Installing microk8s and Kubeflow on the Xeon GPU-enabled Server

If the server is connected to LAN inaccessible from outside we can disable the ufw firewall by issuing the following command on the server:

```
sudo ufw disable
```

Otherwise keep the firewall running and add appropriate firewall rule. For details consult this link.

We are going to use **microk8s** for a single node deployment with GPU support on the server which we assembled. The beauty of microk8s is that it runs directly on the Linux server hardware without the help of virtual machine (e.g. minikube). As such enabling GPU support for microk8s is easy and provides important speedup with ML workloads.

We are going to install microk8s v1.20 via snap:

```
sudo snap install microk8s --classic --channel=1.20/stable
```

For details visit the official microk8s docs on this link and this link.

After about 10 mins the basic microk8s installation completes.

We manually enable the following add-ons:

```
microk8s enable gpu
```

```
microk8s dns ingress dashboard
```

```
kubectl -n kube-system edit service kubernetes-dashboard
```

change the `.spec.type` to `NodePort`

and finally

```
microk8s enable kubeflow.
```

Notice the output of the last command:

```
Congratulations, Kubeflow is now available.

The dashboard is available at http://10.64.140.43.xip.io

    Username: admin
    Password: VZWQ**********************S3

To see these values again, run:

    microk8s juju config dex-auth static-username
    microk8s juju config dex-auth static-password


To tear down Kubeflow and associated infrastructure, run:

    microk8s disable kubeflow
```

We are going to expose the Kubernetes dashboard running on our server to a remote host on the same LAN.

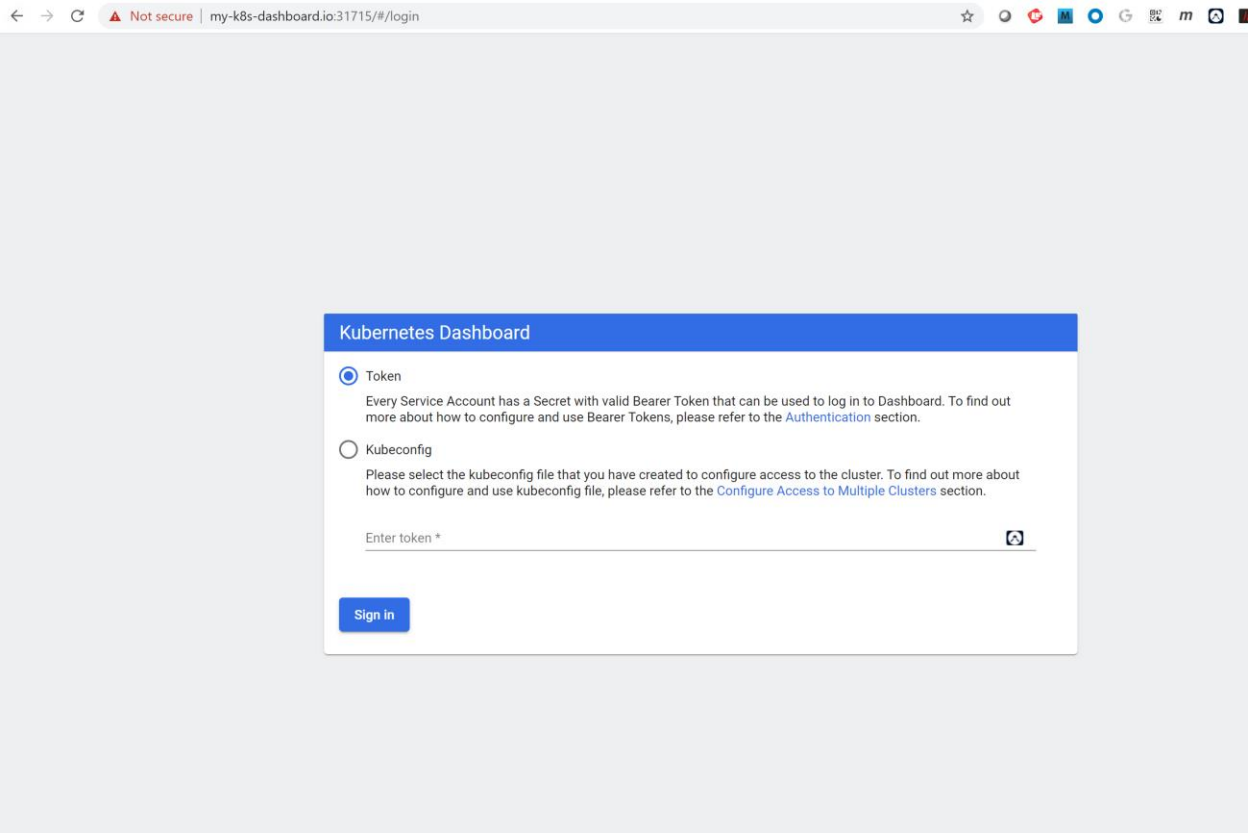
To make our life complicated let's assume our remote host runs Windows 10. Open its hosts file located in C:\Windows\System32\drivers\etc and add the following entry at the bottom:

```
<my-server-ip>   my-k8s-dashboard.io
```

Then on the server find the string value of default token with the lines below:

```
token=$(microk8s kubectl -n kube-system get secret | grep default-token | cut -d " " -f1)
microk8s kubectl -n kube-system describe secret $token
```

The token is sent to **stdout**. Copy the token and paste it below on a browser on our remote host:

