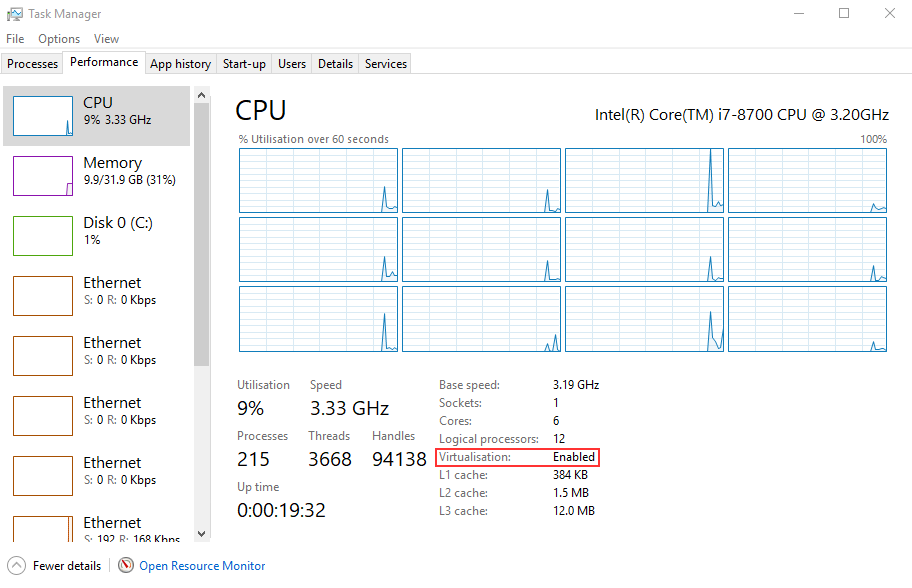
# Preparations

The manual takes a Windows 10 desktop as starting point.

## Virtualization

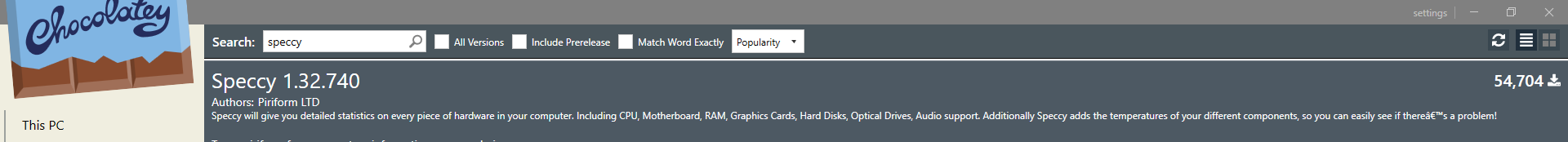
Check if virtualization is enabled. This is required for using hypervisors such as Hyper-V, VirtualBox or KVM. On Windows 10 start the Task Manager, go to the performance tab and check the Virtualization status



