TMS Minikube Quickstart Guide

Table of contents

> ⚠ **Attention**
>
> Let's give readers a helpful hint!

In this quickstart guide, we'll set up a single-node Kubernetes cluster with minikube and install TMS onto it for development and testing. We'll also create an NFS model repository on our host machine to load our models from.

The quickstart guide was written for an Ubuntu Linux machine with the bash shell – you may need to make some modifications depending on your dev environment.

Prerequisites

- Docker
- NGC CLI
- Root access to your server
- (Optional) A CUDA capable GPU and NVIDIA GPU Drivers, if deploying GPU models

Create minikube cluster

In order to deploy TMS, we need to have a Kubernetes cluster available to us. TMS works with a wide variety of Kubernetes flavors – in this guide we'll be using minikube, which makes it easy to deploy a single-node cluster for development and testing. If you already have a Kubernetes cluster available to you, you might not need to go through these steps.

> ⓘ **Note**
>
> Note: Installation instructions for third party components are included in this guide for convenience, but we recommend looking at

each tool's linked documentation for the most up-to-date information.

1. Install minikube

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64
sudo dpkg -i minikube_latest_amd64.deb
```

2. Install kubectl

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo curl -fsSLo /etc/apt/keyrings/Kubernetes-archive-keyring.gpg
https://dl.k8s.io/apt/doc/apt-key.gpg
echo "deb [signed-by=/etc/apt/keyrings/Kubernetes-archive-keyring.gpg]
https://apt.Kubernetes.io/ Kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/Kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
```

3. Install Helm

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

4. Start minikube

```
minikube start
```

Create Model Repository

Our next step is to create and fill our model repository to hold the model artifacts that TMS will deploy. Several different kinds of model repositories are available – see the model repository documentation for details.

In this guide, we'll cover two different kinds of model repositories – NFS and HTTP. You'll only need to create one model repository to use TMS. In either case, we'll need to download the models onto our dev machine.

> **Note**
>
> In this example, we're using the image recognition model from the Triton quickstart guide.

```
mkdir -p model_repository/densenet_onnx/1
curl
https://contentmamluswest001.blob.core.windows.net/content/14b2744cf8d6418c87f
1.2.onnx \
-o model_repository/densenet_onnx/1/model.onnx
curl https://raw.githubusercontent.com/triton-inference-
server/server/main/docs/examples/model_repository/densenet_onnx/config.pbtxt \
-o model_repository/densenet_onnx/config.pbtxt
curl https://raw.githubusercontent.com/triton-inference-
server/server/main/docs/examples/model_repository/densenet_onnx/densenet_label
\
-o model_repository/densenet_onnx/densenet_labels.txt
```

You should now have the following directory structure at `./model_repository`:

```
model_repository/
    densenet_onnx
    1
        model.onnx
    config.pbtxt
```

```
densenet_labels.txt
```

HTTP Model Repository

To serve a model repository over HTTP, all we need to do is create a zip file for each model and run an HTTP server.

1. Zip model

```
cd model_repository
zip -r densenet_onnx densenet_onnx
```

2. Serve over HTTP. In a separate terminal, run the following

> **Note**
>
> You can use any HTTP File server. Python's `http.server` defaults to port 8000.

```
python -m http.server --directory .
```

NFS Model Repository

NFS Model repositories for TMS have the same structure as Triton model repositories. You can check out the Triton documentation to learn more.

1. Move model repository to directory for sharing

```
sudo cp -r model_repository /srv/model_repository
```

2. Enable NFS

```
sudo apt install nfs-kernel-server
```

```
sudo systemctl start nfs-kernel-server.service
```

3. Export NFS Share
    1. Add the following line to the file `/etc/exports`:

    ```
    /srv/model_repository *(rw,sync,no_subtree_check)
    ```

    2. Then, execute the following command

    ```
    sudo exportfs -arv
    ```

4. Create Kubernetes storage resources. We'll need a `PersistentVolume` to expose our NFS share to the cluster, and a `PersistentVolumeClaim` to allow Triton pods to mount it.

```
my_nfs_server = <nfs server IP address>
```

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: PersistentVolume
metadata:
name: repo0
spec:
capacity:
storage: 2Gi
accessModes:
- ReadWriteMany
nfs:
server: $my_nfs_server
path: "/srv/model_repository"
mountOptions:
- nfsvers=4.2
EOF
```

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: repo0-claim
spec:
accessModes:
- ReadWriteMany
storageClassName: ""
resources:
requests:
storage: 1Gi
volumeName: repo0
EOF
```

Deploy TMS

TMS is deployed with a Helm chart. To deploy it, we'll need to install Helm, configure access to the chart on NGC, and modify the deployment values.