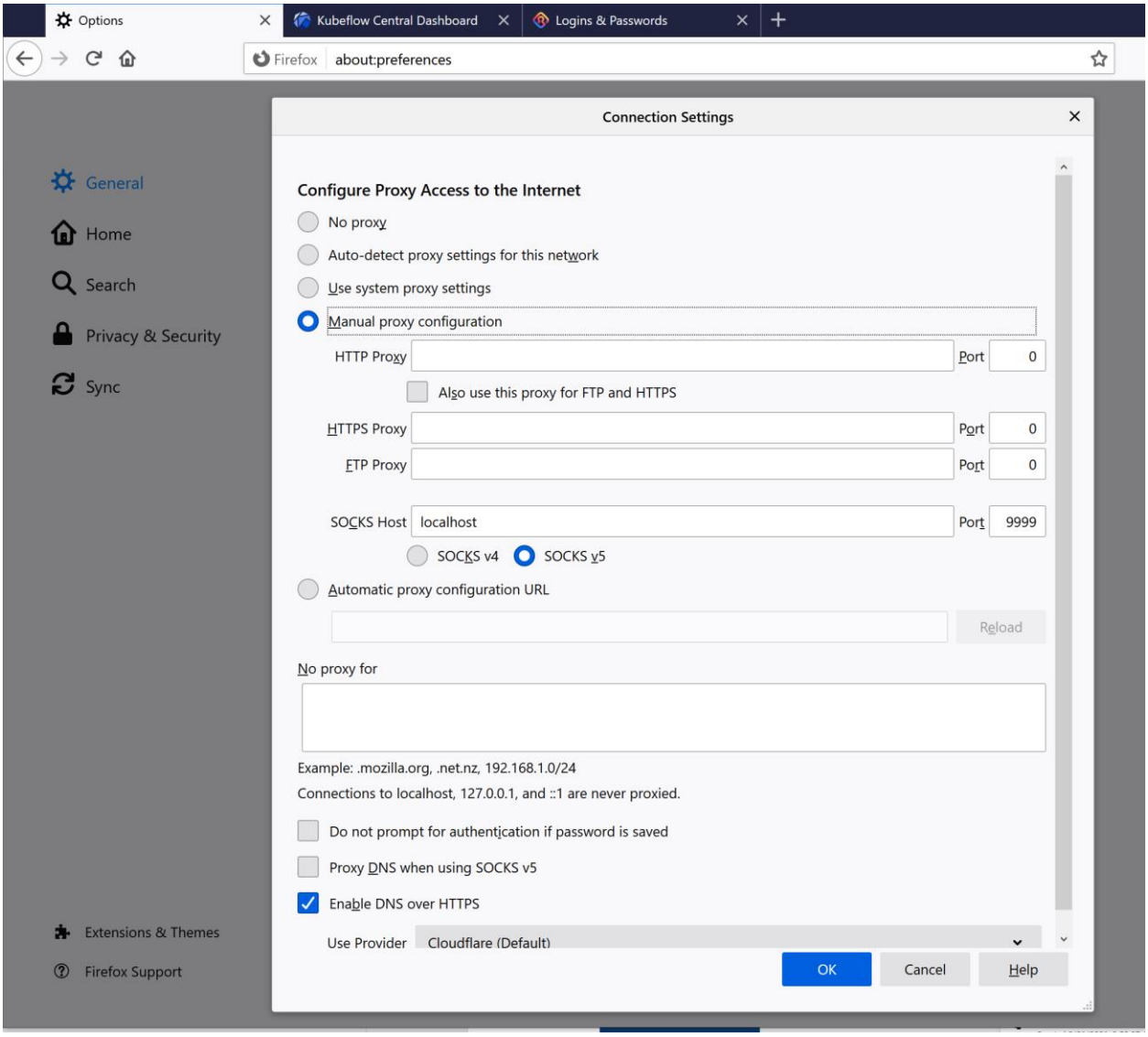


We are going to use SOCKS5 proxy for forwarding our requests to the server:

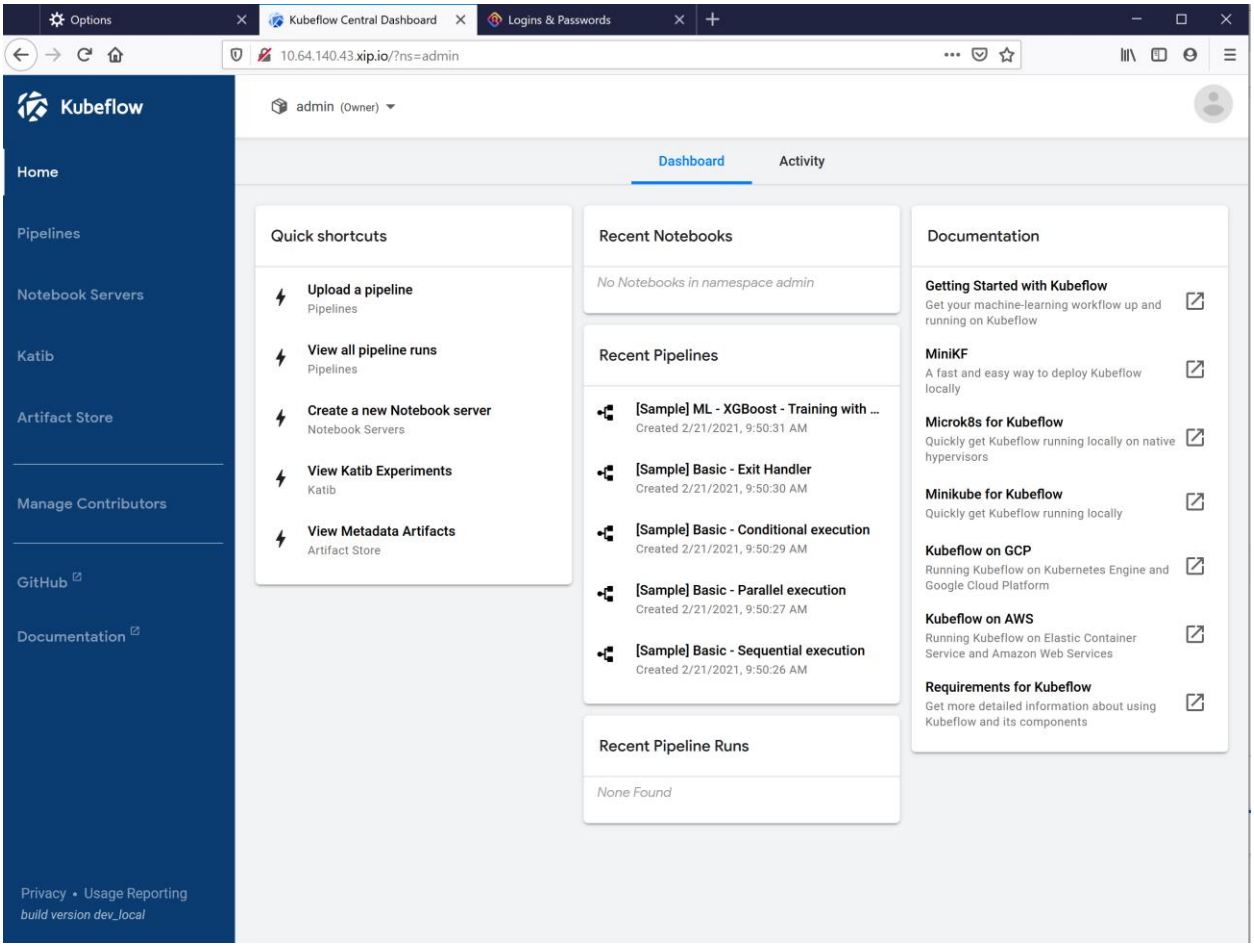
For details see [this link] and [this github microk8s thread].

