Triton Pools & Quota Base Shared Tritons

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 Pools enable TMS administrators to create a set of Triton instances which can be shared by any leases created and assigned to the pool. Multiple pools can exist simultaneously with each pool having its own definition and purpose.

Pool definitions allow for the specification of the container image used to deploy Triton instances, and the specification of the resources reserved for and assigned assigned to each TIS instance present in the pool. In addition, a pool definition includes a minimum and maximum pool size (aka. number of concurrent Triton instances the pool supports), and a per-instance quota value used by TMS to determine which Triton instances are best candidates for new leases to be assigned to.

Note: Triton Pools work best in clusters with homogeneous GPUs. TMS does not take GPU SKU into consideration when determining the capacity of Triton instances.

Triton Pool Options

Name

Triton Pools must be given a name. A pool’s name must be unique among all other existing pools. A name can be reused once the name’s previous pool has been deleted. This name is used to identify the pool when creating leases, or interacting with the pool.

Per-Instance Quota

Triton Pools must have a per-instance quota value. The quota value is used to determine which Triton instances have available “space” for new leases to be assigned to.

The meaning of the units the quota is defined as is determined by the pool’s creator. TMS applies no specific meaning to the value.

For example, a pool could be created in a cluster with GPUs which all have 40Gi of memory. The TMS administrator could then decide that each Triton instance should be assigned a per-instance quota value of 40, and expect that all leases deployed into the pool specify the amount of GPU memory they require in gigabytes.

In the above case, TMS would only compare a lease's quota request against any available quota on Triton instances in the pool. TMS would not inspect Triton or the GPU to determine the actual amount of available GPU memory. There is no enforcement of quota values at runtime.

For example, it might be known ahead of time that each Triton instance in a pool is capable of hosting four leases. Therefore, the pool could be created with a per-instance quota value of 4, and each lease would specify a quota need value of 1.

In the above case, each unit of per-instance quota is equal to a single "computing slot" and each lease would consume one, or more, of them.

Instance Limits

Triton Pools are defined with a **minimum** and **maximum** number of Triton instances they're allowed to create and host. When the minimum value is greater than zero, the pool will attempt to always have at least that number of Triton instances available.

These limits are used to determine when a pool scales the number of Triton instances present in the pool. When a lease is created and assigned to the pool and there are no available Triton instances with sufficient available quota to host the lease, the pool will attempt to add a new Triton instance only if it has not already reached it maximum capacity. When a pool has insufficient capacity and cannot add a new Triton instance, any lease creation attempt will be rejected.

Note: When attempting to create new Triton instances, pools are limited by available cluster resources.

Triton Definition

Triton Pool definitions include a definition for each Triton instance in the pool. Triton definitions include the Triton container image used, the number of logical CPU cores, the number of GPUs, and the amount of memory reserved and assigned to each Triton instance created by the pool.

Enforcement of Triton Backend Uniqueness

Triton Pool definitions include an option to not enforce that each Triton instance be restricted to the Triton backends used by the first lease deployed on it. Enforcement is enabled by default because the mixing of Triton backends is discouraged due to issues with memory management.

For example, a Triton instance is deployed with enforcement enabled. The first least deployed to the instance is an ensemble of two models; the first is a TensorFlow model, the second is a PyTorch model. The TensorFlow and PyTorch backends will each allocate as much memory as possible, effectively splitting the available memory between them.

From this point on, because enforcement is enabled, only leases which depend on the TensorFlow and/or PyTorch backends will be deployed to this Triton instance. When a second lease is deployed to this Triton instance, it will only contain models with backends meeting this requirement.

The first time TMS encounters a model, its backend is considered *unknown* and therefore cannot be assigned to any existing Triton instance. Once deployed, TMS will learn which Triton backend the model depends on and will update its record of the Triton instance to reflect the correct mix of backends active on the instance. Additionally, TMS will record the Triton backend information of the model such that all future deployments of the same model will be able to correctly select which Triton instances match the model's Triton backend requirements.

When enforcement is disabled, TMS will select Triton instances without taking into consideration for model backends and which Triton backends are active on instances. This simplifies instance selection, but incurs the risk of attempting to load a model with a backend that's not present on a Triton instance and the instance having insufficient memory available to the load the model.

Enabling enforcement is recommended unless extensive testing with a restricted set of models has been done to ensure Triton instance stability.

Quota Based Shared Triton Leases

Quota based shared Triton (QBST) leases are defined as one or more models with a specified quota consumption value and assigned to a Triton Pool. The specified quota consumption value, or quota, is used to determine how the lease's models will be hosted. Leases with multiple models will always have all of its models hosted by a single Triton instance.

Quota

The **quota** value of a QBST lease defines the amount of "space" or resources the lease will consume. The units or meaning of the value the quota is defined as is determined by the pool's creator. TMS applies no specific meaning to the value.

For example, a Triton Pool might define units in terms of "compute fraction" with each Triton instance being assigned a denominator value. For this example, we'll assume each Triton instance is assigned a quota capacity of 8. Any lease assigned to this pool must specify what fraction of a while Triton instance the lease will consume. This is done by the lease's quota value.

Continuing with the example, lease could be created which expects to consume a quarter of the capacity of a Triton instance would be assigned a quota value of 2. This lease could share the Triton instance with any combination of other leases whose quota sum is less than or equal to 6 (the remaining quota capacity).

It is up to the pool's administrator to determine the units and meaning of a pool's quota, and the measures by which a lease is expected to determine the amount quota it will consume.

TMS uses lease and pool quota values to determine how and where to place leases within a pool. TMS does not enforce any kind of resource utilization after a lease has been assigned to a Triton instance.

Extending the example above, a lease creator specifies that their lease consumes 2 quota units. In actuality the lease consumes 8 quota units (i.e. an entire Triton instance). Because the lease consumes significantly more resources than advertised, several of the loaded models, including models loaded for other leases, experience significant performance degradation and out of memory errors.

It is important to test the quota consumption values of leases before creating them in a production environment. Undervaluing leases can lead to performance degradation, out of memory errors, and Triton instance instability. Overvaluing leases, while often safer, can leave hardware under utilized and potentially cause capacity issues due external processes being forced to wait for available AI cycles.