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.

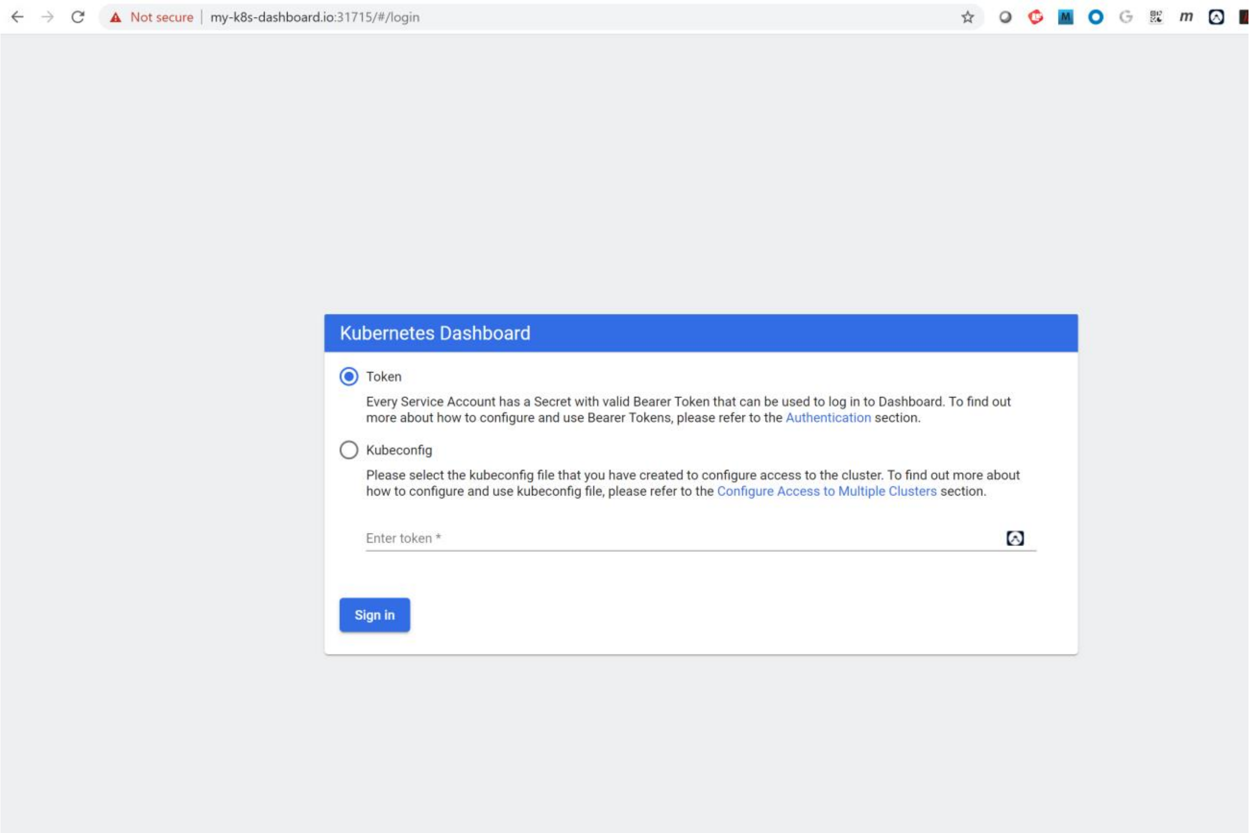
To make our life complicated let's assume our remote host runs Windows 10. Open its hosts file located in C:\Windows\System32\drivers\etc and add the following entry at the bottom:

```
<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:



Not secure | my-k8s-dashboard.io:31715/#/login

Kubernetes Dashboard

● Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

○ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

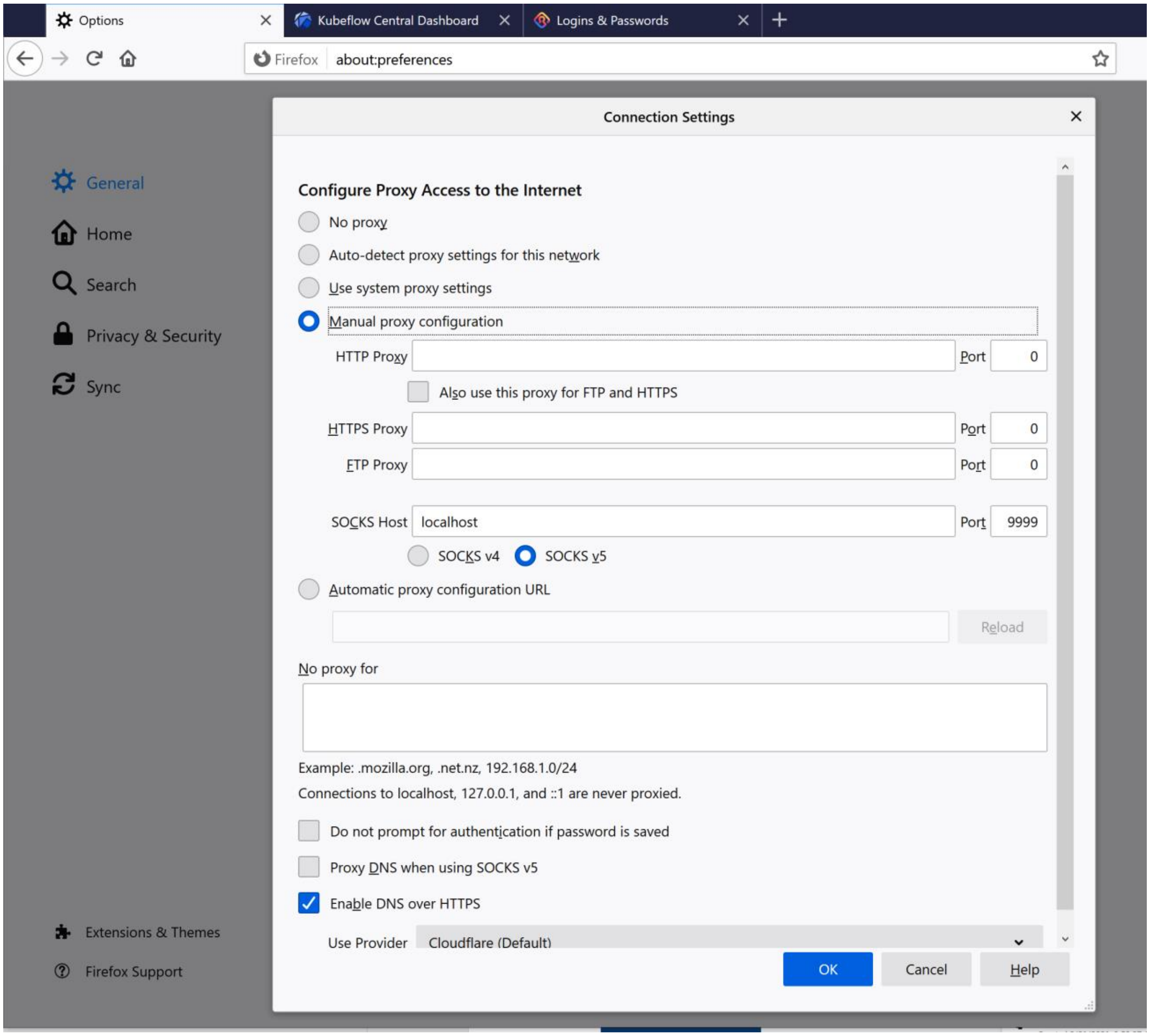
Enter token *

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:



Connection Settings

Configure Proxy Access to the Internet

○ No proxy
○ Auto-detect proxy settings for this network
○ Use system proxy settings
● Manual proxy configuration

HTTP Proxy ______ Port 0

□ Also use this proxy for FTP and HTTPS

