Esempi di processi ad alte prestazioni per le implementazioni di AIPod

NetApp Solutions

NetApp
May 10, 2024

Sommario

Esempi di processi ad alte prestazioni per le implementazioni di AIPod

Eseguire un carico di lavoro ai a nodo singolo

Per eseguire un processo ai e ML a nodo singolo nel cluster Kubernetes, eseguire le seguenti operazioni dall'host di distribuzione jump. Con Trident, è possibile rendere un volume di dati, potenzialmente contenente petabyte di dati, accessibile a un carico di lavoro Kubernetes in modo rapido e semplice. Per rendere un volume di dati accessibile dall'interno di un pod Kubernetes, è sufficiente specificare un PVC nella definizione del pod.

> In questa sezione si presuppone che sia già stato containerizzato (nel formato Docker Container) il carico di lavoro ai e ML specifico che si sta tentando di eseguire nel cluster Kubernetes.

1. I seguenti comandi di esempio mostrano la creazione di un lavoro Kubernetes per un carico di lavoro di benchmark TensorFlow che utilizza il dataset ImageNet. Per ulteriori informazioni sul set di dati ImageNet, vedere "Sito Web ImageNet".

   Questo processo di esempio richiede otto GPU e quindi può essere eseguito su un singolo nodo di lavoro GPU che dispone di otto o più GPU. Questo job di esempio potrebbe essere inviato in un cluster per il quale un nodo di lavoro con otto o più GPU non è presente o è attualmente occupato con un altro workload. In tal caso, il lavoro rimane in uno stato in sospeso fino a quando tale nodo di lavoro non diventa disponibile.

   Inoltre, per massimizzare la larghezza di banda dello storage, il volume contenente i dati di training necessari viene montato due volte all'interno del pod creato da questo lavoro. Nel pod è montato anche un altro volume. Questo secondo volume verrà utilizzato per memorizzare risultati e metriche. Questi volumi vengono referenziati nella definizione del lavoro utilizzando i nomi dei PVC. Per ulteriori informazioni sui job Kubernetes, consultare "Documentazione ufficiale di Kubernetes".

   An `emptyDir` volume con a. `medium` valore di `Memory` è montato su `/dev/shm` nel pod creato da questo lavoro di esempio. La dimensione predefinita di `/dev/shm` Il volume virtuale creato automaticamente dal runtime del container Docker può talvolta essere insufficiente per le esigenze di TensorFlow. Montaggio di un `emptyDir` il volume come nell'esempio seguente fornisce un volume sufficientemente grande `/dev/shm` volume virtuale. Per ulteriori informazioni su `emptyDir` volumes (volumi), vedere "Documentazione ufficiale di Kubernetes".

   Al singolo contenitore specificato in questa definizione di lavoro di esempio viene assegnato un `securityContext > privileged` valore di `true`. Questo valore significa che il container dispone effettivamente dell'accesso root sull'host. Questa annotazione viene utilizzata in questo caso perché il carico di lavoro specifico che viene eseguito richiede l'accesso root. In particolare, un'operazione di cancellazione della cache eseguita dal carico di lavoro richiede l'accesso root. Che sia o meno così `privileged: true` l'annotazione è necessaria a seconda dei requisiti del carico di lavoro specifico che si sta eseguendo.

```
$ 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. Verificare che il lavoro creato al punto 1 sia in esecuzione correttamente. Il seguente comando di esempio conferma che è stato creato un singolo pod per il lavoro, come specificato nella definizione del lavoro, e che questo pod è attualmente in esecuzione su uno dei nodi di lavoro 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. Verificare che il lavoro creato al passo 1 sia stato completato correttamente. I seguenti comandi di esempio confermano che il lavoro è stato completato correttamente.

```
$ 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. **Opzionale:** eliminare gli artefatti del lavoro. I seguenti comandi di esempio mostrano l'eliminazione dell'oggetto di lavoro creato al passo 1.