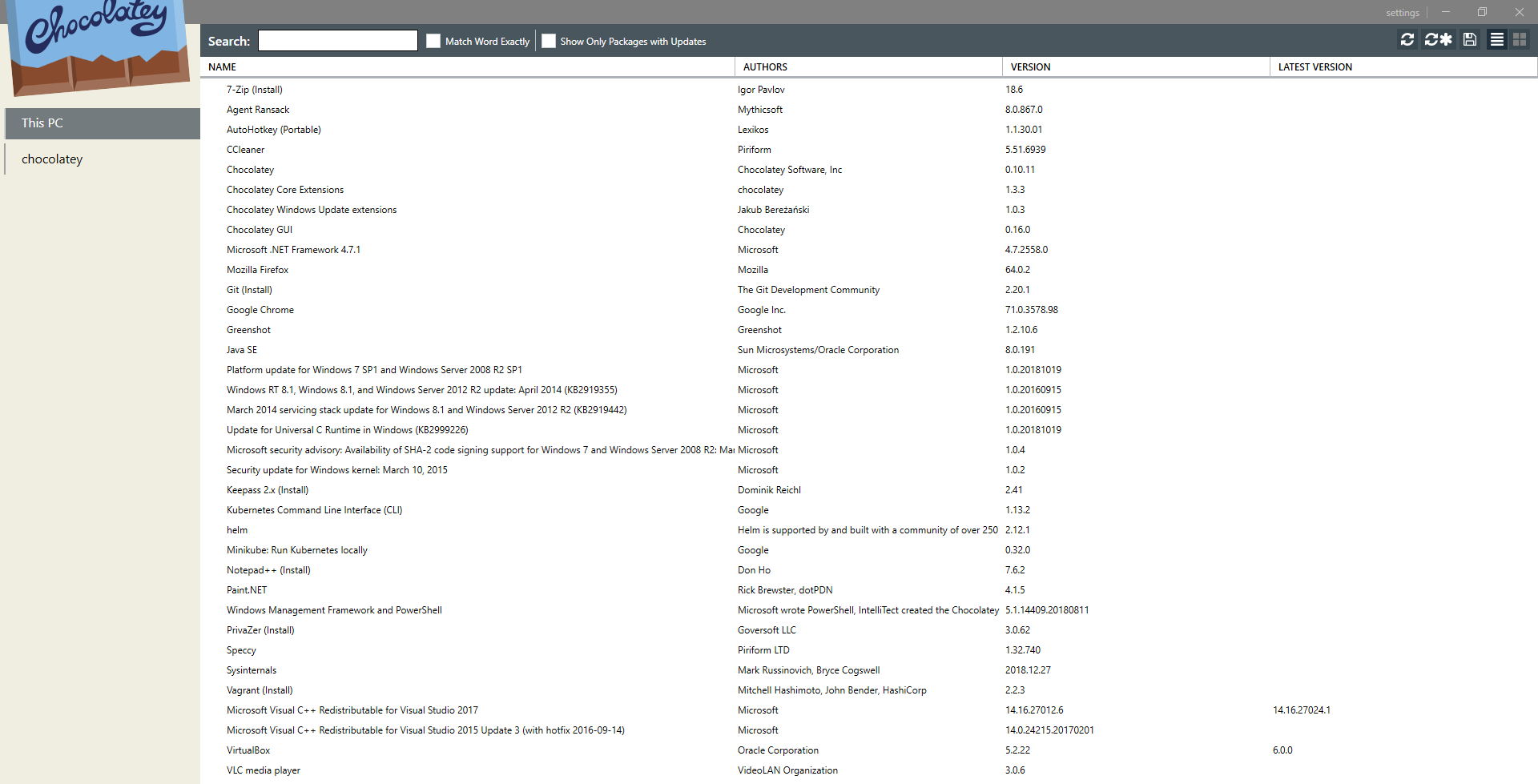
If it is not enabled, a BIOS setting needs to be changed. This differs per laptop. If you cannot change the BIOS setting yourself, contact your laptop support organisation.

## Package manager

Not required but highly recommended is to install Chocolatey (a Windows package manager): https://chocolatey.org/ . A GUI can be downloaded from: <https://chocolatey.org/packages/ChocolateyGUI>



Chocolatey makes it easy to install all kinds of different programs on a Windows machine and allows you to easily and automatically update them such as VirtualBox, Helm, Minikube but also tools like 7zip and Java.



There are two installations of Minikube which are described here.

* Windows: Using Docker for Windows and Hyper-V
* VirtualBox using an Ubuntu VM running minikube locally within that VM

# WindoWs

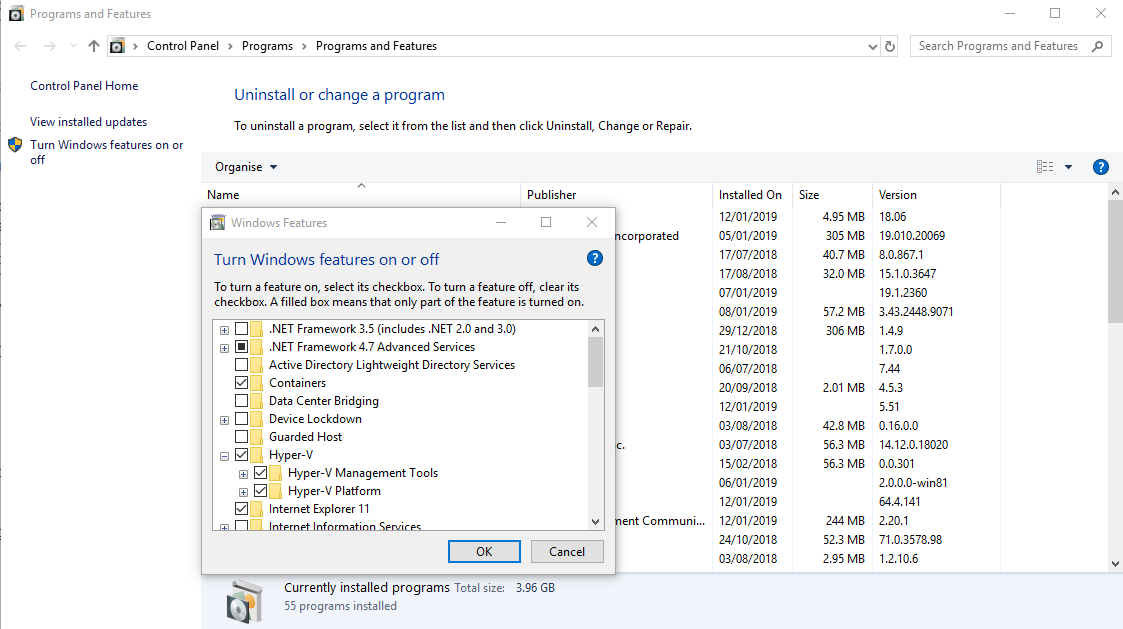
## Docker for Windows (Docker Desktop)