HTTPS Proxy ______ Port 0

FTP Proxy ______ Port 0

SOCKS Host localhost Port 9999

○ SOCKS v4 ● SOCKS v5

○ Automatic proxy configuration URL

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

Connections to localhost, 127.0.0.1, and ::1 are never proxied.

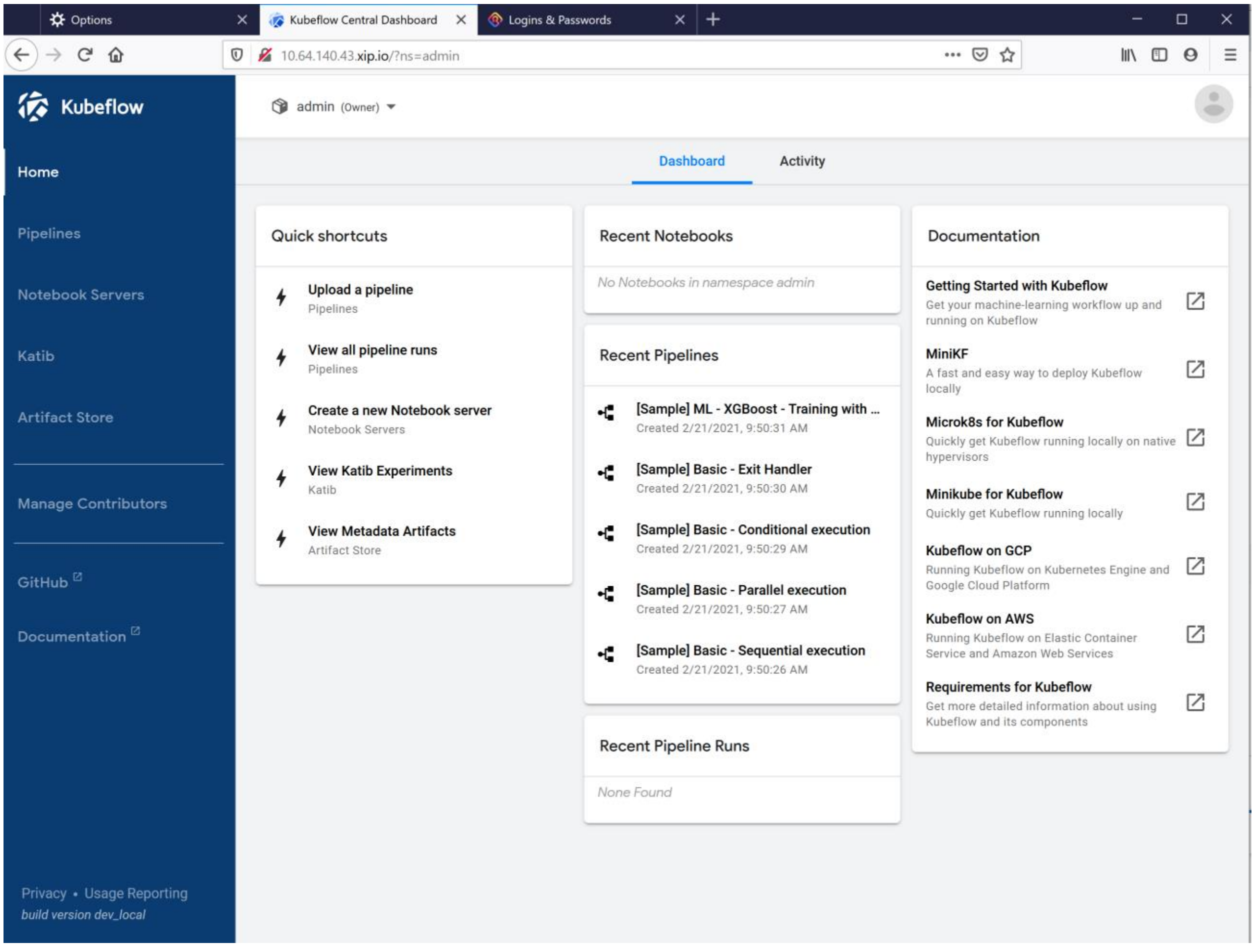
□ Do not prompt for authentication if password is saved

□ Proxy DNS when using SOCKS v5

☑ Enable DNS over HTTPS

Use Provider Cloudflare (Default)

Then we can access the Kubeflow dashboard



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

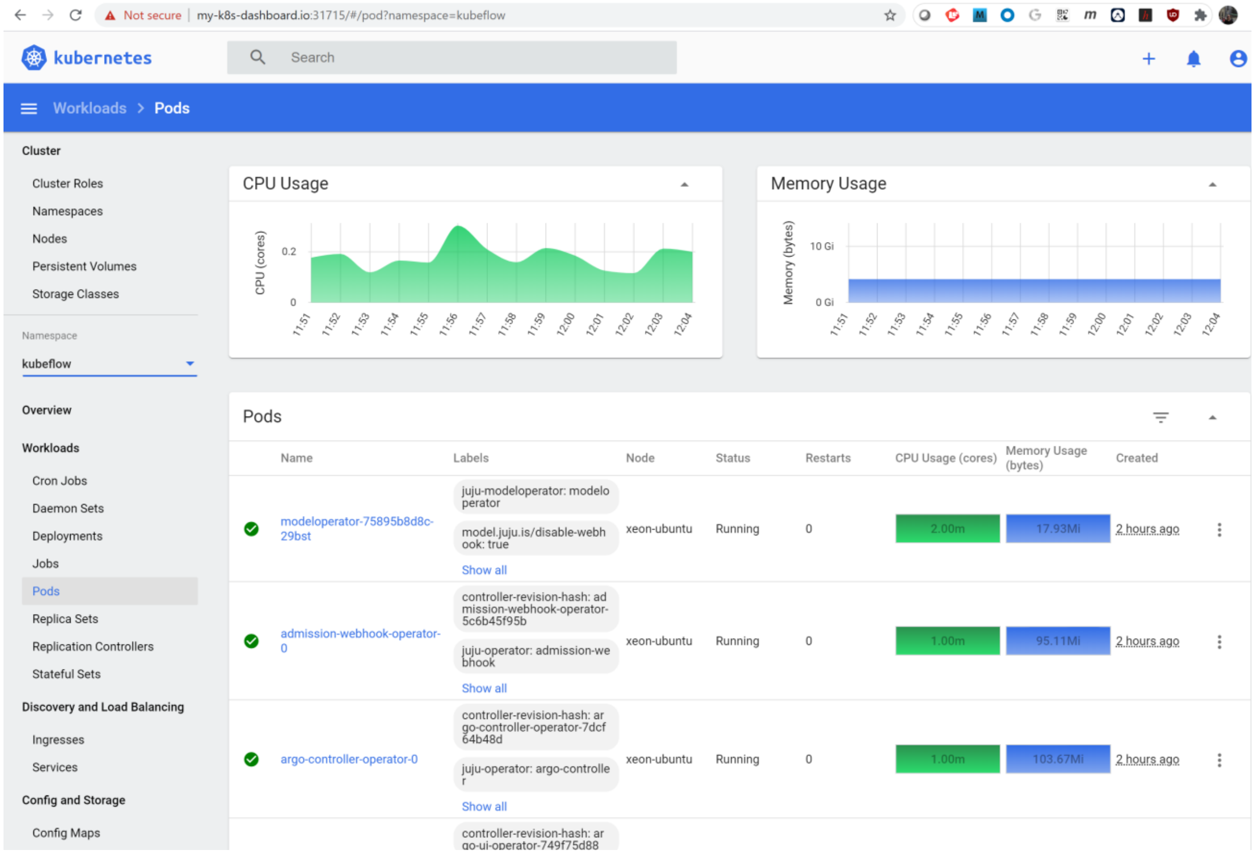
Recent Pipeline Runs

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

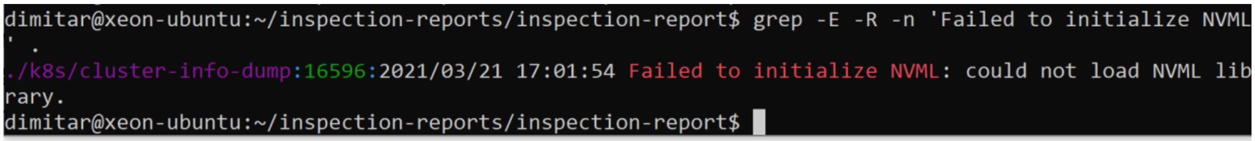
Privacy • Usage Reporting
build version dev_local



	Name	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created	
