# Exemple de tâches hautes performances pour les déploiements AIPod

NetApp Solutions

NetApp
May 10, 2024

# Sommaire

# Exemple de tâches hautes performances pour les déploiements AIPod

## Exécutez un workload d'IA à un seul nœud

Pour exécuter une tâche d'IA et DE ML à un seul nœud dans votre cluster Kubernetes, effectuez les tâches suivantes à partir de l'hôte de démarrage du déploiement. Trident vous permet de faire rapidement et facilement un volume de données, potentiellement contenant des pétaoctets de données, accessibles pour un workload Kubernetes. Pour rendre ce volume de données accessible depuis un pod Kubernetes, il vous suffit de spécifier une demande de volume persistant dans la définition du pod.

Dans cette section, vous devez déjà avoir conteneurisé (au format du conteneur Docker) le workload d'IA et DE ML spécifique que vous essayez d'exécuter dans votre cluster Kubernetes.

1. L'exemple de commandes suivant montre la création d'un travail Kubernetes pour un workload de banc d'essai TensorFlow qui utilise le dataset ImageNet. Pour plus d'informations sur le dataset ImageNet, reportez-vous à la "Site Web ImageNET".

   Cet exemple de tâche nécessite huit GPU, soit un nœud worker doté d'au moins huit GPU. Cet exemple de tâche peut être envoyée dans un cluster pour lequel un nœud worker doté d'au moins huit GPU n'est pas présent ou est actuellement occupé par une autre charge de travail. Si c'est le cas, le travail reste à l'état en attente jusqu'à ce qu'un nœud de travail soit disponible.

   En outre, pour optimiser la bande passante de stockage, le volume contenant les données d'entraînement requises est monté deux fois dans le pod que cette tâche crée. Un autre volume est également monté dans le pod. Ce deuxième volume sera utilisé pour stocker les résultats et les mesures. Ces volumes sont référencés dans la définition de travail en utilisant les noms des ESV. Pour plus d'informations sur les tâches Kubernetes, consultez le "Documentation officielle Kubernetes".

   An `emptyDir` volume avec un `medium` valeur de `Memory` est monté sur `/dev/shm` dans le pod créé par cet exemple de travail. La taille par défaut du `/dev/shm` Le volume virtuel créé automatiquement par le runtime des conteneurs Docker peut parfois manquer pour TensorFlow. Montage d'un `emptyDir` comme dans l'exemple suivant, le volume est suffisamment grand `/dev/shm` volume virtuel. Pour plus d'informations sur `emptyDir` les volumes, voir "Documentation officielle Kubernetes".

   Le conteneur unique spécifié dans cet exemple de définition de travail est donné un `securityContext > privileged` valeur de `true`. Cette valeur signifie que le conteneur dispose d'un accès racine sur l'hôte. Cette annotation est utilisée dans ce cas, car la charge de travail spécifique exécutée nécessite un accès racine. Plus précisément, une opération de mise en cache claire exécutée par la charge de travail nécessite un accès racine. Si cela est ou non `privileged: true` l'annotation est nécessaire dépend des exigences de la charge de travail spécifique que vous exécutez.

```
$ 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. Vérifiez que le travail que vous avez créé à l'étape 1 fonctionne correctement. L'exemple de commande

suivant confirme qu'un seul pod a été créé pour le travail, comme spécifié dans la définition du travail, et que ce pod s'exécute actuellement sur l'un des nœuds workers 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. Vérifiez que le travail que vous avez créé à l'étape 1 s'est terminé avec succès. L'exemple de commandes suivant confirme que le travail a été terminé avec succès.

```
$ 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. **Facultatif:** nettoyer les artefacts de travail. Les exemples de commandes suivants montrent la suppression de l'objet de travail créé à l'étape 1.

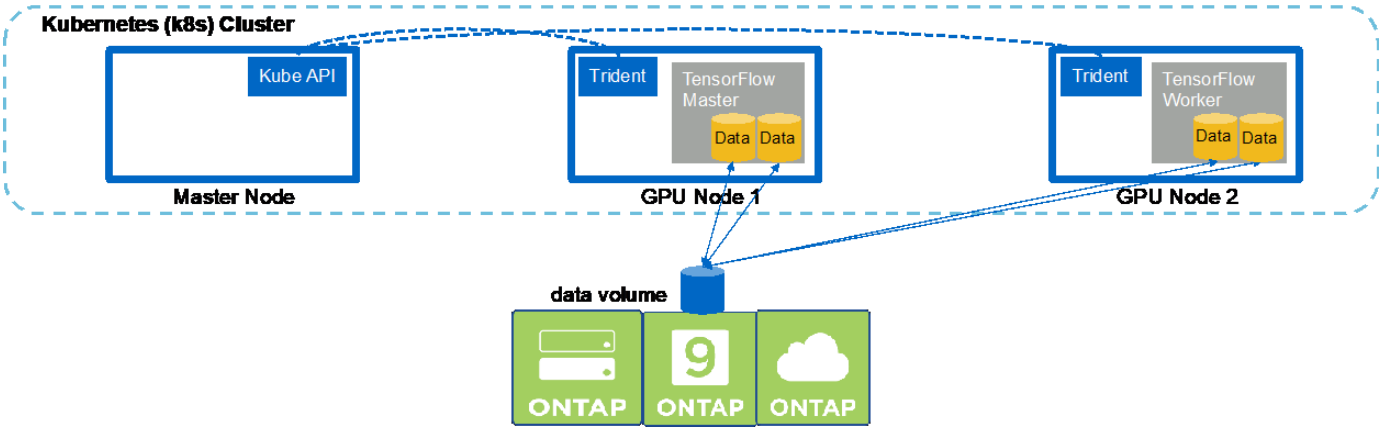
   Lorsque vous supprimez l'objet travail, Kubernetes supprime automatiquement les pods associés.

```
$ 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.
```

# Exécutez un workload d'IA distribué synchrone

Pour exécuter une tâche d'IA et DE ML à plusieurs nœuds synchrones dans votre cluster Kubernetes, exécutez les tâches suivantes sur l'hôte de démarrage du déploiement. Ce processus vous permet de exploiter les données stockées sur un volume NetApp et d'utiliser plus de GPU que n'en fournir un seul nœud de travail. Reportez-vous à la figure suivante pour obtenir une description d'une tâche d'IA distribuée synchrone.

