Model Repositories

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.

Model repositories hold the model artifacts that will be loaded into and served by the deployed Triton Inference Servers. Model repositories for Triton Management Service are similar in structure and content to Triton Inference Server model repositories, but there are different options and configurations for the available locations.

In general, model repositories are configured by specifying the remote location of the repository (where the method of specifying the location is dependent on the type of repository) as well as a *Repository Name* when you deploy TMS. TMS operations requiring references to the model repository (i.e. lease creation requests) will use the configured Repository Names. Several different types of model repository are available.

HTTP(S)

TMS Configuration

HTTP(S) model repositories are not required to be pre-specified in the TMS `values.yaml` file. However, you can associate Kubernetes Secret with a particular HTTP url in the `values.yaml` file, in which case TMS will provide the contents of the secret in the `Authorization` request header:

```
# values.yaml
server:
modelRepositories:
https:
- secretName: Name of the Kubernetes secret to read and provide as a
Authorization header for download requests.
targetUri: URL of the remote web-sever in \<domain_label_or_ip_address\>/\<path\>
format, used to determine if secrets apply to a model request or not.
```

The default `values.yaml` file contains an example secret named “ngc-model-pull”.

The `targetUri` is used to determine which secret is best suited to be used to download a given model based on the model’s URN. URN matching is broken up into two parts:

1. Match the DNS label right to left, or absolute match of an IP Address. For example: `models.company.com` would match `cdn.models.company.com`, but would not match `models.cdn.company.com`.

2. Match the path portion of URN from left-to-right. For example: `internal-cdn/repository` would match `internal-cdn/repository/ai_models`, but would not match `internal-cdn/ai_models/repository`.

To create a model-pull secret, use:

```
kubectl create secret generic <secret-name> --from-file <secret-name>
```

Then which ever value was chosen for `&lt;secret-name&gt;` add to the `values.yaml#server.modelRepositories.https` list with the corresponding `targetUri` value.

Setting up the Repository

Models in HTTP(S) repositories should be zipped versions of the directories in a Triton Model Repository, served by some kind of web server and accessible through HTTP GET requests.

For example, if the triton model repository is structured as follows:

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

Then you should serve a file `my_model.zip` that contains one of the following file layouts:

1. ```
   $ unzip -l my_model.zip
   Archive: my_model.zip
   Length Date Time Name
   --------- ---------- ----- ----
   0 2022-04-28 22:27 1/
   356 2022-04-28 22:27 1/model.onnx
   59 2022-06-01 21:12 config.pbtxt
   --------- -------
   415 3 files
   ```
2. ```
   $ unzip -l my_model.zip
   Archive: my_model.zip
   Length Date Time Name
   --------- ---------- ----- ----
   0 2022-07-08 00:23 my_model/
   59 2022-06-01 21:12 my_model/config.pbtxt
   0 2022-04-28 22:27 my_model/1/
   356 2022-04-28 22:27 my_model/1/model.onnx
   --------- -------
   415 4 files
   ```

The `my_model.zip` file – and any other zip files with a similar structure – can be served by a wide variety of web server. One approach is to use the `http.server` module in the Python standard library. In a directory containing the zip file, execute the command

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

This will serve the model with a URI `http://localhost:8000/my_models.zip`.

Model URI

To refer to a model in an HTTP(S) repository, use the full URL of the server. For example:

```
tmsctl lease create -t ${tms_address} -m
```

```
"name=my_model,uri=http://www.example.com/models/my_model.zip"
```

Persistent Volume Claim

TMS Configuration

TMS enables TMS administrators to provide model repositories from Kubernetes Persistent Volume Claims for requested Triton instances.

To enable requested Triton instances to load models from a persistent volume claim, provide the name of the particular Kubernetes persistent volume claim in an entry under `values.yaml#server.modelRepositories.volumes`, along with a valid name for the repository. The Persistent Volume Claim will then be mounted as a volume onto any Triton pod launched by TMS.

```
# values.yaml
server:
modelRepositories:
volumes:
# Name used to reference this model repository as part of lease acquisition.
# May contain only lowercase alphanumeric characters (without spaces, hyphens `-` are permitted).
- repositoryName: volume-models
# Kubernetes persistent volume claim (pvc) used to fetch models.
volumeClaimName: example-volume-claim
```

Setting up the Repository

Persistent Volumes in Kubernetes are cluster resources that can be consumed – like a node, while a Persistent Volume Claim is particular request to use that resource – like a pod. Since model repositories in TMS are used by multiple Triton instances, you'll need to create a specific PVC for your repository that can then be mounted onto multiple pods.

One way to set up the repository is to create the model repository outside of kubernetes in a piece of storage that can be consumed as a Persistent Volume. Then, you can define that piece of storage as a Persistent Volume, and then attach a Persistent Volume Claim to it that allows kubernetes pods to consume that particular piece of storage. The NFS Model Repository path in the quickstart guide gives an example of this.