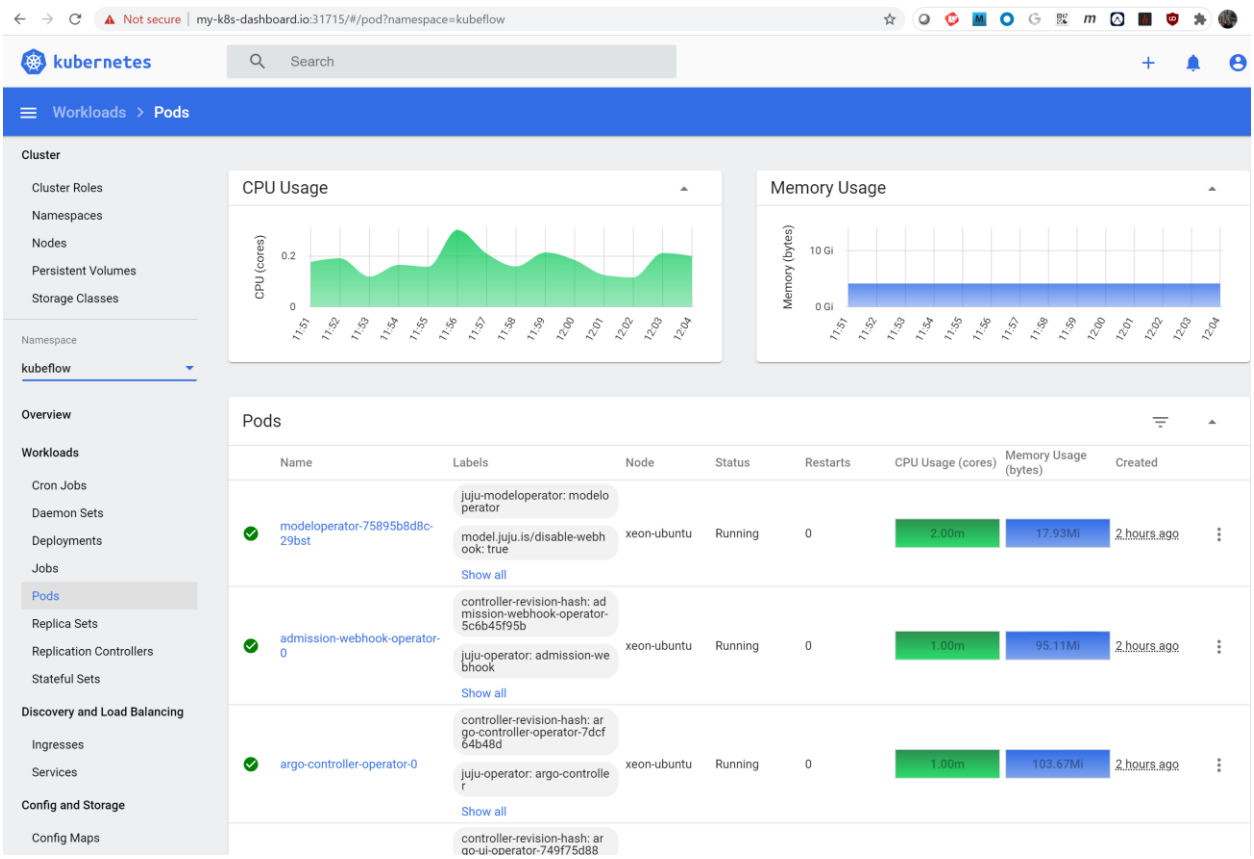
```
ssh -D9999 <username>@<network-server-ip>
```

We are going to configure Mozilla browser to use SOCKS5 Proxy Access to internet on port 9999 as shown below:



Then we can access the Kubeflow dashboard


Options
Kubeflow Central Dashboard
Logins & Passwords
10.64.140.43.xip.io/?ns=admin
Kubeflow
admin (Owner)
Home
Pipelines
Notebook Servers
Katib
Artifact Store
Manage Contributors
GitHub
Documentation
Privacy • Usage Reporting
build version dev_local
Dashboard
Activity
Quick shortcuts
Upload a pipeline
Pipelines
View all pipeline runs
Pipelines
Create a new Notebook server
Notebook Servers
View Katib Experiments
Katib
View Metadata Artifacts
Artifact Store
Recent Notebooks
No Notebooks in namespace admin
Recent Pipelines
[Sample] ML - XGBoost - Training with ...
Created 2/21/2021, 9:50:31 AM
[Sample] Basic - Exit Handler
Created 2/21/2021, 9:50:30 AM
[Sample] Basic - Conditional execution
Created 2/21/2021, 9:50:29 AM
[Sample] Basic - Parallel execution
Created 2/21/2021, 9:50:27 AM
[Sample] Basic - Sequential execution
Created 2/21/2021, 9:50:26 AM
Recent Pipeline Runs
None Found
Documentation
Getting Started with Kubeflow
Get your machine-learning workflow up and running on Kubeflow
MiniKF
A fast and easy way to deploy Kubeflow locally
Microk8s for Kubeflow
Quickly get Kubeflow running locally on native hypervisors
Minikube for Kubeflow
Quickly get Kubeflow running locally
Kubeflow on GCP
Running Kubeflow on Kubernetes Engine and Google Cloud Platform
Kubeflow on AWS
Running Kubeflow on Elastic Container Service and Amazon Web Services
Requirements for Kubeflow
Get more detailed information about using Kubeflow and its components

