Triton Management Service Deployment Guide

Table of contents

> ⚠ **Attention**
>
> NVIDIA Triton Management Service (TMS) will reach the end of life on July 31, 2024. The version 1.4.0 is the last release.

Triton Management Service (TMS) is a Kubernetes microservice, and expects to be deployed into a Kubernetes managed cluster. To more easily facilitate its deployment into your Kubernetes cluster, TMS provides a Helm chart designed to simplify the deployment, or installation, process.

In order to deploy TMS the `helm` tool (download) and the TMS Helm chart (download must be installed on the local system. Additionally, the local user will require cluster administrator privileges.

TMS Pre-deployment Configuration

Preparing Your Cluster

In order to run TMS, you will need a properly-configured Kubernetes cluster. Depending on which TMS features you wish to leverage and whether you plan to run inference on GPUs, you will need to install some additional dependencies over a default installation.

As a baseline, production TMS installations are recommended to have at least two nodes – one on which to run the API server and database, and one on which to run inference. Typical deployments will have many nodes on which to run inference. One important note about the inference nodes is that they need to be able to run large container images. The default images for Triton can exceed over fourteen gigabytes, so make sure your cluster is properly configured to handle that (also, be prepared for Triton to take a bit of time the first time it starts on each node, as it can take some time for the image to transfer).

If you will be running inference on GPUs, you need to ensure that your inference nodes properly recognize the GPUs and list them as resources. You can check whether this is the case by running `kubectl describe node $NODE_NAME` and seeing whether there is an entry with a key of `nvidia.com/gpu` in the `Capacity` and `Allocatable` sections. If your

cluster is not already properly configured, please see the documentation for the GPU operator or your cloud service provider.

If your deployment requires the autoscaling feature, please see the autoscaling section below.

For the specifics about the versions of Kubernetes and other tools with which TMS was tested, please see the release notes for the version of TMS your are deploying.

Obtaining TMS Helm Chart

The TMS Helm chart can be downloaded from NVIDIA NGC. To do so, use the following command:

```
helm fetch https://helm.ngc.nvidia.com/nvaie/charts/triton-management-service-1.4.0.tgz --username='$oauthtoken' --password=<YOUR API KEY>
```

Extracting the `values.yaml` file from the downloaded chart's TAR file is easy. To do so, use the following command:

```
helm show values triton-management-service-1.4.0.tgz > values.yaml
```

This will create a `values.yaml` file the current directory, which can modified to meet deployment needs.

> **Note**
>
> Download TMS Helm Chart from NGC

See Helm Chart Values for a listing of the configurable values.

Configuring the API Server Pod

By default, TMS requests minimal CPU and memory resources from Kubernetes to run the pod containing the API server and database. While this works fine for initial testing of

TMS's features and for smaller, more stable deployments, it is likely to be insufficient if many clients are expected to be making concurrent API calls. In that situation, it is highly recommended that system administrators change the default settings.

To change the default settings, use the configuration options in `server.resources` in the `values.yaml` file. The amount of CPU and memory resources is relatively low compared to that of the database. For that reason, it is recommended that initially the database be allocated 75% of the available resources, and the API server the other 25%. Below is a sample configuration which would do this on a node with 8 CPUs and 16Gi of memory.

```
resources:
apiServer:
cpu: 2
memory: 4Gi
database:
cpu: 6
memory: 12Gi
```

Kubernetes Secrets

Setting up secrets in Kubernetes for TMS is fairly straightforward, and we'll cover the basics here.

Note that creation of Kubernetes secrets requires sufficient cluster privileges, and therefore might, if you lack sufficient privileges, require a cluster administrator to create them on your behalf.

Container Pull Secrets

TMS Helm chart will include any secrets listed under `values.yaml#images.secrets`. The default `values.yaml` file contains an example secret named "ngc-container-pull".

To create an image-pull secret, use:

```
kubectl create secret docker-registry <secret-name> --docker-server=<docker-server-urn> --docker-username=<username> --docker-password=<password>
```

Then which ever value was chosen for `<secret-name>` add to the `values.yaml#images.secrets` list.