✓	modeloperator-75895b8d8c-29bst	juju-modeloperator: modelo perator model.juju.is/disable-webh ook: true Show all	xeon-ubuntu	Running	0	2.00m	17.93Mi	2 hours ago	⋮
✓	admission-webhook-operator-0	controller-revision-hash: ad mission-webhook-operator-5c6b45f95b juju-operator: admission-we bhook Show all	xeon-ubuntu	Running	0	1.00m	95.11Mi	2 hours ago	⋮
✓	argo-controller-operator-0	controller-revision-hash: ar go-controller-operator-7dcf 64b48d juju-operator: argo-controlle r Show all	xeon-ubuntu	Running	0	1.00m	103.67Mi	2 hours ago	⋮
		controller-revision-hash: ar go-ui-operator-749f75d88							

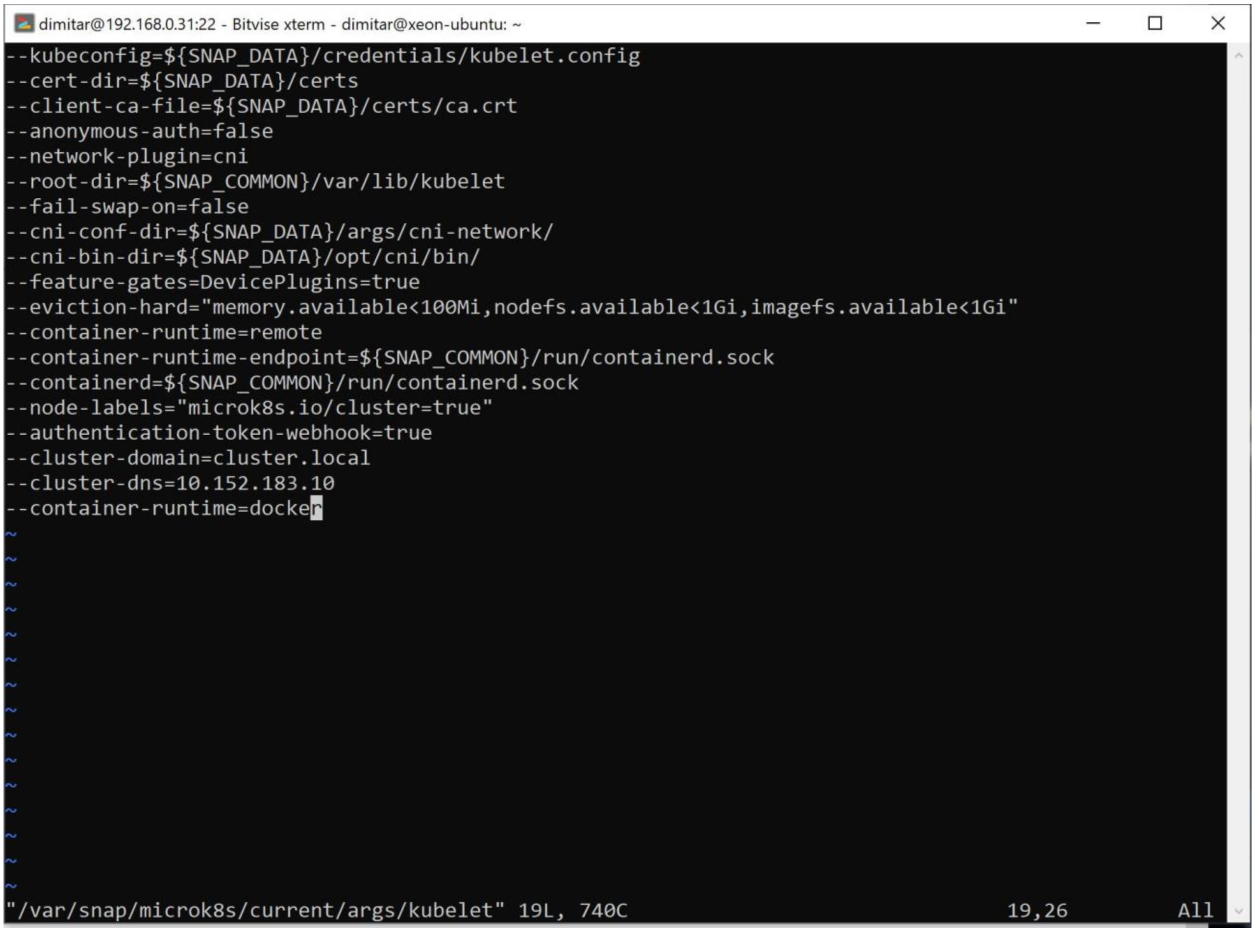
Possible issues with NVIDIA GPU support in containerd

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:


dimitar@192.168.0.31:22 - Bitvise xterm - dimitar@xeon-ubuntu: ~

```
--kubeconfig=${SNAP_DATA}/credentials/kubelet.config
--cert-dir=${SNAP_DATA}/certs
--client-ca-file=${SNAP_DATA}/certs/ca.crt
--anonymous-auth=false
