Ejemplo de trabajos de alto rendimiento para implementaciones AIPod

NetApp Solutions

NetApp
November 12, 2024

Tabla de contenidos

Ejemplo de trabajos de alto rendimiento para implementaciones AIPod

Ejecute una carga de trabajo de IA de un solo nodo

Para ejecutar una tarea DE IA y ML de un solo nodo en el clúster de Kubernetes, realice las siguientes tareas desde el host de puesta en marcha. Con Trident, puede crear de forma rápida y sencilla un volumen de datos, con potencialmente petabytes de datos al que se puede acceder una carga de trabajo de Kubernetes. Para que un volumen de datos de este tipo sea accesible desde un pod de Kubernetes, solo tiene que especificar una RVP en la definición del pod.

> En esta sección se supone que ya ha realizado un contenedor (en el formato de contenedor de Docker) con la carga de trabajo específica DE IA y ML que intenta ejecutar en su clúster de Kubernetes.

1. Los siguientes comandos de ejemplo muestran la creación de un trabajo de Kubernetes para una carga de trabajo de prueba de ImageNET que utiliza el conjunto de datos de TensorFlow. Para obtener más información acerca del conjunto de datos ImageNET, consulte "Sitio web de ImageNET".

   Este trabajo de ejemplo solicita ocho GPU y, por lo tanto, puede ejecutarse en un solo nodo de trabajo de GPU con ocho o más GPU. Este trabajo de ejemplo se puede enviar en un clúster para el que no hay un nodo de trabajo con ocho o más GPU o esté ocupado actualmente con otra carga de trabajo. Si es así, el trabajo permanece en estado pendiente hasta que dicho nodo de trabajo esté disponible.

   Además, para maximizar el ancho de banda de almacenamiento, el volumen que contiene los datos de entrenamiento necesarios se monta dos veces en el pod que crea este trabajo. Otro volumen también se monta en el pod. Este segundo volumen se utilizará para almacenar resultados y métricas. Estos volúmenes se hacen referencia en la definición de trabajo utilizando los nombres de las RVP. Para obtener más información sobre los trabajos de Kubernetes, consulte "Documentación oficial sobre Kubernetes".

   An `emptyDir` volumen con un `medium` valor de `Memory` está montado en `/dev/shm` en el pod que crea este trabajo de ejemplo. El tamaño predeterminado de `/dev/shm` El volumen virtual que se crea automáticamente mediante el tiempo de ejecución del contenedor Docker puede en ocasiones ser insuficiente para las necesidades de TensorFlow. Montaje de un `emptyDir` volumen como en el ejemplo siguiente proporciona un tamaño suficiente `/dev/shm` volumen virtual. Para obtener más información acerca de `emptyDir` volúmenes, consulte "Documentación oficial sobre Kubernetes".

   El contenedor único que se especifica en esta definición de trabajo de ejemplo se proporciona un `securityContext > privileged` valor de `true`. Este valor significa que el contenedor tiene acceso raíz en el host de forma efectiva. Esta anotación se utiliza en este caso porque la carga de trabajo específica que se está ejecutando requiere acceso raíz. Específicamente, una operación de caché clara que ejecuta la carga de trabajo requiere acceso raíz. Si esto o no `privileged: true` la anotación es necesaria depende de los requisitos de la carga de trabajo específica que se esté ejecutando.

```
$ 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
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-single-imagenet.yaml
job.batch/netapp-tensorflow-single-imagenet created
$ kubectl get jobs
NAME                                    COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet       0/1           24s        24s
```

2. Confirme que el trabajo que ha creado en el paso 1 se está ejecutando correctamente. El siguiente comando de ejemplo confirma que se creó un solo pod para el trabajo, tal como se especifica en la definición de trabajos, y que este pod se ejecuta actualmente en uno de los nodos de trabajo de la GPU.

```
$ kubectl get pods -o wide
NAME                                                READY   STATUS
RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-single-imagenet-m7x92             1/1     Running     0
3m    10.233.68.61    10.61.218.154   <none>
```