Install Docker for Windows

* Manually: <https://docs.docker.com/docker-for-windows/install/>
* Automated using Chocolatey

This enables Hyper-V. If you do not enable it during installation, you can do so afterwards by using the GUI:

On Windows 10: Programs and Features, Hyper-V



You can also run (as an alternative to using the GUI) from a Powershell window with Administrator rights (when the setting changes, a restart is required):

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V –All

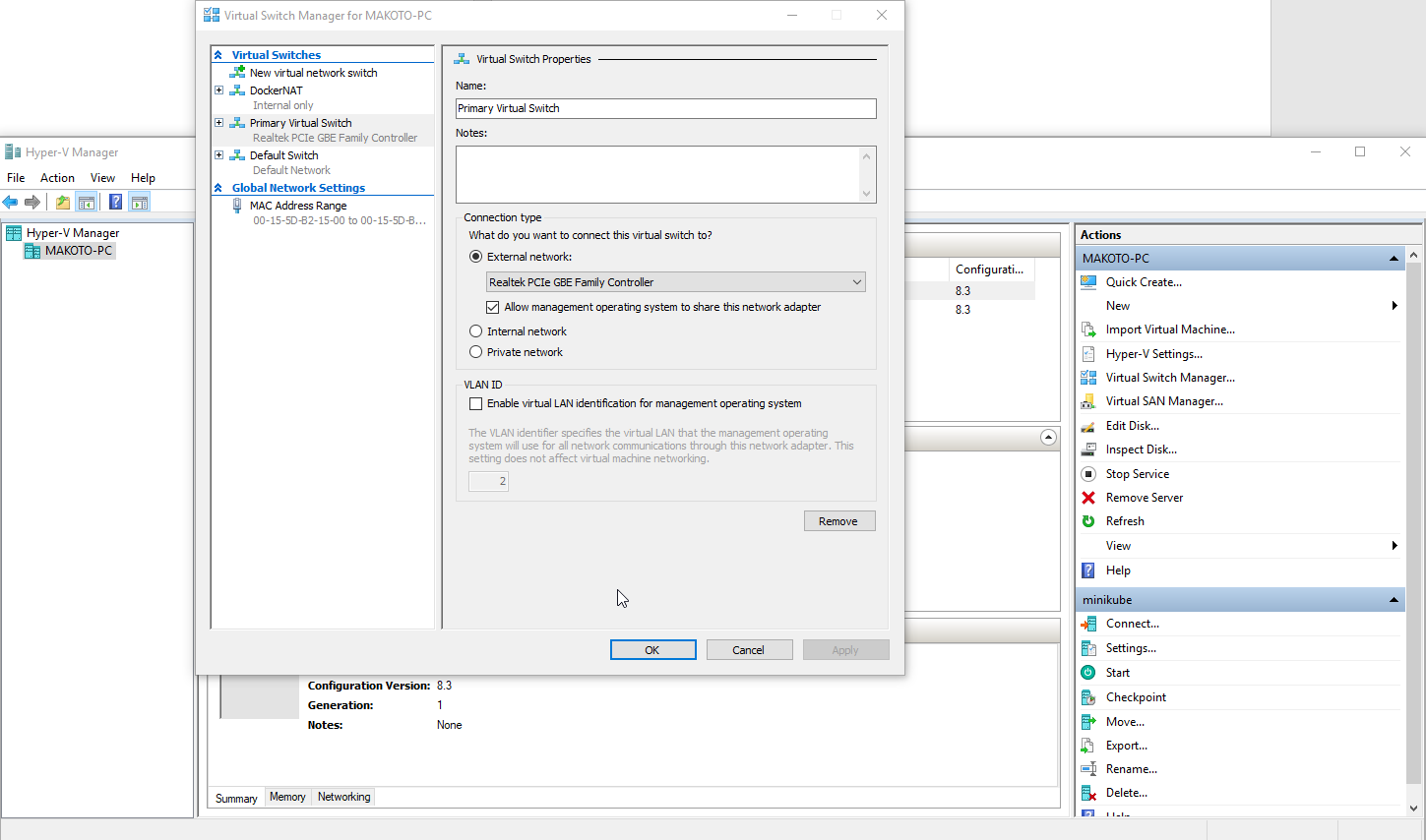
When you like to disable Hyper-V (when Hyper-V is enabled, you cannot run Virtualbox) you can do:

Disable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-All

Of course you can also use the GUI to achieve the same.

## Create a Virtual Switch in the Hyper-V manager

Create a Virtual Switch using the Hyper-V manager. Make sure it is connected to a connected controller (wired or wireless). Name it Primary Virtual Switch. This is required in order not to get issues with the Docker for Windows / Docker Desktop installation.



## Installing Minikube and Helm

You can install minikube using Chocolatey (from a terminal which runs as Administrator):

choco install minikube kubernetes-cli helm

You can also do it manually by downloading and running an installer. Download and run minikube-installer.exe from:

<https://github.com/kubernetes/minikube/releases/latest>

A similar procedure is possible with helm

<https://storage.googleapis.com/kubernetes-helm/helm-v2.12.3-windows-amd64.zip>

## Start Minikube

When using Hyper-V (you should have created the Virtual switch as described before):

minikube start --vm-driver hyperv --hyperv-virtual-switch "Primary Virtual Switch" --show-libmachine-logs --alsologtostderr --v=5

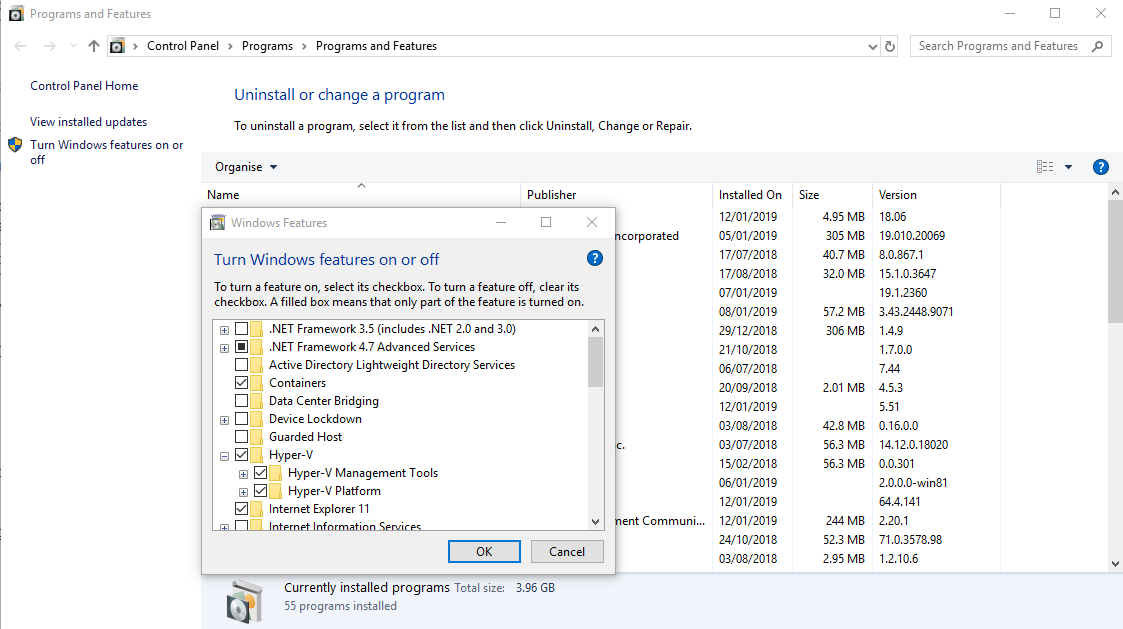
# Linux in Virtualbox

## Preparations

Make sure Hyper-V is disabled if you want to run a Linux VM in Virtualbox which has a local minikube running. From an elevated Powershell:

Disable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-All

On Windows 10: Programs and Features, Hyper-V



The above screenshot shows the feature enabled. You should disable them. When the setting changes, a restart is required.