--network-plugin=cni
--root-dir=${SNAP_COMMON}/var/lib/kubelet
--fail-swap-on=false
--cni-conf-dir=${SNAP_DATA}/args/cni-network/
--cni-bin-dir=${SNAP_DATA}/opt/cni/bin/
--feature-gates=DevicePlugins=true
--eviction-hard="memory.available<100Mi,nodefs.available<1Gi,imagefs.available<1Gi"
--container-runtime=remote
--container-runtime-endpoint=${SNAP_COMMON}/run/containerd.sock
--containerd=${SNAP_COMMON}/run/containerd.sock
--node-labels="microk8s.io/cluster=true"
--authentication-token-webhook=true
--cluster-domain=cluster.local
--cluster-dns=10.152.183.10
--container-runtime=docker
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/var/snap/microk8s/current/args/kubelet" 19L, 740C                      19,26          All
```

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
```

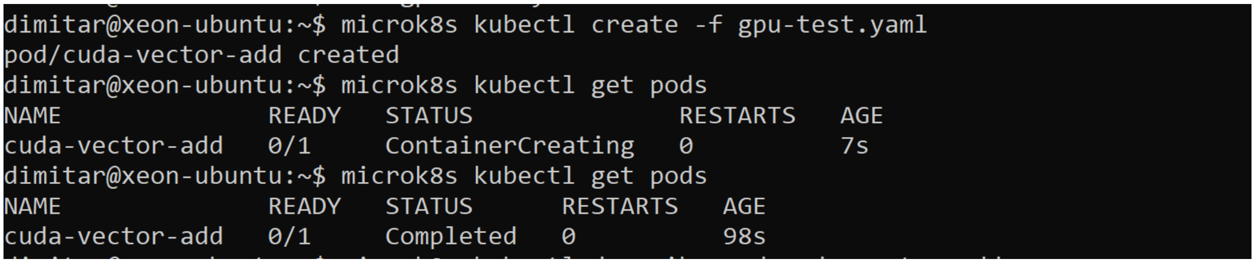