You'll need your NGC API Key, which can be found here.

```
export NGC_CLI_API_KEY=<your key>
```

1. Add TMS Helm repository

```
helm repo add tms-helm https://helm.ngc.nvidia.com/nvaie --
username=\$oauthtoken --password=$NGC_CLI_API_KEY
```

2. Add container pull secret

```
kubectl delete secret ngc-container-pull
kubectl create secret docker-registry ngc-container-pull \
```

```
--docker-server=nvcr.io --docker-username='$oauthtoken' --docker-
password=$NGC_CLI_API_KEY
```

3. Install TMS. We'll need to use different deployment parameters depending on the model repository type.

    1. HTTP Model Repository

    ```
    helm install tms tms-helm/triton-management-service \
    --set images.secrets={"ngc-container-pull"} \
    --set server.apiService.type="NodePort"
    ```

    2. NFS Model Repository

    ```
    helm install tms tms-helm/triton-management-service \
    --set images.secrets={"ngc-container-pull"} \
    --set server.apiService.type="NodePort" \
    --set "server.modelRepositories.volumes[0].repositoryName=modelrepo"
    \
    --set "server.modelRepositories.volumes[0].volumeClaimName=repo0-
    claim"
    ```

Create Your First Lease

To deploy a model, we need to create a `lease` with TMS. This lease will include a unique identifier for the model(s) you want to deploy, along with some associated metadata. See the `tmsctl` for all of the available lease options.

1. Download `tmsctl`. Check the NGC Console to ensure you're getting the latest version. `tmsctl` is the command line tool for managing TMS.

```
ngc registry resource download-version "nvaie/triton-management-service-
control:v1.4.0"
unzip triton-management-service-control_v1.4.0/tmsctl.zip
```

2. Set `tmsctl` target. By doing this, you won't need to specify the TMS URL in future commands.

```
tms_url=`minikube service tms --url`
./tmsctl target add --force --set-default tms $tms_url
```

3. Make lease creation request. With this, TMS will download the model you specify and create a Triton deployment that serves this model. Depending on the kind of model repository you used, the model URI might be different.

    1. HTTP Model Repository

    ```
    ./tmsctl lease create -m
    "name=densenet_onnx,uri=http://host.minikube.internal:8000/densenet_o
    --triton-resources gpu=0
    ```

    2. NFS Model Repository

    ```
    ./tmsctl lease create -m
    "name=densenet_onnx,uri=model://modelrepo/densenet_onnx" --triton-
    resources gpu=0
    ```

> **Note**
>
> Depending on your network speed, this command may time out due to minikube setting a low threshold for the time it allows for pulling images. If that occurs, you can pull the images manually with `minikube ssh docker image pull` for the respective `tritonserver` and `triton-management-sidecar` images.

4. Add lease name

By default, TMS assigns a random URL to the created Triton server. To make it easier to address the models from other applications, we can choose a specific name to attach to the lease and use that as part of the URL.

Let's first get the Lease ID of the lease we just created:

```
lease_id=`./tmsctl lease list -z | grep -oP 'lease:\K[^\s]+'`
```

Then use that to add a new name to it:

```
./tmsctl lease name create test-lease $lease_id
```

Now we can use the url `test-lease.default.svc.cluster.local:8001` within the cluster to address this lease. Note that if you installed TMS into a namespace other than `default`, you should replace that part of the URL with the namespace you are using.

Make a Triton Request

To simplify the networking, we'll be making the Triton request from within the Kubernetes cluster. So

```
kubectl run -it --rm triton-client --image=nvcr.io/nvidia/tritonserver:23.03-py3-sdk
```

```
/workspace/install/bin/image_client -m densenet_onnx -c 3 -s INCEPTION -i grpc
/workspace/images/mug.jpg -u test-lease.default.svc.cluster.local:8001
```

Congrats! You've successfully deployed and served a model with TMS!

Clean-up

To delete the Triton deployments created by TMS, you can use the `lease list` command to find all existing leases, and the `lease release` command to delete them.

```
tmsctl lease release <lease-id>
```

To remove TMS you can uninstall with Helm

```
helm uninstall tms
```

To tear down your minikube cluster, you can use

```
minikube stop
```