3. Confirme que el trabajo que ha creado en el paso 1 se ha completado correctamente. Los siguientes comandos de ejemplo confirman que el trabajo se ha completado correctamente.

```
$ 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_be
nchmarks.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
--nodistortions --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_ima
genet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

4. **Opcional:** limpiar artefactos de trabajo. Los siguientes comandos de ejemplo muestran la eliminación del objeto de trabajo creado en el paso 1.

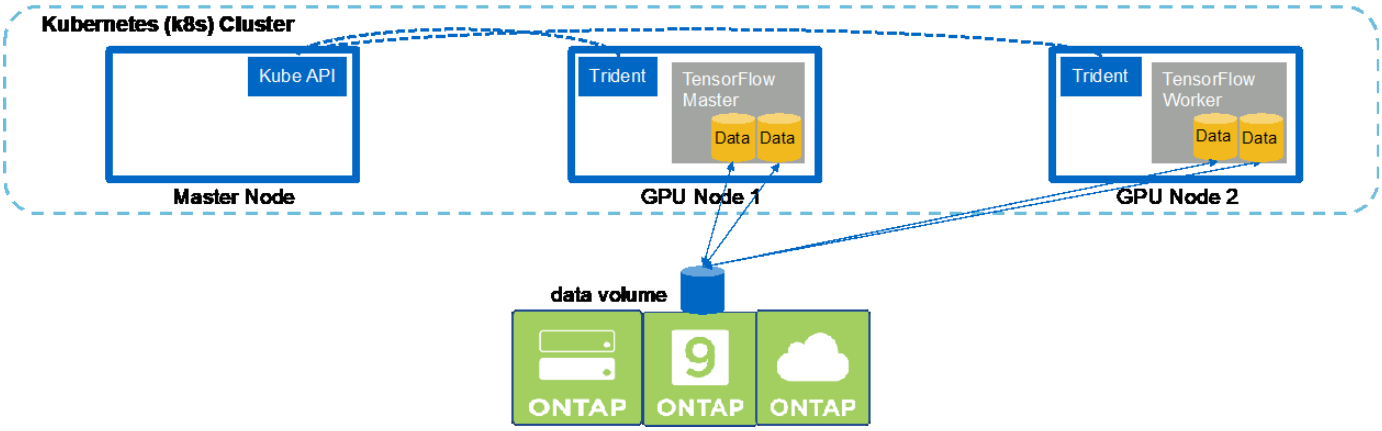
   Cuando se elimina el objeto de trabajo, Kubernetes elimina automáticamente todos los pods asociados.

```
$ 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.
```

Ejecute una carga de trabajo de IA distribuida síncrona

Para ejecutar un trabajo de IA y ML multinodo síncrono en un clúster de Kubernetes, lleve a cabo las siguientes tareas en el host de saltos de la puesta en marcha. Este proceso le permite aprovechar los datos almacenados en un volumen de NetApp y utilizar más GPU de las que puede proporcionar un único nodo de trabajo. Consulte la siguiente figura para obtener una descripción de un trabajo de IA distribuido sincrónica.

Los trabajos distribuidos síncronos pueden ayudar a aumentar el rendimiento y la precisión de la formación en comparación con los trabajos distribuidos de manera asíncrona. Un análisis de los pros y los contras de los trabajos síncronos frente a los trabajos asincrónicos está fuera del alcance de este documento.



1. En los siguientes comandos de ejemplo, se muestra la creación de un trabajador que participa en la ejecución síncrona y distribuida de la misma tarea de prueba de rendimiento TensorFlow que se ejecutó en un solo nodo en el ejemplo de la sección "Ejecute una carga de trabajo de IA de un solo nodo". En este ejemplo específico, sólo se implementa un único trabajador porque el trabajo se ejecuta en dos nodos de

trabajo.

Esta puesta en marcha de trabajo de ejemplo solicita ocho GPU y, por lo tanto, puede ejecutarse en un único nodo de trabajo de GPU con ocho o más GPU. Si los nodos de trabajo de la GPU tienen más de ocho GPU, para maximizar el rendimiento, es posible que desee aumentar este número para ser igual al número de GPU que disponen los nodos de trabajo. Para obtener más información sobre las puestas en marcha de Kubernetes, consulte "Documentación oficial sobre Kubernetes".

En este ejemplo se crea una puesta en marcha de Kubernetes, ya que este trabajador en contenedor específico nunca lo completaría por sí solo. Por lo tanto, no tiene sentido implementarlo usando la construcción de trabajos de Kubernetes. Si su trabajador está diseñado o escrito para completar por sí solo, entonces podría tener sentido utilizar la construcción del trabajo para desplegar a su trabajador.

El POD especificado en esta especificación de implementación de ejemplo recibe una `hostNetwork` valor de `true`. Este valor significa que el pod utiliza la pila de red del nodo de trabajo del host en lugar de la pila de red virtual que Kubernetes suele crear para cada pod. Esta anotación se utiliza en este caso porque la carga de trabajo específica depende de Open MPI, NCCL y Horovod para ejecutar la carga de trabajo de forma síncrona distribuida. Por lo tanto, requiere acceso a la pila de red del host. Un debate sobre Open MPI, NCCL y Horovod está fuera del alcance de este documento. Si esto o no `hostNetwork: true` la anotación es necesaria depende de los requisitos de la carga de trabajo específica que se esté ejecutando. Para obtener más información acerca de `hostNetwork` consulte "Documentación oficial sobre Kubernetes".

