# Deploy NVIDIA Triton Inference Server (Automated Deployment)

NetApp Solutions

Kevin Hoke

April 20, 2021

# Table of Contents

# Deploy NVIDIA Triton Inference Server (Automated Deployment)

To set up automated deployment for the Triton Inference Server, complete the following steps:

1. Open a VI editor and create a PVC yaml file `vi pvc-triton-model- repo.yaml`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: triton-pvc  namespace: triton
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: ontap-flexvol
```

2. Create the PVC.

```
kubectl create -f pvc-triton-model-repo.yaml
```

3. Open a VI editor, create a deployment for the Triton Inference Server, and call the file `triton_deployment.yaml`.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-3gpu
  name: triton-3gpu
  namespace: triton
spec:
  ports:
  - name: grpc-trtis-serving
    port: 8001
    targetPort: 8001
  - name: http-trtis-serving
    port: 8000
    targetPort: 8000
  - name: prometheus-metrics
```

```
    port: 8002
    targetPort: 8002
  selector:
    app: triton-3gpu
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-1gpu
  name: triton-1gpu
  namespace: triton
spec:
  ports:
  - name: grpc-trtis-serving
    port: 8001
    targetPort: 8001
  - name: http-trtis-serving
    port: 8000
    targetPort: 8000
  - name: prometheus-metrics
    port: 8002
    targetPort: 8002
  selector:
    app: triton-1gpu
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-3gpu
  name: triton-3gpu
  namespace: triton
spec:
  replicas: 1
  selector:
    matchLabels:
      app: triton-3gpu      version: v1
  template:
    metadata:
      labels:
        app: triton-3gpu
        version: v1
    spec:
      containers:
      - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
        command: ["/bin/sh", "-c"]
        args: ["trtserver --model-store=/mnt/model-repo"]
        imagePullPolicy: IfNotPresent
        name: triton-3gpu
        ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
          requests:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
        volumeMounts:
        - name: triton-model-repo
          mountPath: /mnt/model-repo      nodeSelector:
        gpu-count: “3”
      volumes:
      - name: triton-model-repo
        persistentVolumeClaim:
          claimName: triton-pvc---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-1gpu
  name: triton-1gpu
  namespace: triton
spec:
  replicas: 3
  selector:
    matchLabels:
      app: triton-1gpu
      version: v1
  template:
    metadata:
      labels:
        app: triton-1gpu
        version: v1
    spec:
      containers:
      - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
        command: ["/bin/sh", "-c", “sleep 1000”]
        args: ["trtserver --model-store=/mnt/model-repo"]
        imagePullPolicy: IfNotPresent
        name: triton-1gpu
        ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 1
          requests:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 1
        volumeMounts:
        - name: triton-model-repo
          mountPath: /mnt/model-repo      nodeSelector:
        gpu-count: “1”
      volumes:
      - name: triton-model-repo
        persistentVolumeClaim:
          claimName: triton-pvc
```

Two deployments are created here as an example. The first deployment spins up a pod that uses three GPUs and has replicas set to 1. The other deployment spins up three pods each using one GPU while the replica is set to 3. Depending on your requirements, you can change the GPU allocation and replica counts.

Both of the deployments use the PVC created earlier and this persistent storage is provided to the Triton inference servers as the model repository.

For each deployment, a service of type LoadBalancer is created. The Triton Inference Server can be accessed by using the LoadBalancer IP which is in the application network.

A nodeSelector is used to ensure that both deployments get the required number of GPUs without any issues.

4. Label the K8 worker nodes.

```
kubectl label nodes hci-ai-k8-worker-01 gpu-count=3
kubectl label nodes hci-ai-k8-worker-02 gpu-count=1
```

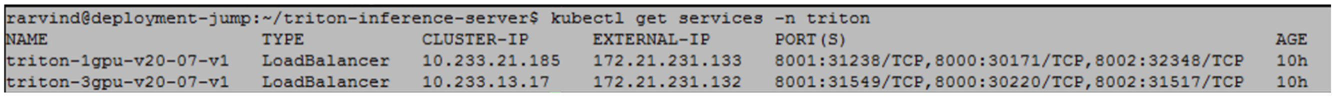
5. Create the deployment.

```
kubectl apply -f triton_deployment.yaml
```

6. Make a note of the LoadBalancer service external LPS.

```
kubectl get services -n triton
```

The expected sample output is as follows:



```
rarvind@deployment-jump:~/triton-inference-server$ kubectl get services -n triton
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                                        AGE
triton-1gpu-v20-07-v1   LoadBalancer   10.233.21.185   172.21.231.133   8001:31238/TCP,8000:30171/TCP,8002:32348/TCP   10h
triton-3gpu-v20-07-v1   LoadBalancer   10.233.13.17    172.21.231.132   8001:31549/TCP,8000:30220/TCP,8002:31517/TCP   10h
```

7. Connect to any one of the pods that were created from the deployment.

```
kubectl exec -n triton --stdin --tty triton-1gpu-86c4c8dd64-545lx --
/bin/bash
```

8. Set up the model repository by using the example model repository.

```
git clone
cd triton-inference-server
git checkout r20.07
```

9. Fetch any missing model definition files.

```
cd docs/examples
./fetch_models.sh
```

10. Copy all the models to the model repository location or just a specific model that you wish to use.

```
cp -r model_repository/resnet50_netdef/ /mnt/model-repo/
```

In this solution, only the resnet50_netdef model is copied over to the model repository as an example.

11. Check the status of the Triton Inference Server.

```
curl -v <<LoadBalancer_IP_recorded earlier>>:8000/api/status
```

The expected sample output is as follows:

```
curl -v 172.21.231.132:8000/api/status
*   Trying 172.21.231.132...
* TCP_NODELAY set
* Connected to 172.21.231.132 (172.21.231.132) port 8000 (#0)
> GET /api/status HTTP/1.1
> Host: 172.21.231.132:8000
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< NV-Status: code: SUCCESS server_id: "inference:0" request_id: 9
< Content-Length: 1124
< Content-Type: text/plain
<
id: "inference:0"
version: "1.15.0"
uptime_ns: 377890294368
model_status {
  key: "resnet50_netdef"
  value {
    config {
      name: "resnet50_netdef"
      platform: "caffe2_netdef"
      version_policy {
        latest {
          num_versions: 1
        }
      }
      max_batch_size: 128
      input {
        name: "gpu_0/data"
        data_type: TYPE_FP32
        format: FORMAT_NCHW
        dims: 3
        dims: 224
        dims: 224
      }
      output {
        name: "gpu_0/softmax"
        data_type: TYPE_FP32
        dims: 1000
        label_filename: "resnet50_labels.txt"
      }
      instance_group {
        name: "resnet50_netdef"
        count: 1
        gpus: 0
        gpus: 1
        gpus: 2
        kind: KIND_GPU
      }
      default_model_filename: "model.netdef"
      optimization {
        input_pinned_memory {
          enable: true
        }
        output_pinned_memory {
          enable: true
        }
      }
    }
    version_status {
      key: 1
      value {
        ready_state: MODEL_READY
        ready_state_reason {
        }
      }
    }
  }
}
ready_state: SERVER_READY
* Connection #0 to host 172.21.231.132 left intact
```

Next: Deploy the Client for Triton Inference Server (Automated Deployment)