Install VirtualBox from <https://www.virtualbox.org/wiki/Downloads>. If you’re using Vagrant, consider using a 5.2 version of VirtualBox instead of the latest 6.x version.

Install Vagrant from <https://www.vagrantup.com/downloads.html>

## Build an Ubuntu VM in Virtualbox

Make sure you have a git client installed. On Windows: <https://git-scm.com> or from an elevated Powershell

choco install git

Clone the repository

git clone <https://github.com/AMIS-Services/sig-kubernetes.git>

Go to the sig-kubernetes/SIG01/lab03-helm/env directory

Install two plugins:

vagrant plugin install vagrant-disksize

vagrant plugin install vagrant-vbguest

Build the VM

vagrant up

After building you can start it from Virtualbox. Login with user developer and password Welcome01. The below actions should be executed from within the VM.

## Start Minikube

sudo minikube start --vm-driver none

Minikube using vm-driver none requires root in order to run. In order to make it easier to use minikube:

sudo chmod a+s /usr/local/bin/minikube

## Start the Minikube dashboard

developer@ubuntudev:~$ minikube dashboard

Enabling dashboard ...

Verifying dashboard health ...

Launching proxy ...

Verifying proxy health ...

Opening http://127.0.0.1:40763/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/ in your default browser...

Open in a browser the specified URL. In my case: <http://127.0.0.1:40763/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/>

Do mind that every time you start the dashboard, the port is different.

## Helm

Helm is already installed inside the VM with snap install helm –classic so you do not have to do this.

Install Tiller. This is the Kubernetes side component of Helm

developer@ubuntudev:~$ helm init

Creating /home/developer/.helm

Creating /home/developer/.helm/repository

Creating /home/developer/.helm/repository/cache

Creating /home/developer/.helm/repository/local

Creating /home/developer/.helm/plugins

Creating /home/developer/.helm/starters

Creating /home/developer/.helm/cache/archive

Creating /home/developer/.helm/repository/repositories.yaml

Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com

Adding local repo with URL: http://127.0.0.1:8879/charts

$HELM\_HOME has been configured at /home/developer/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