Before applying this manifest to the running microk8s cluster make sure you label your current K8s node as:

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

After applying the manifest in the last Figure as:

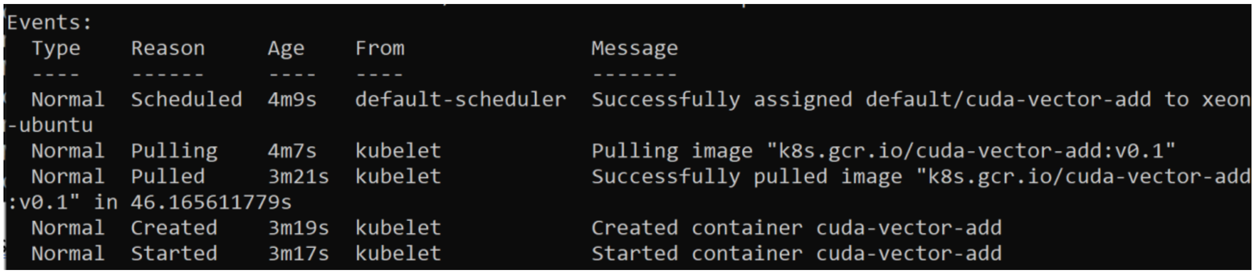
```
microk8s kubectl create -f gpu-test.yaml
```

check the current state of the newly created pod `gpu-test` 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 gpu-test one should see the following sequence of events indicating normal creation and completion:

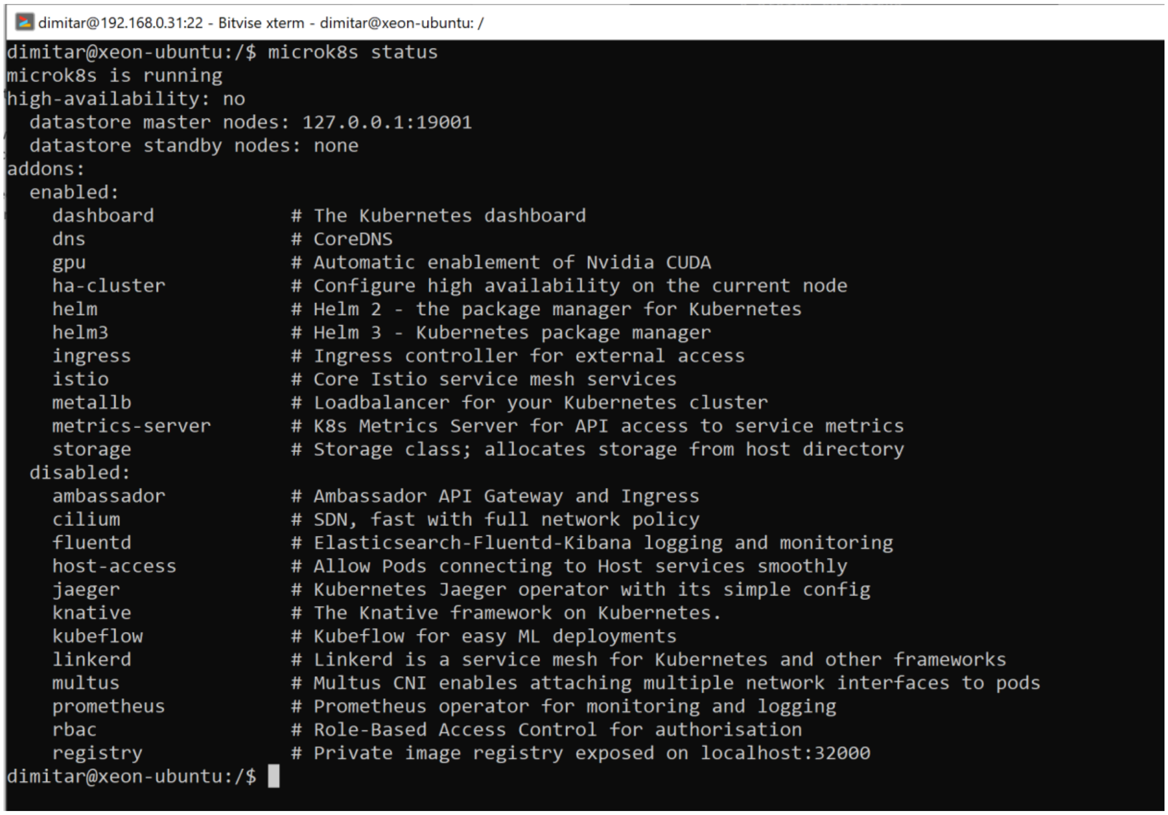


```
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  4m9s   default-scheduler  Successfully assigned default/cuda-vector-add to xeon
-ubuntu
  Normal  Pulling    4m7s   kubelet            Pulling image "k8s.gcr.io/cuda-vector-add:v0.1"
  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 `gpu-test` won't be scheduled at all due to insufficient GPU resources.