```
$ 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
```

```
      - 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. Confirme que el despliegue del trabajador que creó en el paso 1 se inició correctamente. Los siguientes comandos de ejemplo confirman que se creó un solo pod de trabajadores para la implementación, tal y como se indica en la definición de la puesta en marcha, y que este pod se ejecuta actualmente en uno de los nodos de trabajo de la GPU.

```
$ 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. Cree un trabajo de Kubernetes en un maestro que se inicia, participe y realice un seguimiento de la ejecución de un trabajo de varios nodos síncronos. Los siguientes comandos de ejemplo crean un maestro que inicia sesión, participa en y realiza un seguimiento de la ejecución síncrona distribuida de la misma tarea de prueba de rendimiento TensorFlow que se ejecutó en un solo nodo del ejemplo de la sección "Ejecute una carga de trabajo de IA de un solo nodo".

   Este trabajo maestro de ejemplo solicita ocho GPU y, por lo tanto, puede ejecutarse en un único nodo de trabajo de GPU con ocho o más GPU. Si los nodos de trabajo de la GPU tienen más de ocho GPU, para maximizar el rendimiento, es posible que desee aumentar este número para ser igual al número de GPU que disponen los nodos de trabajo.

   El POD maestro especificado en esta definición de trabajo de ejemplo recibe una `hostNetwork` valor de `true`, así como se le dio a la cápsula de trabajo un `hostNetwork` valor de `true` en el paso 1. Consulte el paso 1 para obtener más información acerca de por qué es necesario este valor.

```
$ 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
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-master.yaml
job.batch/netapp-tensorflow-multi-imagenet-master created
$ kubectl get jobs
NAME                                        COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   0/1           25s        25s
```

4. Confirme que el trabajo maestro que creó en el paso 3 se está ejecutando correctamente. El siguiente comando de ejemplo confirma que se creó un único pod maestro para el trabajo, tal como se indica en la definición de trabajos, y que este pod se ejecuta actualmente en uno de los nodos de trabajo de la GPU. También debe ver que el pod de trabajo que originalmente vio en el paso 1 sigue en ejecución y que los pods maestro y trabajador se ejecutan en diferentes nodos.

```
$ 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. Confirme que el trabajo maestro que ha creado en el paso 3 se ha completado correctamente. Los siguientes comandos de ejemplo confirman que el trabajo se ha completado correctamente.

```
$ 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
$ kubectl logs netapp-tensorflow-multi-imagenet-master-ppwwj
[10.61.218.152:00008] WARNING: local probe returned unhandled
shell:unknown assuming bash
rm: cannot remove '/lib': Is a directory
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 702
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 702
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
Total images/sec = 12881.33875
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 2 -H 10.61.218.152:1,10.61.218.154:1 -mca
pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" bash -c 'sync; echo 1 >
/proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 16 -H 10.61.218.152:8,10.61.218.154:8
-bind-to none -map-by slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH
-mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -x
NCCL_IB_HCA=mlx5 -x NCCL_NET_GDR_READ=1 -x NCCL_IB_SL=3 -x
NCCL_IB_GID_INDEX=3 -x
NCCL_SOCKET_IFNAME=enp5s0.3091,enp12s0.3092,enp132s0.3093,enp139s0.3094
-x NCCL_IB_CUDA_SUPPORT=1 -mca orte_base_help_aggregate 0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_be
nchmarks.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
--nodistortions --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_161609_tensorflow_horovod_rdma_resnet50_gpu_16_256_b500_im
agenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

6. Elimine la implementación del trabajador cuando ya no la necesite. Los siguientes comandos de ejemplo muestran la eliminación del objeto de implementación de trabajo que se creó en el paso 1.

Cuando se elimina el objeto de implementación de trabajo, Kubernetes elimina automáticamente todos los pods de trabajador asociados.

```
$ kubectl get deployments
NAME                                        DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   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. **Opcional:** Limpie los artefactos del trabajo maestro. Los siguientes comandos de ejemplo muestran la eliminación del objeto de trabajo maestro que se creó en el paso 3.

   Cuando se elimina el objeto de trabajo maestro, Kubernetes elimina automáticamente todos los pods maestros asociados.

```
$ 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.
```