Table of Contents

Example High-performance Jobs for ONTAP AI Deployments ........................................... 1
Example High-performance Jobs for ONTAP AI Deployments ........................................... 1
Execute a Single-Node AI Workload .................................................................................. 1
Execute a Synchronous Distributed AI Workload .............................................................. 5
Example High-performance Jobs for ONTAP AI Deployments

This section includes examples of various high-performance jobs that can be executed when Kubernetes is deployed on an ONTAP AI pod.


Execute a Single-Node AI Workload

To execute a single-node AI and ML job in your Kubernetes cluster, perform the following tasks from the deployment jump host. With Trident, you can quickly and easily make a data volume, potentially containing petabytes of data, accessible to a Kubernetes workload. To make such a data volume accessible from within a Kubernetes pod, simply specify a PVC in the pod definition. This step is a Kubernetes-native operation; no NetApp expertise is required.

This section assumes that you have already containerized (in the Docker container format) the specific AI and ML workload that you are attempting to execute in your Kubernetes cluster.

1. The following example commands show the creation of a Kubernetes job for a TensorFlow benchmark workload that uses the ImageNet dataset. For more information about the ImageNet dataset, see the ImageNet website.

   This example job requests eight GPUs and therefore can run on a single GPU worker node that features eight or more GPUs. This example job could be submitted in a cluster for which a worker node featuring eight or more GPUs is not present or is currently occupied with another workload. If so, then the job remains in a pending state until such a worker node becomes available.

   Additionally, in order to maximize storage bandwidth, the volume that contains the needed training data is mounted twice within the pod that this job creates. Another volume is also mounted in the pod. This second volume will be used to store results and metrics. These volumes are referenced in the job definition by using the names of the PVCs. For more information about Kubernetes jobs, see the official Kubernetes documentation.

   An emptyDir volume with a medium value of Memory is mounted to /dev/shm in the pod that this example job creates. The default size of the /dev/shm virtual volume that is automatically created by the Docker container runtime can sometimes be insufficient for TensorFlow's needs. Mounting an emptyDir volume as in the following example provides a sufficiently large /dev/shm virtual volume. For more information about emptyDir volumes, see the official Kubernetes documentation.
The single container that is specified in this example job definition is given a `securityContext > privileged` value of true. This value means that the container effectively has root access on the host. This annotation is used in this case because the specific workload that is being executed requires root access. Specifically, a clear cache operation that the workload performs requires root access. Whether or not this `privileged: true` annotation is necessary depends on the requirements of the specific workload that you are executing.

```bash
$ cat << EOF > ./netapp-tensorflow-single-imagenet.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-single-imagenet
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--dataset_dir=/mnt/mount_0/dataset/imagenet", "--dgx_version=dgx1", "--num_devices=8"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
```


2. Confirm that the job that you created in step 1 is running correctly. The following example command confirms that a single pod was created for the job, as specified in the job definition, and that this pod is currently running on one of the GPU worker nodes.

```bash
$ kubectl get pods -o wide
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>netapp-tensorflow-single-imagenet-m7x92</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td>3m 10.233.68.61 10.61.218.154 &lt;none&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Confirm that the job that you created in step 1 completes successfully. The following example commands confirm that the job completed successfully.

```bash
$ kubectl get jobs
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>COMPLETIONS</th>
<th>DURATION</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>netapp-tensorflow-single-imagenet</td>
<td>0/1</td>
<td>24s</td>
<td>24s</td>
</tr>
</tbody>
</table>

EOF
$ kubectl get jobs
NAME                                 COMPLETIONS  DURATION
AGE
netapp-tensorflow-single-imagenet     1/1           5m42s
10m

$ kubectl get pods
NAME                                                  READY   STATUS
RESTARTS    AGE
netapp-tensorflow-single-imagenet-m7x92                0/1     Completed
0           11m

$ kubectl logs netapp-tensorflow-single-imagenet-m7x92
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 702
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 711
Total images/sec = 6530.59125
=============== Clean Cache !!! ==================
mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1 > /proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nondistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers=False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000
--datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu >
/tmp/20190814_105450_tensorflow_horovod_rdma_resnet50_gpu_8_256_b500_imagenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1

4. Optional: Clean up job artifacts. The following example commands show the deletion of the job object that was created in step 1.

When you delete the job object, Kubernetes automatically deletes any associated pods.
$ kubectl get jobs
NAME                                           COMPLETIONS   DURATION
AGE                                            
netapp-tensorflow-single-imagenet               1/1           5m42s
10m
$ kubectl get pods
NAME                                           READY   STATUS
RESTARTS   AGE                                 
netapp-tensorflow-single-imagenet-m7x92         0/1     Completed
0       11m
$ kubectl delete job netapp-tensorflow-single-imagenet
job.batch "netapp-tensorflow-single-imagenet" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.

Next: Execute a Synchronous Distributed Al Workload.

Execute a Synchronous Distributed Al Workload

To execute a synchronous multinode AI and ML job in your Kubernetes cluster, perform the following tasks on the deployment jump host. This process enables you to take advantage of data that is stored on a NetApp volume and to use more GPUs than a single worker node can provide. See the following figure for a depiction of a synchronous distributed AI job.

Synchronous distributed jobs can help increase performance and training accuracy compared with asynchronous distributed jobs. A discussion of the pros and cons of synchronous jobs versus asynchronous jobs is outside the scope of this document.

1. The following example commands show the creation of one worker that participates in the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in the example in the section Execute a Single-Node AI Workload. In this specific example, only a single worker
is deployed because the job is executed across two worker nodes.