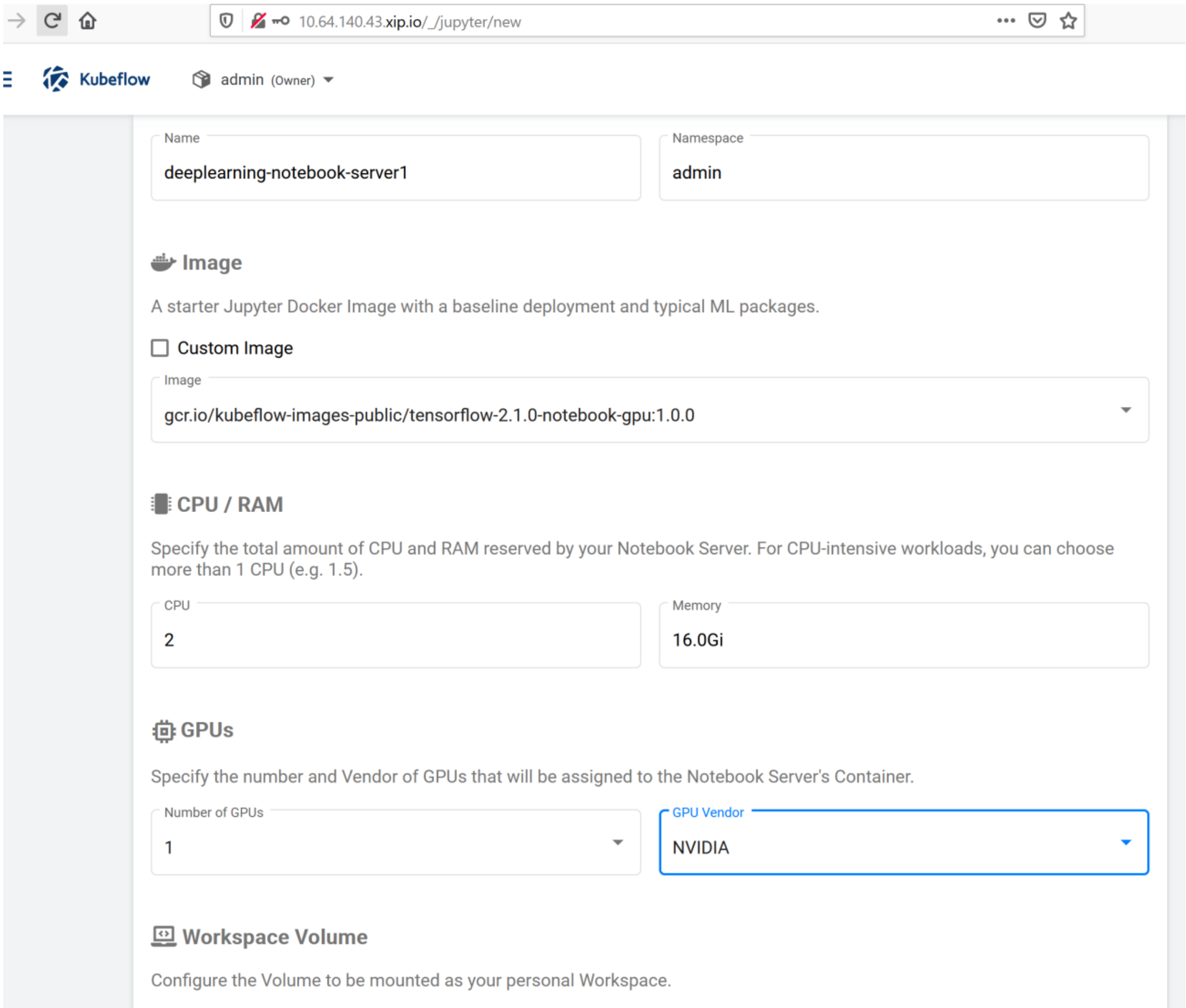
Note that it is possible that the GPU ***is not recognized*** even if the result of `microk8s enable gpu` is not an error and `microk8s status` showing that the gpu has been enabled as shown below.