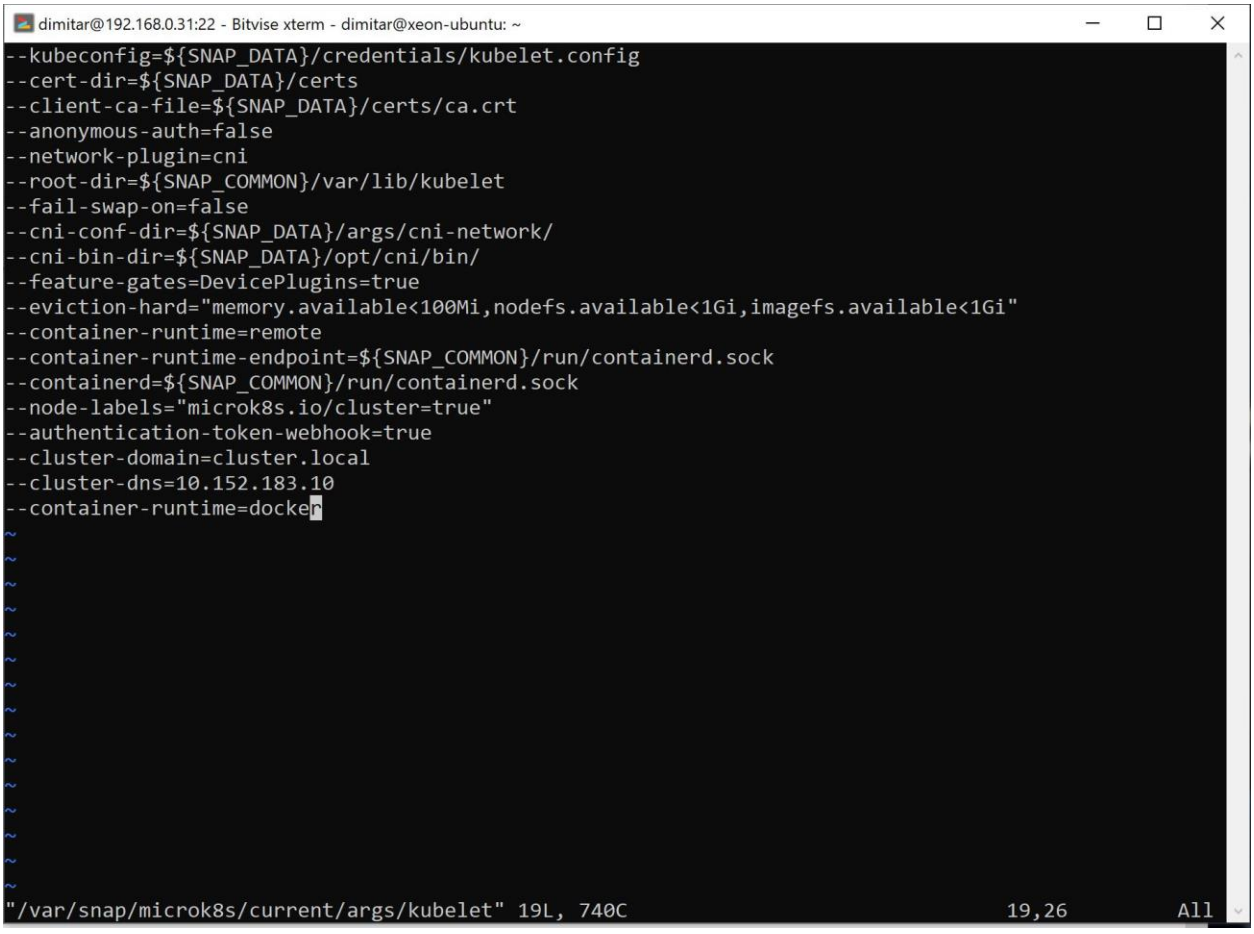

Not secure | my-k8s-dashboard.io:31715/#/pod?namespace=kubeflow
kubernetes
Search
Workloads > Pods
Cluster
Cluster Roles
Namespaces
Nodes
Persistent Volumes
Storage Classes
Namespace
kubeflow
Overview
Workloads
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Discovery and Load Balancing
Ingresses
Services
Config and Storage
Config Maps
CPU Usage
CPU (cores)
0.2
0
11:51 11:52 11:53 11:54 11:55 11:56 11:57 11:58 11:59 12:00 12:01 12:02 12:03 12:04
Memory Usage
Memory (bytes)
10 Gi
0 Gi
11:51 11:52 11:53 11:54 11:55 11:56 11:57 11:58 11:59 12:00 12:01 12:02 12:03 12:04
Pods
Name
Labels
Node
Status
Restarts
CPU Usage (cores)
Memory Usage (bytes)
Created
modeloperator-75895b8d8c-29bst
juju-modeloperator: modeloperator
model.juju.is/disable-webhook: true
Show all
xeon-ubuntu
Running
0
2.00m
17.93Mi
2 hours ago
admission-webhook-operator-0
controller-revision-hash: admission-webhook-operator-5c6b45f95b
juju-operator: admission-webhook
Show all
xeon-ubuntu
Running
0
1.00m
95.11Mi
2 hours ago
argo-controller-operator-0
controller-revision-hash: argo-controller-operator-7dcf64b48d
juju-operator: argo-controller
Show all
xeon-ubuntu
Running
0
1.00m
103.67Mi
2 hours ago
controller-revision-hash: argo-ui-operator-749f75d88

Possible issues with NVIDIA GPU support in containerd

By default, microk8s as of version v1.19 by default uses `containerd` as the container runtime.
With certain releases of microk8s there might be issues with the NVIDIA GPU support in the currently used version of `containerd`. This can be seen by running `` `microk8s inspect` `` and looking through the inspection report for an error `` `Failed to initialize NVML` ``:

```
dimitar@xeon-ubuntu:~/inspection-reports/inspection-report$ grep -E -R -n 'Failed to initialize NVML
' .
./k8s/cluster-info-dump:16596:2021/03/21 17:01:54 Failed to initialize NVML: could not load NVML lib
rary.
dimitar@xeon-ubuntu:~/inspection-reports/inspection-report$
```