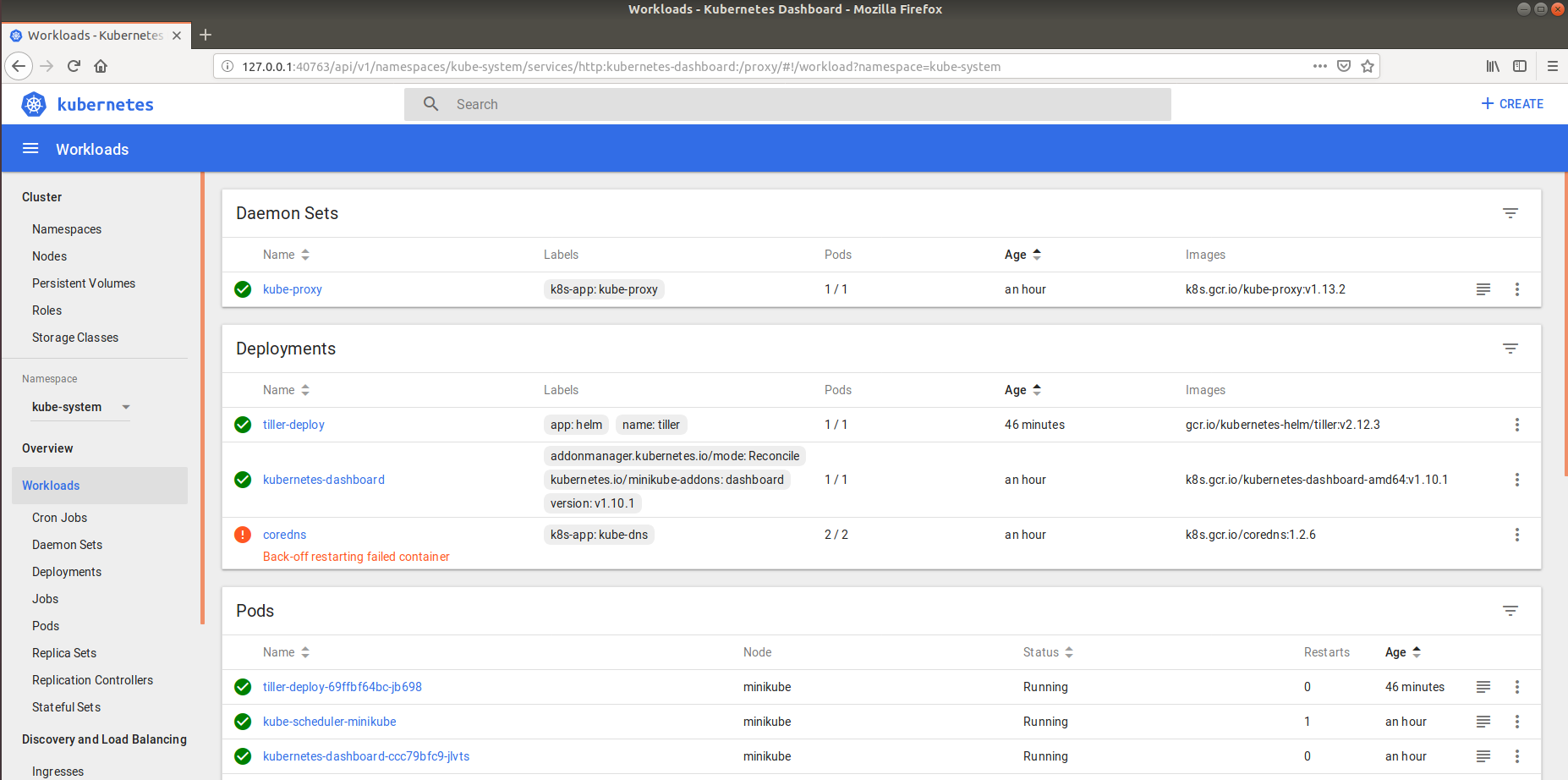
Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.

To prevent this, run `helm init` with the --tiller-tls-verify flag.

For more information on securing your installation see: https://docs.helm.sh/using\_helm/#securing-your-helm-installation

Happy Helming!

Start the Kubernetes Dashboard and confirm Tiller has been deployed:



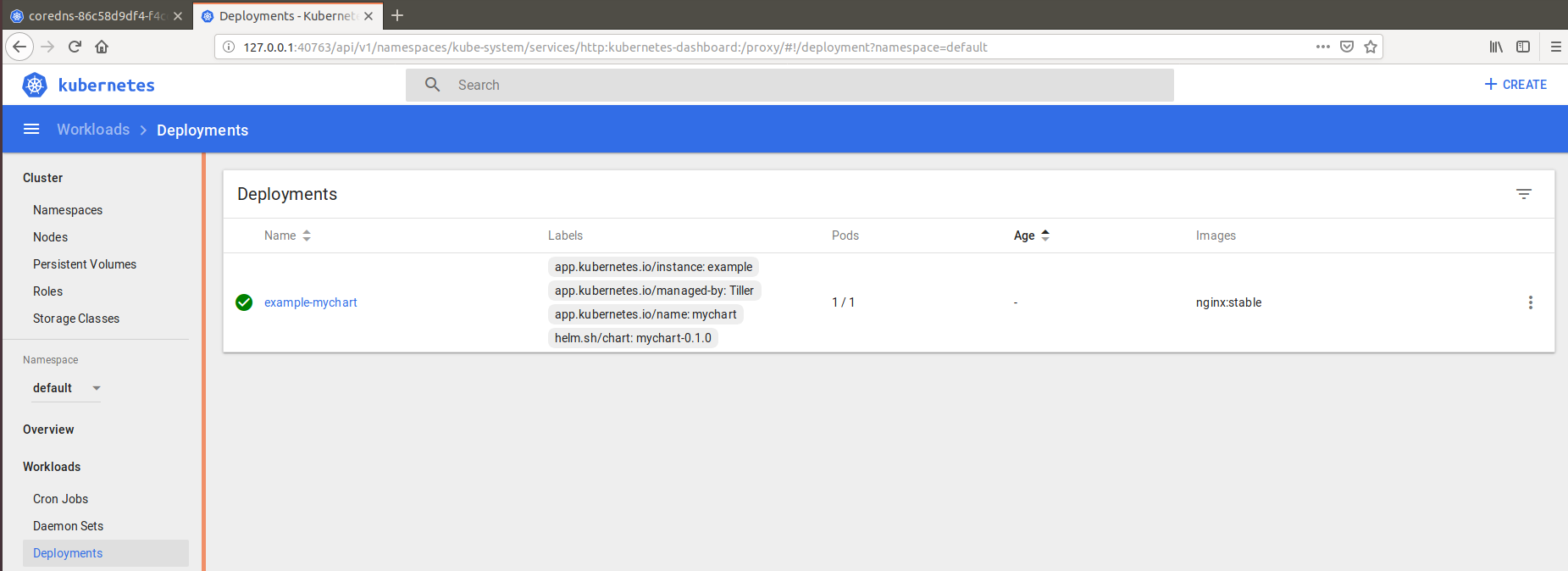
Make sure you’ve selected the kube-system namespace