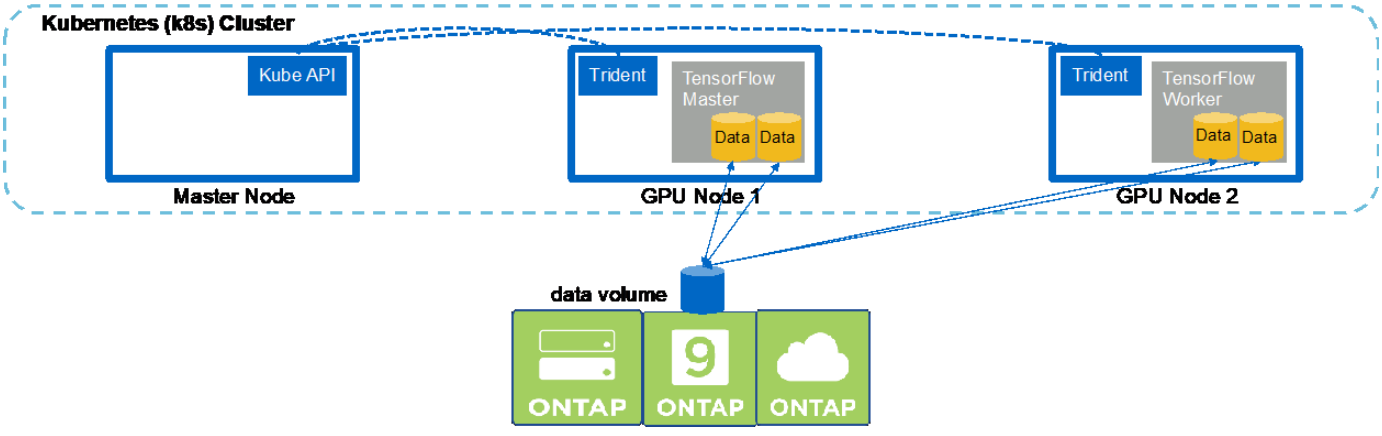
   Quando si elimina l'oggetto di lavoro, Kubernetes elimina automaticamente tutti i pod associati.

```
$ 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.
```

Eseguire un carico di lavoro ai distribuito sincrono

Per eseguire un processo ai e ML multinodo sincrono nel cluster Kubernetes, eseguire le seguenti operazioni sull'host di distribuzione jump. Questo processo consente di sfruttare i dati memorizzati su un volume NetApp e di utilizzare più GPU di quelle che un singolo nodo di lavoro può fornire. Vedere la figura seguente per un'illustrazione di un lavoro di ai distribuito sincrono.

I lavori distribuiti sincroni possono contribuire ad aumentare la precisione delle performance e della formazione rispetto ai lavori distribuiti asincroni. Una discussione sui pro e contro dei lavori sincroni rispetto ai lavori asincroni non rientra nell'ambito di questo documento.



1. I seguenti comandi di esempio mostrano la creazione di un worker che partecipa all'esecuzione distribuita sincrona dello stesso job di benchmark TensorFlow eseguito su un singolo nodo nell'esempio della sezione "Eseguire un carico di lavoro ai a nodo singolo". In questo esempio specifico, viene implementato solo un singolo worker perché il lavoro viene eseguito su due nodi di lavoro.

In questo esempio, l'implementazione di lavoro richiede otto GPU e può quindi essere eseguita su un singolo nodo di lavoro GPU che dispone di otto o più GPU. Se i nodi di lavoro GPU dispongono di più di otto GPU, per massimizzare le performance, è possibile aumentare questo numero in modo da essere uguale al numero di GPU presenti nei nodi di lavoro. Per ulteriori informazioni sulle implementazioni di Kubernetes, vedere "Documentazione ufficiale di Kubernetes".