Configuring Model Repositories

To connect to a model repository, see the model repository page.

Configuring Autoscaling

To enable and configure autoscaling, see the separate autoscaling configuration guide.

Configuring Triton Containers

TMS allows the TMS administrator to configure some aspect of the containers that will be created for Triton instances. These can be configured via the top-level `triton` object in `values.yaml`.

Currently, only resource constraints are specified in this section. These are all listed under `resources`. TMS admins may specify both the `default` resources that Triton containers will get, as well as the `limits.maximum` values that users may request on a per-lease basis.

A sample configuration is shown below.

```
triton:
resources:
default:
cpu: 2
gpu: 1
systemMemory: 4Gi
sharedMemory: 256Mi
limits:
minimum:
cpu: 1
gpu: 1
systemMemory: 1Gi
sharedMemory: 128Mi
maximum:
```

```
cpu: 4
gpu: 2
systemMemory: 8Gi
sharedMemory: 512Mi
```

The fields in both `default`, `minimum` and `maximum` sections are defined as follows.

Each value in the `maximum` section must be at least as large as the `default` and `minimum` value.

Each value in the `minimum` section must be smaller than the `default` and `maximum` value.

- `cpu`: The number of whole or factional CPUs assigned to Triton. Can be specified either a number of cores (e.g. `4`), or a number followed by `m`, which represents milli-CPUs (e.g. `1500m`).
    - Minimum value: `1` (or `1000m`).
    - Default: `2`
- `gpu`: The number of whole GPUs assigned to Triton. Must be a whole number – GPUs cannot be fractionally assigned.
    - Minimum value: `0`
    - Default: `1`
- `repositorySize`: The amount of disk space allocated for Triton model repository, as a number plus units (e.g. `4Gi`).
    - Units allowed: `Mi`, `Gi`, `Ti`
    - Minimum value: `256Mi`
    - Default: `2Gi`

- `systemMemory`: The amount of system memory, as a number plus units (e.g. `4Gi`).
    - Units allowed: `Ki`, `Mi`, `Gi`, `Ti`
    - Minimum value: `256Mi`, and at least `128Mi` more than `sharedMemory`.
    - Default: `4Gi`
- `sharedMemory`: The amount of shared memory, as number plus units (same units as `memory`).
    - Minimum value: `32Mi`
    - Default: `256Mi`
    - *Note*: Some backends (e.g. PyTorch) allow the user to use shared memory to allocate tensors.

      If you plan on using this, make sure you set a higher value.

Configuring Persisted Database

To enable and configure TMS to persist database contents, a volume claim bounded to a sizeable kuberenetes persistent volume must be provided to `values.yaml#server.databaseStorage.volumeClaimName`.

In the case of server failure or restart, TMS will be able to reload the contents of the database from this volume.

It should be noted that server performance can be affected by slow or unreliable storage solutions used for the persisted volume.

TMS Deployment Using Helm

Assuming you've followed the steps above, and downloaded the TMS Helm chart, exported its `values.yaml` file, and modified it as necessary, use the following command to install (aka deploy) TMS:

```
helm install <name-of-tms-installation> -f values.yaml triton-management-service-1.0.tgz
```

Security Considerations

The Kubernetes cluster where TMS is installed should be properly secured according to best practices and the security posture of your organization.

Any additional, optional services connected to TMS such as Prometheus and Prometheus adapter should also be secured. We recommend the cluster administrator properly secure access to any S3 or other external model repositories which TMS will utilize. We reccomend leverating encryption in transit and at rest, scoping access to cluster resources following the principle of least privilege, as well as configuring audit logging for your cluster.

TMS default configuration does not allow connections from outside of the Kubernetes cluster. The user assumes responsibility for securing any external connections when changing the default configuration values.

Useful Links & Additional Resources

- NVIDIA GPU Cloud
- Kubernetes
    - Secrets
- Helm
    - Download & Installation
    - Commands
    - Charts
- Triton User Guide