This example worker deployment requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature. For more information about Kubernetes deployments, see the official Kubernetes documentation.

A Kubernetes deployment is created in this example because this specific containerized worker would never complete on its own. Therefore, it doesn’t make sense to deploy it by using the Kubernetes job construct. If your worker is designed or written to complete on its own, then it might make sense to use the job construct to deploy your worker.

The pod that is specified in this example deployment specification is given a hostNetwork value of true. This value means that the pod uses the host worker node’s networking stack instead of the virtual networking stack that Kubernetes usually creates for each pod. This annotation is used in this case because the specific workload relies on Open MPI, NCCL, and Horovod to execute the workload in a synchronous distributed manner. Therefore, it requires access to the host networking stack. A discussion about Open MPI, NCCL, and Horovod is outside the scope of this document. Whether or not this hostNetwork: true annotation is necessary depends on the requirements of the specific workload that you are executing. For more information about the hostNetwork field, see the official Kubernetes documentation.

```yaml
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-worker.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapp-tensorflow-multi-imagenet-worker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapp-tensorflow-multi-imagenet-worker
  template:
    metadata:
      labels:
        app: netapp-tensorflow-multi-imagenet-worker
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
        medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
EOF
```
- name: results
  persistentVolumeClaim:
    claimName: tensorflow-results
containers:
- name: netapp-tensorflow-py2
  image: netapp/tensorflow-py2:19.03.0
  command: ['bash', '/netapp/scripts/start-slave-multi.sh', "22122"]
  resources:
    limits:
      nvidia.com/gpu: 8
  volumeMounts:
  - mountPath: /dev/shm
      name: dshm
  - mountPath: /mnt/mount_0
      name: testdata-iface1
  - mountPath: /mnt/mount_1
      name: testdata-iface2
  - mountPath: /tmp
      name: results
  securityContext:
    privileged: true
EOF

$ kubectl create -f ./netapp-tensorflow-multi-imagenet-worker.yaml
deployment.apps/netapp-tensorflow-multi-imagenet-worker created

$ kubectl get deployments
NAME                                      DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
  1           4s

2. Confirm that the worker deployment that you created in step 1 launched successfully. The following example commands confirm that a single worker pod was created for the deployment, as indicated in the deployment definition, and that this pod is currently running on one of the GPU worker nodes.

$ kubectl get pods -o wide
NAME                                                       READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0       60s    10.61.218.154 10.61.218.154 <none>

$ kubectl logs netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725
22122

3. Create a Kubernetes job for a master that kicks off, participates in, and tracks the execution of the
synchronous multinode job. The following example commands create one master that kicks off, participates in, and tracks the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in the example in the section Execute a Single-Node AI Workload.

This example master job requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature.

The master pod that is specified in this example job definition is given a hostNetwork value of true, just as the worker pod was given a hostNetwork value of true in step 1. See step 1 for details about why this value is necessary.

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-master.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-multi-imagenet-master
spec:
  backoffLimit: 5
  template:
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
          dataset_dir=/mnt/mount_0/dataset/imagenet", "--port=22122", "--
          num_devices=16", "--dgx_version=dgx1", "--
          nodes=10.61.218.152,10.61.218.154"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
EOF
```
4. Confirm that the master job that you created in step 3 is running correctly. The following example command confirms that a single master pod was created for the job, as indicated in the job definition, and that this pod is currently running on one of the GPU worker nodes. You should also see that the worker pod that you originally saw in step 1 is still running and that the master and worker pods are running on different nodes.

```
$ kubectl get pods -o wide

NAME                                                        READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
---
netapp-tensorflow-multi-imagenet-master-ppwwj              1/1
Running   0          45s   10.61.218.152   10.61.218.152   <none>
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running   0          26m   10.61.218.154   10.61.218.154   <none>
```

5. Confirm that the master job that you created in step 3 completes successfully. The following example commands confirm that the job completed successfully.

```
$ kubectl get jobs

NAME                                      COMPLETIONS   DURATION   AGE
---
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      9m18s
$ kubectl get pods

NAME                                                        READY
STATUS      RESTARTS   AGE
---
netapp-tensorflow-multi-imagenet-master-ppwwj              0/1
Completed   0          9m38s
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0          35m
```

EOF
6. Delete the worker deployment when you no longer need it. The following example commands show the deletion of the worker deployment object that was created in step 1.

When you delete the worker deployment object, Kubernetes automatically deletes any associated worker pods.
$ kubectl get deployments
NAME                                      DESIRED   CURRENT   UP-TO-DATE
AVAILabe   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
1           43m
$ kubectl get pods
NAME                                      READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj 0/1
Completed   0          17m
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725 1/1
Running     0          43m
$ kubectl delete deployment netapp-tensorflow-multi-imagenet-worker
deployment.extensions "netapp-tensorflow-multi-imagenet-worker" deleted
$ kubectl get deployments
No resources found.
$ kubectl get pods
NAME                                      READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj 0/1
Completed   0          18m

7. **Optional**: Clean up the master job artifacts. The following example commands show the deletion of the master job object that was created in step 3.

When you delete the master job object, Kubernetes automatically deletes any associated master pods.

$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      19m
$ kubectl get pods
NAME                                      READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj 0/1
Completed   0          19m
$ kubectl delete job netapp-tensorflow-multi-imagenet-master
job.batch "netapp-tensorflow-multi-imagenet-master" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.

Next: Performance Testing.
Copyright Information

Copyright © 2022 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.