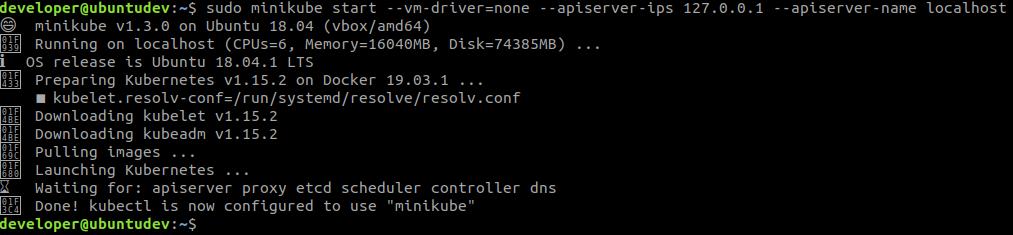
# Quarkus on Kubernetes (Bonus)

## Start minikube

sudo minikube start --vm-driver=none --apiserver-ips 127.0.0.1 --apiserver-name localhost

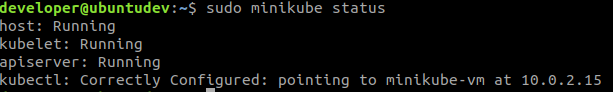
sudo chown -R developer /home/developer/.minikube

sudo chgrp -R developer /home/developer/.minikube



sudo minikube update-context

sudo minikube status

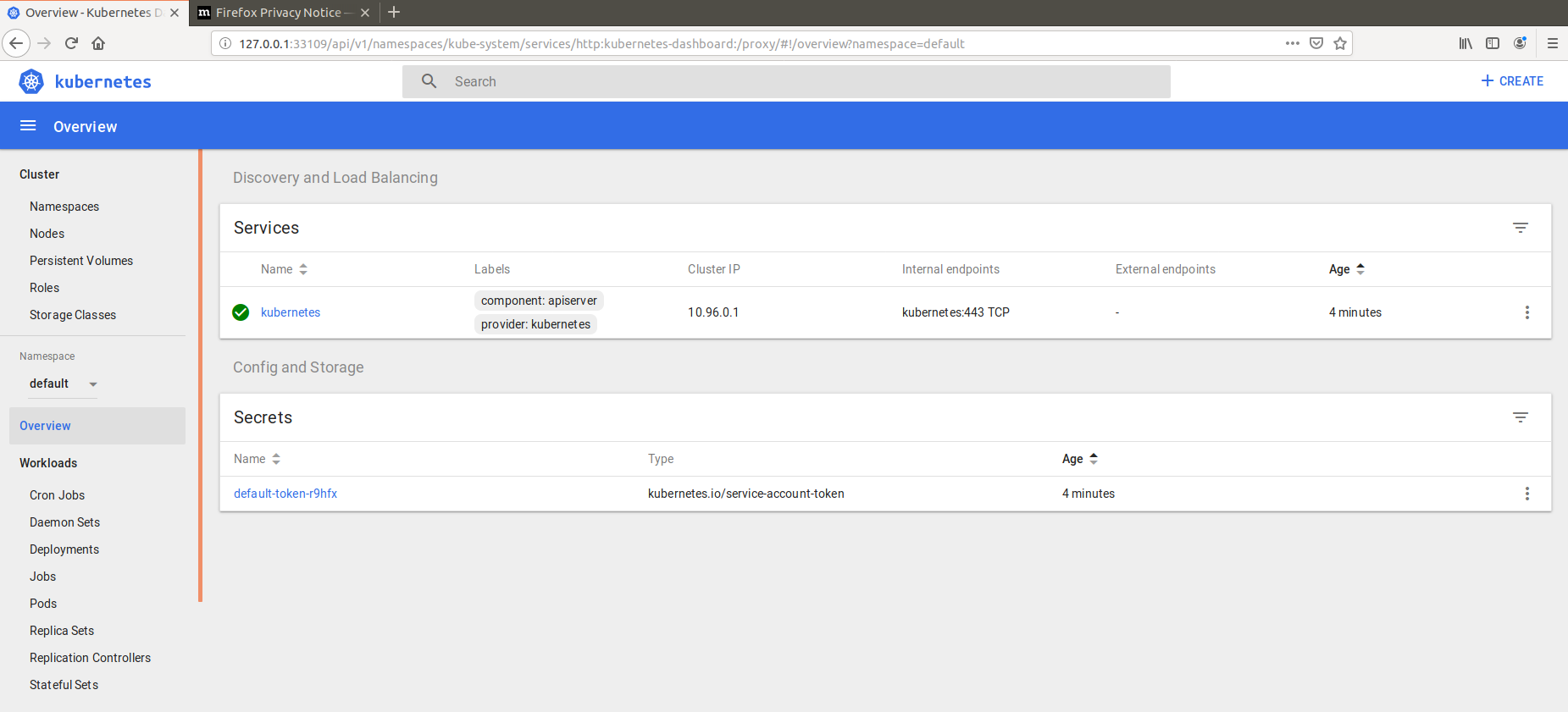


sudo minikube dashboard

This displays an URL.



Open this URL in a browser to display the Minikube dashboard



## Deploy the container to Kubernetes

Based on <https://blog.hasura.io/sharing-a-local-registry-for-minikube-37c7240d0615/>

Download and create a Minikube hosted Docker registry and expose it

wget https://gist.githubusercontent.com/coco98/b750b3debc6d517308596c248daf3bb1/raw/6efc11eb8c2dce167ba0a5e557833cc4ff38fa7c/kube-registry.yaml -O kube-registry.yaml

sudo kubectl create -f kube-registry.yaml

Change the push repository of the container

docker tag quarkus-quickstart/getting-started localhost:5000/quarkus-quickstart/getting-started

Push to the locally hosted registry (so Minikube can find it)

docker push localhost:5000/quarkus-quickstart/getting-started

Start the container

sudo kubectl run quarkus-quickstart --image=localhost:5000/quarkus-quickstart/getting-started --port=8080 --image-pull-policy=IfNotPresent

Expose the container

sudo kubectl expose deployment quarkus-quickstart --type=NodePort

Test the deployment

curl $(sudo minikube service quarkus-quickstart --url)/hello

## Generating and using Kubernetes resources (Extra)

Update the getting-started project with the Kubernetes extension. Generate resources and use these to deploy to the running Minikube instance.

Get inspiration from the following and the above example on how to do this: <https://quarkus.io/guides/kubernetes-resources>