The problem is due to intermittent problem/bug with the currently used containerd version and the solution is to switch from using `containerd` as a runtime to `docker` by editing the file `/var/snap/microk8s/current/args/kubelet` and adding the line:
`--container-runtime=docker`
For details on this error check this discussion thread.
The edited `/var/snap/microk8s/current/args/kubelet` should look like:



Below it is shown is a simple test manifest which can be used to test if the GPU has been enabled and recognized by microk8s correctly:

Figure: **gpu_test.yaml**

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
    - name: cuda-vector-add
      # https://github.com/kubernetes/kubernetes/blob/v1.7.11/test/images/nvidia-
cuda/Dockerfile
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 1 GPU per container
  nodeSelector:
    accelerator: nvidia-tesla-m40 # or nvidia-tesla-k80 etc.
  tolerations:
  - effect: NoSchedule
    operator: Exists
```

Before applying this manifest to the running microk8s cluster make sure you label your current K8s node as:

```
microk8s kubectl label nodes xeon-ubuntu accelerator=nvidia-tesla-m40
```

After applying the manifest in the last Figure as:

```
microk8s kubectl create -f gpu-test.yaml
```

check the current state of the newly created pod `cuda-verctor-add` in the default namespace. If the GPU is recognized by microk8s the event *ContainerCreating* indicates that the pod is scheduled for creation.

```
dimitar@xeon-ubuntu:~$ microk8s kubectl create -f gpu-test.yaml
pod/cuda-vector-add created
dimitar@xeon-ubuntu:~$ microk8s kubectl get pods
NAME              READY   STATUS              RESTARTS   AGE
cuda-vector-add   0/1     ContainerCreating   0          7s
dimitar@xeon-ubuntu:~$ microk8s kubectl get pods
NAME              READY   STATUS      RESTARTS   AGE
cuda-vector-add   0/1     Completed   0          98s
```

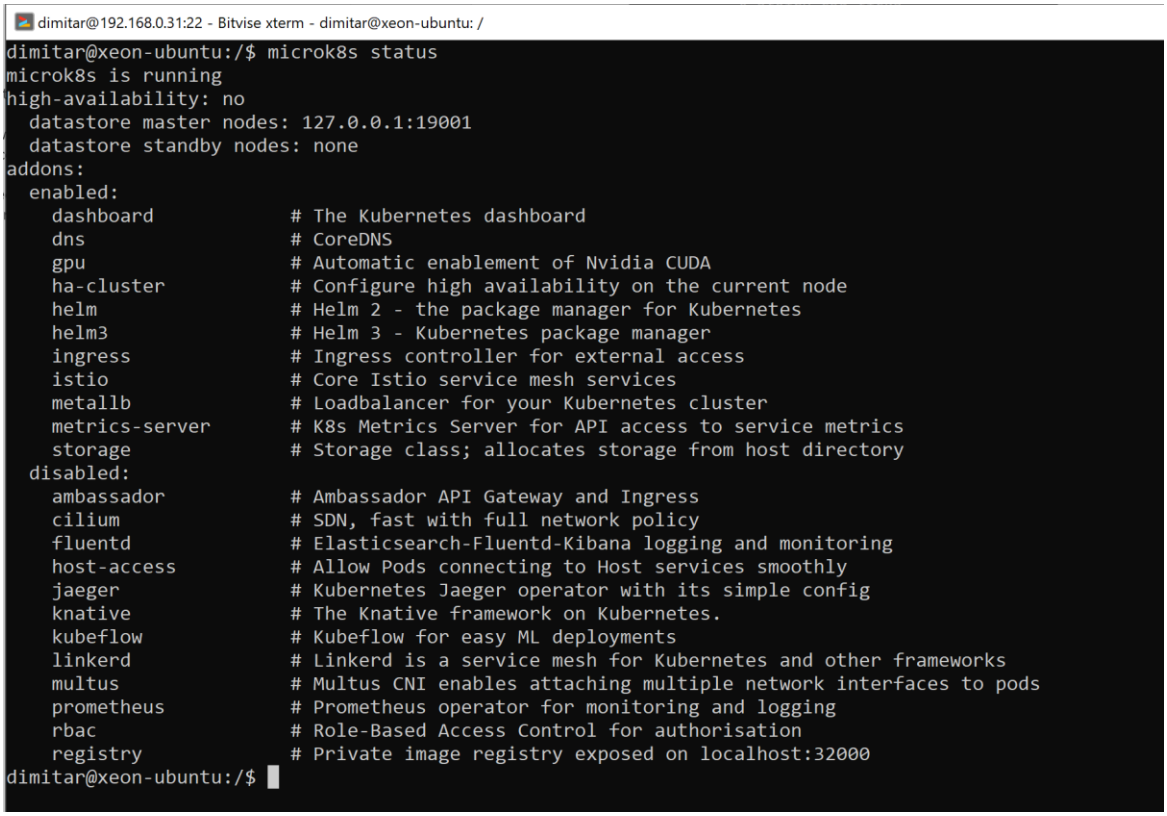
Also if one inspects the result from kubectl describe pod `cuda-vector-add` one should see the following sequence of events indicating normal creation and completion:

```
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  4m9s   default-scheduler  Successfully assigned default/cuda-vector-add to xeon
-ubuntu
  Normal  Pulling    4m7s   kubelet            Pulling image "k8s.gcr.io/cuda-vector-add:v0.1"
  Normal  Pulled     3m21s  kubelet            Successfully pulled image "k8s.gcr.io/cuda-vector-add
:v0.1" in 46.165611779s
  Normal  Created    3m19s  kubelet            Created container cuda-vector-add
  Normal  Started    3m17s  kubelet            Started container cuda-vector-add
```

If the GPU is not recognized the pod `cuda-vector-add` won't be scheduled at all due to insufficient GPU resources.

Note that it is possible that the GPU ***is not recognized*** even if the result of `microk8s enable gpu` is not an error and `microk8s status` showing that the gpu has been enabled as shown below.

Figure: showing that the NVIDIA GPU is enabled in microk8s can be misleading as it still may not be recognized and thus unavailable



And here is how the Jupyter Notebook would look like with GPU enabled. For the illustration it is used the following Deep Learning example and the MNIST database:
https://github.com/tensorflow/docs/blob/master/site/en/tutorials/quickstart/advanced.ipynb

Figure: Configure the Jupyter Notebook server to use 2 Logical cores, 16GB RAM and 1 GPU

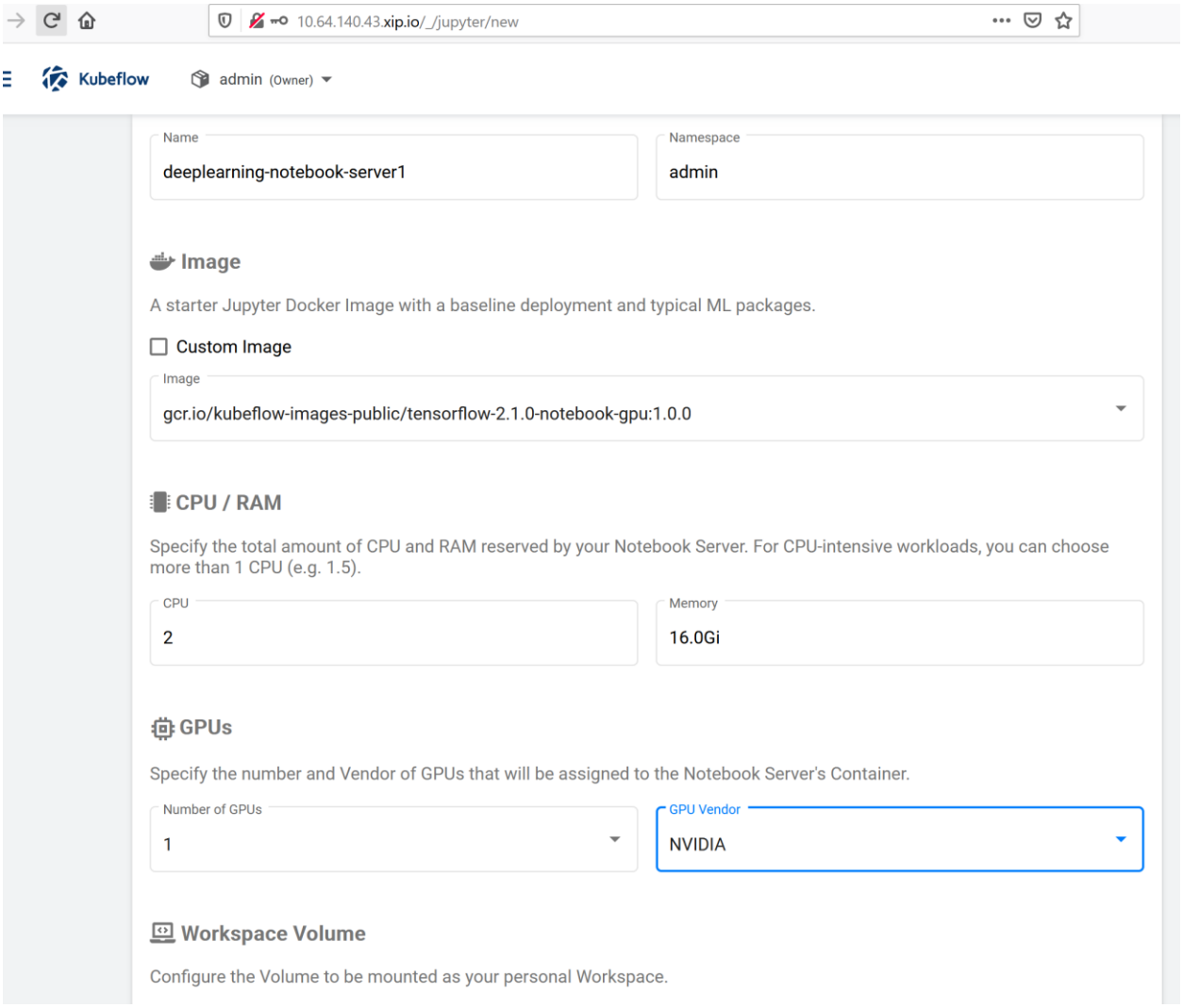


Figure: **GPU-enabled** Jupyter notebook server and another one **with no GPU** are launched

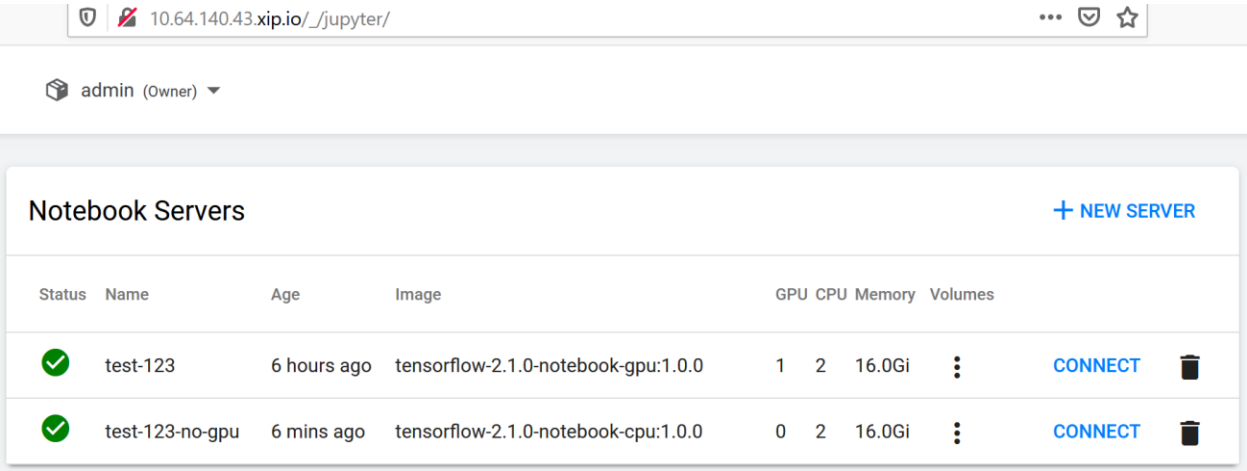


Figure: The result of execution of the Advanced Deep learning example with 5 training epochs **with GPU enabled**: ***39 secs*** for the training and testing loop.

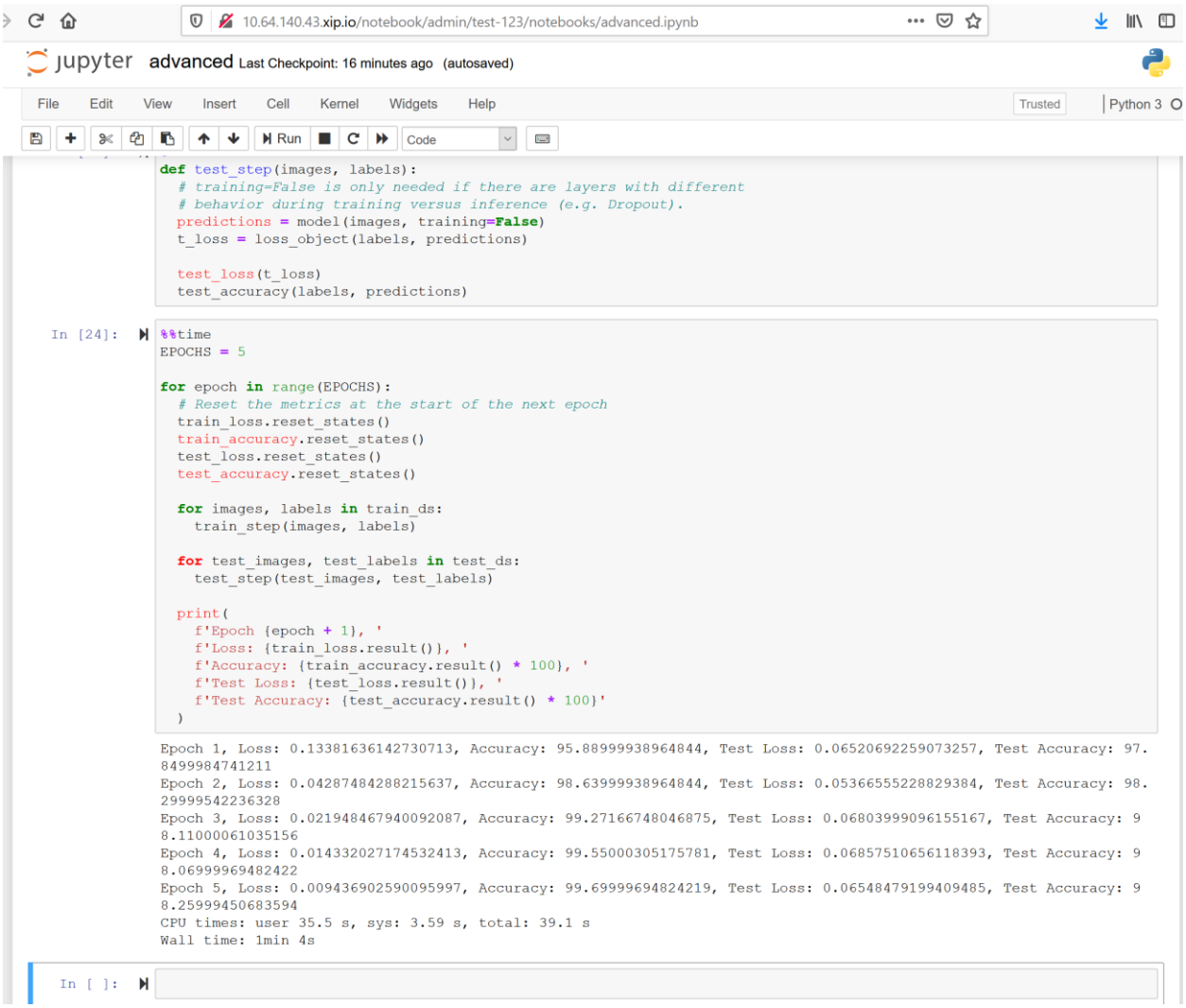


Figure: The result of execution of the Advanced Deep learning example with 5 training epochs **with no GPU**: ***17 minutes*** for the training and testing loop. *It takes forever…* ☹

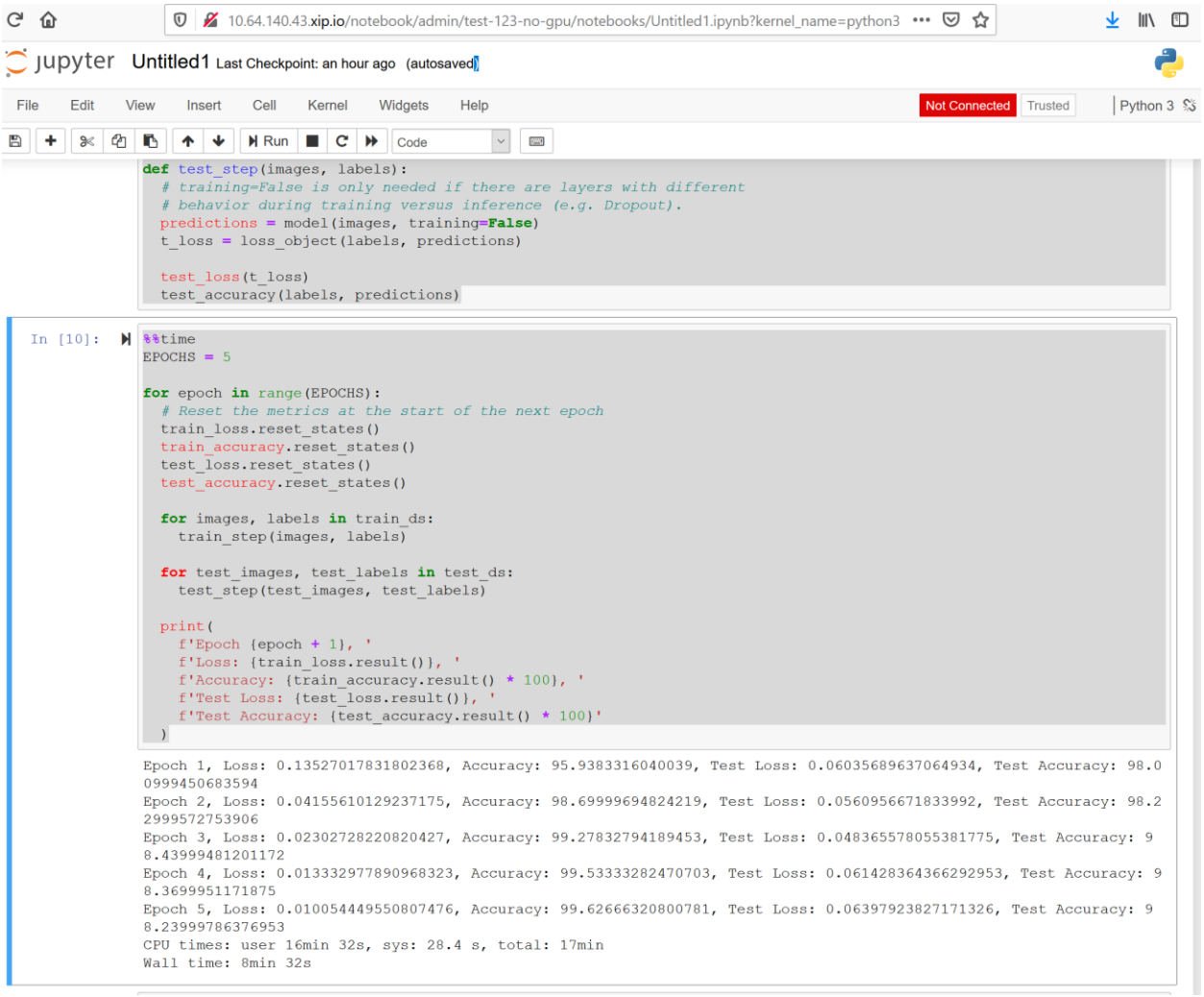


Figure: The result of execution of the Advanced Deep learning example with 5 training epochs on my notebook Surface Book2 with 16GB of RAM and GTX 1060 with 6GB dedicated VRAM **with GPU enabled**: ***34 secs*** for the training and testing loop.