> Les tâches distribuées synchrones permettent d'améliorer les performances et la précision de l'entraînement par rapport aux tâches distribuées asynchrones. Un examen des avantages et inconvénients des emplois synchrones par rapport aux emplois asynchrones est hors du champ d'application de ce document.




1. Les commandes d'exemple suivantes montrent la création d'un travailleur qui participe à l'exécution distribuée synchrone du même travail de banc d'essai TensorFlow qui a été exécuté sur un seul nœud dans l'exemple de la section "Exécutez un workload d'IA à un seul nœud". Dans cet exemple spécifique, seul un travailleur est déployé car le travail est exécuté sur deux nœuds worker.

Cet exemple de déploiement utilisateur nécessite huit GPU, et peut donc s'exécuter sur un seul nœud worker GPU doté d'au moins huit GPU. Si les nœuds workers GPU disposent de plus de huit GPU, afin d'optimiser les performances, il est possible que vous souhaitiez augmenter ce nombre afin qu'il soit égal au nombre de GPU dont bénéficient les nœuds workers. Pour en savoir plus sur les déploiements Kubernetes, rendez-vous sur le "Documentation officielle Kubernetes".

Un déploiement Kubernetes est créé dans cet exemple, car ce travailleur conteneurisé ne s'en serait jamais achevé seul. C'est pourquoi il n'est pas logique de le déployer via la construction de tâches Kubernetes. Si votre travailleur est conçu ou écrit de manière à le compléter seul, il peut être judicieux d'utiliser la structure de travail pour déployer votre travailleur.

Le pod spécifié dans cet exemple de spécification de déploiement est donné un `hostNetwork` valeur de `true`. Cette valeur signifie que le pod utilise la pile réseau du nœud du worker hôte au lieu de la pile de réseau virtuel que Kubernetes crée habituellement pour chaque pod. Cette annotation est utilisée dans ce cas car la charge de travail spécifique repose sur Open MPI, NCCL et Horovod pour exécuter la charge de travail de façon synchrone distribuée. Par conséquent, elle nécessite l'accès à la pile réseau de l'hôte. Une discussion sur Open MPI, NCCL et Horovod n'est pas dans le cadre du présent document. Si cela est ou non `hostNetwork: true` l'annotation est nécessaire dépend des exigences de la charge de travail spécifique que vous exécutez. Pour plus d'informations sur le `hostNetwork` voir "Documentation officielle 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. Confirmez que le déploiement de collaborateur que vous avez créé à l'étape 1 a été lancé avec succès. Les exemples de commandes suivants confirment qu'un seul pod worker a été créé pour le déploiement, comme indiqué dans la définition du déploiement, et que ce pod s'exécute actuellement sur l'un des nœuds workers 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. Créez un travail Kubernetes pour un master qui démarre, participe et suit l'exécution du travail multinœud synchrone. Les commandes d'exemple suivantes créent un master qui démarre, participe à et assure le suivi de l'exécution distribuée synchrone du même travail de banc d'essai TensorFlow qui a été exécuté

sur un seul nœud dans l'exemple de la section "Exécutez un workload d'IA à un seul nœud".

Cet exemple de tâche maître demande huit GPU, puis peut être exécuté sur un seul nœud worker GPU doté d'au moins huit GPU. Si les nœuds workers GPU disposent de plus de huit GPU, afin d'optimiser les performances, il est possible que vous souhaitiez augmenter ce nombre afin qu'il soit égal au nombre de GPU dont bénéficient les nœuds workers.

Le pod maître spécifié dans cet exemple de définition de travail est donné un `hostNetwork` valeur de `true`, tout comme le pod de travailleur a été donné `hostNetwork` valeur de `true` à l'étape 1. Voir l'étape 1 pour plus de détails sur la raison pour laquelle cette valeur est nécessaire.

```
$ 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. Vérifiez que le travail principal que vous avez créé à l'étape 3 fonctionne correctement. L'exemple de commande suivant confirme qu'un module maître unique a été créé pour le travail, comme indiqué dans la définition du travail, et que ce pod s'exécute actuellement sur l'un des nœuds workers GPU. Vous devriez également voir que le pod de worker que vous avez initialement vu à l'étape 1 est toujours en cours d'exécution et que les pods master et worker exécutent sur différents nœuds.

```
$ 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. Confirmez que le travail principal que vous avez créé à l'étape 3 s'est terminé avec succès. L'exemple de commandes suivant confirme que le travail a été terminé avec succès.

```
$ 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. Supprimez le déploiement de collaborateur lorsque vous n'en avez plus besoin. L'exemple de commandes suivant montre la suppression de l'objet de déploiement de travail qui a été créé à l'étape 1.

   Lorsque vous supprimez l'objet de déploiement worker, Kubernetes supprime automatiquement les pods workers associés.

```
$ 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. **Facultatif:** nettoyez les artefacts du travail principal. Les exemples de commandes suivants montrent la suppression de l'objet de travail maître créé à l'étape 3.

   Lorsque vous supprimez l'objet de travail maître, Kubernetes supprime automatiquement les modules maîtres associés.

```
$ 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.
```

## Informations sur le copyright



## Informations sur les marques commerciales