In questo esempio viene creata un'implementazione di Kubernetes perché questo specifico lavoratore containerizzato non viene mai completato da solo. Pertanto, non ha senso implementarlo utilizzando il costrutto di lavoro Kubernetes. Se il tuo lavoratore è stato progettato o scritto per essere completato da solo, potrebbe essere opportuno utilizzare il costrutto di lavoro per implementare il tuo lavoratore.

Al pod specificato in questa specifica di implementazione di esempio viene assegnato un `hostNetwork` valore di `true`. Questo valore significa che il pod utilizza lo stack di rete del nodo di lavoro host invece dello stack di rete virtuale creato da Kubernetes per ciascun pod. Questa annotazione viene utilizzata in questo caso perché il carico di lavoro specifico si basa su Open MPI, NCCL e Horovod per eseguire il carico di lavoro in maniera sincrona e distribuita. Pertanto, richiede l'accesso allo stack di rete host. Una discussione su Open MPI, NCCL e Horovod non rientra nell'ambito di questo documento. Che sia o meno così `hostNetwork: true` l'annotazione è necessaria a seconda dei requisiti del carico di lavoro specifico che si sta eseguendo. Per ulteriori informazioni su `hostNetwork` vedere il campo "Documentazione ufficiale di 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. Verificare che l'implementazione worker creata al punto 1 sia stata avviata correttamente. I seguenti comandi di esempio confermano che è stato creato un singolo pod di lavoro per l'implementazione, come indicato nella definizione di implementazione, e che questo pod è attualmente in esecuzione su uno dei nodi di lavoro 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. Creare un lavoro Kubernetes per un master che inizia, partecipa e tiene traccia dell'esecuzione del lavoro sincrono a più nodi. I seguenti comandi di esempio creano un master che inizia, partecipa e tiene traccia dell'esecuzione distribuita sincrona dello stesso job di benchmark TensorFlow eseguito su un singolo nodo

nell'esempio nella sezione "Eseguire un carico di lavoro ai a nodo singolo".

Questo processo master di esempio richiede otto GPU e può quindi essere eseguito su un singolo nodo di lavoro GPU che dispone di otto o più GPU. Se i nodi di lavoro GPU dispongono di più di otto GPU, per massimizzare le performance, è possibile aumentare questo numero in modo da essere uguale al numero di GPU presenti nei nodi di lavoro.

Al pod master specificato in questa definizione di lavoro di esempio viene assegnato un `hostNetwork` valore di `true`, proprio come al pod di lavoro è stato assegnato un `hostNetwork` valore di `true` nella fase 1. Per ulteriori informazioni sul motivo per cui questo valore è necessario, vedere il passaggio 1.

```
$ 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. Verificare che il lavoro principale creato al punto 3 sia in esecuzione correttamente. Il seguente comando di esempio conferma che è stato creato un singolo pod master per il lavoro, come indicato nella definizione del lavoro, e che questo pod è attualmente in esecuzione su uno dei nodi di lavoro GPU. Inoltre, il pod di lavoro inizialmente visto al punto 1 è ancora in esecuzione e i pod master e di lavoro sono in esecuzione su nodi diversi.

```
$ 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. Verificare che il lavoro principale creato al punto 3 sia stato completato correttamente. I seguenti comandi di esempio confermano che il lavoro è stato completato correttamente.

```
$ 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. Eliminare l'implementazione dei lavoratori quando non è più necessaria. I seguenti comandi di esempio mostrano l'eliminazione dell'oggetto di implementazione worker creato nel passaggio 1.

   Quando si elimina l'oggetto di implementazione worker, Kubernetes elimina automaticamente tutti i worker pod associati.

```
$ 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. **Opzionale:** eliminare gli artefatti del job master. I seguenti comandi di esempio mostrano l'eliminazione dell'oggetto di lavoro master creato nel passaggio 3.

   Quando si elimina l'oggetto di lavoro master, Kubernetes elimina automaticamente tutti i pod master associati.

```
$ 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.
```