## Create a first chart

helm create mychart

helm install --name example ./mychart --set service.type=NodePort

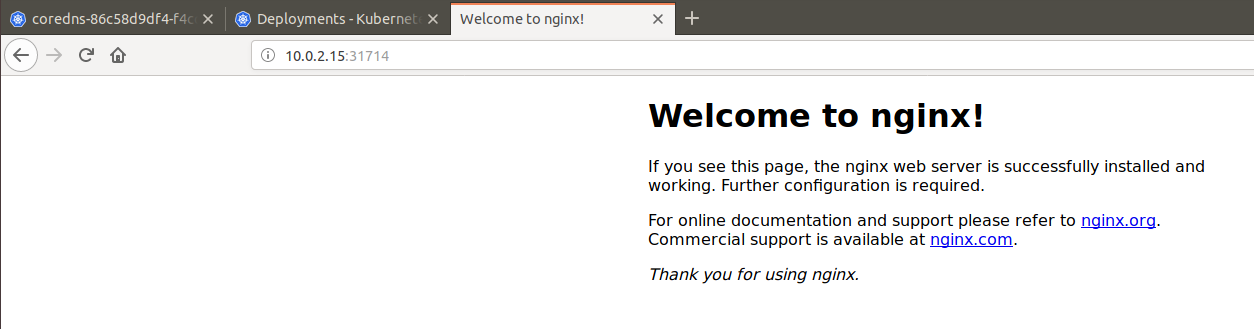
Confirm deployment in the default namespace



Determine the URL of the deployment by executing the output from helm install …



Open the URL in a browser and confirm NGINX is running



## Whiskeyshop

git clone <https://github.com/AMIS-Services/sig-springboot-1.git>

cd sig-springboot-1/lab7-docker/whiskeyshop

mvn clean package

docker build -t whiskeyshop .

Confirm the image has been build:



Test the container by running it

docker run -d -p 8080:8080 whiskeyshop

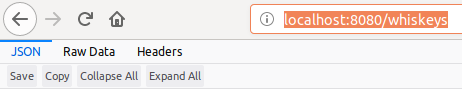
Check it is running

developer@ubuntudev:~/mychart$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

**whiskeyshop latest** d4e736777070 About an hour ago

And going to <http://localhost:8080/whiskeys> in the browser. An empty JSON list should be returned.



Stop the container

docker stop whiskeyshop

docker rm whiskeyshop

## Update the chart to use this container

In values.yaml update image and service:

image:

**repository: whiskeyshop**

**tag: latest**

pullPolicy: IfNotPresent

service:

type: NodePort

**port: 8080**

In templates/service.yaml update the targetport to 8080

spec:

type: {{ .Values.service.type }}

ports:

- port: {{ .Values.service.port }}

targetPort: 8080

In deployment.yaml update the containerPort and probes

spec:

containers:

- name: {{ .Chart.Name }}

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

imagePullPolicy: {{ .Values.image.pullPolicy }}

ports:

- name: http

containerPort: 8080

protocol: TCP

livenessProbe:

httpGet:

path: /whiskeys

port: http

readinessProbe:

httpGet:

path: /whiskeys

port: http

Go to the mychart directory and install the chart

helm install .

Go to the kubernetes dashboard and locate your deployment. Go to your browser and confirm the deployment is accessible at port 8080.

## Cleanup

Remove all releases

helm ls --short | xargs -L1 helm delete