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


dimitar@192.168.0.31:22 - Bitvise xterm - dimitar@xeon-ubuntu: /

```
dimitar@xeon-ubuntu:/$ microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
addons:
  enabled:
    dashboard            # The Kubernetes dashboard
    dns                  # CoreDNS
    gpu                  # Automatic enablement of Nvidia CUDA
    ha-cluster           # Configure high availability on the current node
    helm                 # Helm 2 - the package manager for Kubernetes
    helm3                # Helm 3 - Kubernetes package manager
    ingress              # Ingress controller for external access
    istio                # Core Istio service mesh services
    metallb              # Loadbalancer for your Kubernetes cluster
    metrics-server       # K8s Metrics Server for API access to service metrics
    storage              # Storage class; allocates storage from host directory
  disabled:
    ambassador           # Ambassador API Gateway and Ingress
    cilium               # SDN, fast with full network policy
    fluentd              # Elasticsearch-Fluentd-Kibana logging and monitoring
    host-access          # Allow Pods connecting to Host services smoothly
    jaeger               # Kubernetes Jaeger operator with its simple config
    knative              # The Knative framework on Kubernetes.
    kubeflow             # Kubeflow for easy ML deployments
    linkerd              # Linkerd is a service mesh for Kubernetes and other frameworks
    multus               # Multus CNI enables attaching multiple network interfaces to pods
    prometheus           # Prometheus operator for monitoring and logging
    rbac                 # Role-Based Access Control for authorisation
    registry             # Private image registry exposed on localhost:32000
dimitar@xeon-ubuntu:/$ 
```

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



Name: deeplearning-notebook-server1

Namespace: admin

Image

A starter Jupyter Docker Image with a baseline deployment and typical ML packages.

□ Custom Image

Image: gcr.io/kubeflow-images-public/tensorflow-2.1.0-notebook-gpu:1.0.0

CPU / RAM

Specify the total amount of CPU and RAM reserved by your Notebook Server. For CPU-intensive workloads, you can choose more than 1 CPU (e.g. 1.5).

CPU: 2

Memory: 16.0Gi

GPUs

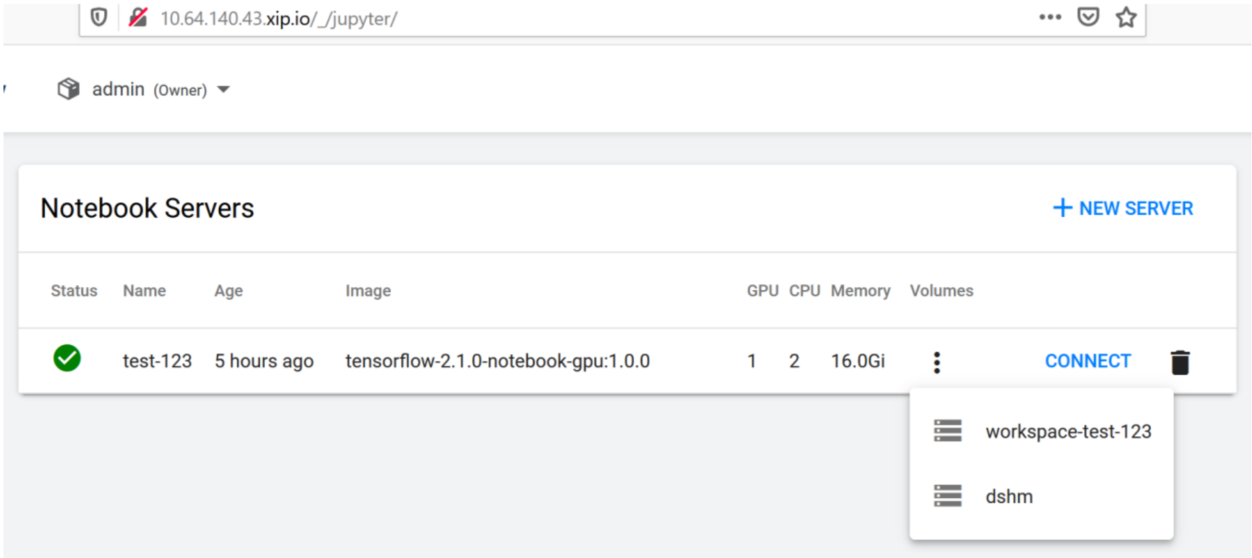
Specify the number and Vendor of GPUs that will be assigned to the Notebook Server's Container.