Persistent Volume Claims are generally exposed directly as file systems, so to create a model repository you can use the same structure as a Triton Inference Server model repositories – for example:

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

See the following resources for creating Persistent Volumes and Claims backed by various types of storage:

- NFS: TMS Quickstart Guide
- AWS Elastic Block Storage: AWS Documentation. Only supported on Amazon EKS clusters.
- Azure Blob Storage: Azure Documentation. Only supported on Azure Kubernetes Service clusters.
- Azure Files: Azure Documentation. Only supported on Azure Kubernetes Service clusters.

Model URI

To refer to a model in a PVC repository, prefix the model name with `model://` and the name of the model repository configured in the `values.yaml` file. For example:

```
tmsctl lease create -t ${tms_address} -m "name=my_model,uri=model://volume-models/my_model"
```

S3 Object Store

TMS Configuration

To configure access to an S3 compatible object store, you must specify a Repository Name, a Bucket Name, and an S3 service Endpoint.

```
#values.yaml
server:
modelRepositories:
s3:
# Name used to reference this model repository as part of lease acquisition.
# May contain only lowercase alphanumeric characters (without spaces, hyphens `-` are permitted).
- repositoryName: repo0
# Name of the S3 bucket used to fetch models.
bucketName: tms-models
# Service URL of the S3 bucket.
# If both 'endpoint' and 'awsRegion' fields are specified, TMS will default to using the value from 'endpoint'.
# Must be a valid URL designating to an existing endpoint (eg. "http:/s3.us-west-2.amazonaws.com" or "http:/play.min.io:9000").
# Refer here to learn more: https://docs.aws.amazon.com/general/latest/gr/s3.html#amazon_s3_website_endpoints.
endpoint: "https://s3.us-west-2.amazonaws.com"
```

If your S3 Object Store is an actual AWS S3 bucket, you can provide the AWS Region of your bucket instead of the explicit endpoint

```
#values.yaml
server:
modelRepositories:
s3:
- repositoryName: repo0
bucketName: tms-models
# Service region code of AWS S3 Bucket.
# Field is for S3 buckets exclusively deployed through AWS.
# Non-AWS S3 Buckets should be configured through the `endpoint` field.
# Must be a valid code designating to existing AWS region (eg. "us-west-2").
# Refer here to learn more: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html).
```

```
awsRegion: "us-west-2"
```

If your model repository is in a private S3 bucket that requires access credentials, you have two options.

First, you can create a Kubernetes Secrets containing an *access key ID* and one containing a *secret access key* that represent the authority to list and retrieve the objects in the bucket. Then, you specify those secrets in the `values.yaml` file:

```
#values.yaml
server:
modelRepositories:
S3:
- repositoryName: repo0
bucketName: tms-models
endpoint: "https://s3.us-west-2.amazonaws.com"
# Name of the Kubernetes secret to read and provide as the access key ID to download objects from the S3 bucket.
# Optional value when IAM or default AWS environment variables are not used for authorizing TMS to read from an S3 bucket.
accessKey: "access-key-secret-name"
# Name of the Kubernetes secret containing the secret access key to read from the S3 bucket
# Optional value when IAM or default AWS environment variables are not used for authorizing TMS to read from an S3 bucket.
secretKey: "secret-key-secret-name"
```

The other option is applicable when you are using AWS S3 buckets and when TMS is to be deployed on EKS. Instead of explicitly providing the key ID and secret key, you must associate an AWS IAM role which has `s3:ListBucket` and `s3:GetObject` permissions for that bucket with the TMS kubernetes service account. You can do this by providing the Amazon Resource Name of that IAM role in the `values.yaml` file.

```
#values.yaml
server:
security:
```

```
aws:
  # AWS IAM role used read models S3 buckets configured in `modelRepositories.S3`.
  role: arn:aws:iam::00000000:role/Tms-s3-role
```

You should also ensure that the role you provide here has a trust policy that allows the `tms-triton` service account to assume that role. For example, you can create this IAM role with the following `eksctl` command:

```
eksctl create iamserviceaccount --cluster tms-cluster --name=tms-server --attach-policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --role-only --role-name=Tms-s3-role --approve
```

Please see the documentation on Configuring a Kubernetes service account to assume an IAM role to learn more.

Setting up the Repository

S3 model repositories should be set organized into folders that are similar to the following structure:

```
tms-models #bucket name
    my_model #S3 folder
    1
      model.onnx
    config.pbtxt
```

All model folders (like the `my_model` folder above) should either be at the top level of your bucket, or contained in a single parent directory. If your model repository is not at the top level folder of your bucket you should include the full path when referring to the model in `lease` commands.

You should also ensure that you have an IAM role available that has access to the bucket (and folder) containing the models or that the bucket is publicly accessible.

Model URI

To refer to a model in an S3 repository, prefix the model name with `model://` and the name of the model repository configured in the `values.yaml` file. TMS will internally resolve this to the correct S3 url. For example:

```
tmsctl lease create -t ${tms_address} -m "name=my_model,uri=model://aws-models/my_model"
```