Number of GPUs: 1

GPU Vendor: NVIDIA

Workspace Volume

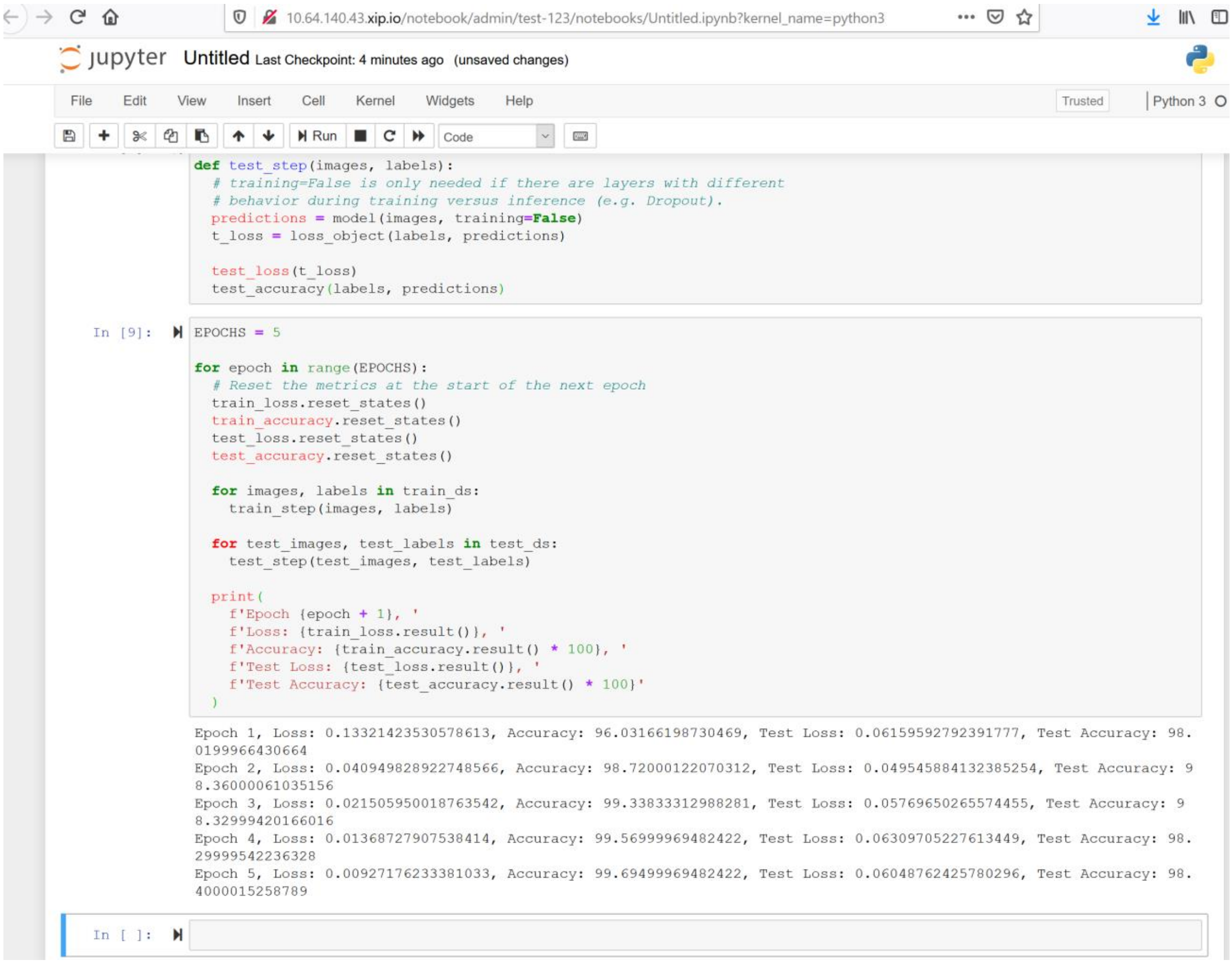
Configure the Volume to be mounted as your personal Workspace.



Notebook Servers

Status	Name	Age	Image	GPU	CPU	Memory	Volumes		
✓	test-123	5 hours ago	tensorflow-2.1.0-notebook-gpu:1.0.0	1	2	16.0Gi	⋮	CONNECT	🗑

Volumes:
- workspace-test-123
- dshm



10.64.140.43.xip.io/notebook/admin/test-123/notebooks/Untitled.ipynb?kernel_name=python3
jupyter
Untitled Last Checkpoint: 4 minutes ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_accuracy(labels, predictions)
```

In [9]:

```
EPOCHS = 5

for epoch in range(EPOCHS):
  # Reset the metrics at the start of the next epoch
  train_loss.reset_states()
  train_accuracy.reset_states()
  test_loss.reset_states()
  test_accuracy.reset_states()

  for images, labels in train_ds:
    train_step(images, labels)

  for test_images, test_labels in test_ds:
    test_step(test_images, test_labels)

  print(
    f'Epoch {epoch + 1}, '
    f'Loss: {train_loss.result()}, '
    f'Accuracy: {train_accuracy.result() * 100}, '
    f'Test Loss: {test_loss.result()}, '
    f'Test Accuracy: {test_accuracy.result() * 100}'
  )
```

```
Epoch 1, Loss: 0.13321423530578613, Accuracy: 96.03166198730469, Test Loss: 0.06159592792391777, Test Accuracy: 98.
0199966430664
Epoch 2, Loss: 0.040949828922748566, Accuracy: 98.72000122070312, Test Loss: 0.049545884132385254, Test Accuracy: 9
8.36000061035156
Epoch 3, Loss: 0.021505950018763542, Accuracy: 99.33833312988281, Test Loss: 0.05769650265574455, Test Accuracy: 9
8.32999420166016
Epoch 4, Loss: 0.01368727907538414, Accuracy: 99.56999969482422, Test Loss: 0.06309705227613449, Test Accuracy: 98.
29999542236328
Epoch 5, Loss: 0.00927176233381033, Accuracy: 99.69499969482422, Test Loss: 0.06048762425780296, Test Accuracy: 98.
4000015258789
```

In [ ]: