

TP Docker Swarm : Déploiement et gestion de services dans un cluster

Objectif :

Ce TP vous a permis de découvrir les principales commandes de gestion d'un cluster Docker Swarm, de la création du cluster à la gestion des services et des nœuds. Vous avez appris à orchestrer des services, à scaler dynamiquement les conteneurs, à gérer la disponibilité des nœuds et à mettre à jour des applications sans interruption.

Docker Swarm: Scenario d'utilisation

1. Création d'un cluster Swarm

- Création des nœuds
- Activation du mode swarm
- Jointure des machines workers au cluster

2. Déploiement d'une application

- Création des services
- Orchestration des conteneurs

Étape 1 : Création du cluster Swarm

1. **Initialiser le cluster Swarm sur le premier nœud (Node 1)** : Sur Node 1, exécutez la commande suivante pour initier le Swarm :

docker swarm init --advertise-addr <IP_NODE_1>

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm init --advertise-addr 192.168.0.13
Swarm initialized: current node (aiat4o7lp5wj6ebck6l6ne9kt) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-68jtle5x375bugjxc2z8ck4n3xqfftcrnos22g24w4f0fexke3-7h3s9geqzys8302oog
i8l4bcs 192.168.0.13:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

- Cette commande initialise le Swarm et désigne le nœud 1 comme **Manager**.
 - L'option `--advertise-addr` permet d'indiquer l'adresse IP du manager pour que les autres nœuds puissent le rejoindre.
2. **Obtenir le token pour ajouter des nœuds** : Après l'initialisation du Swarm, Docker génère un token que vous devez utiliser pour ajouter des nœuds au cluster.
 - Pour ajouter un **nœud Manager** :

docker swarm join-token manager

- Pour ajouter un **nœud Worker** :

docker swarm join-token worker

Étape 2 : Ajouter des nœuds au Swarm

1. **Ajouter le Nœud 2** : Sur le Nœud 2, exécutez la commande join avec le token obtenu dans l'étape précédente :

docker swarm join --token <manager_token>

```
[node2] (local) root@192.168.0.12 ~
$ docker swarm join --token SWMTKN-1-68jtlesx375bugjxc2z8ck4n3xqfftcnos22g24w4f0fexke3-7h3s9geqzys8302oogi8
14bcs 192.168.0.13:2377
This node joined a swarm as a worker.
[node2] (local) root@192.168.0.12 ~
```

2. **Ajouter le Nœud 3 comme *manager*** : Sur le Nœud 3, exécutez la commande join avec le token pour Worker :

Node 1 : docker swarm join-token manager

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-68jtlesx375bugjxc2z8ck4n3xqfftcnos22g24w4f0fexke3-8jljz5itxgyz1759ty
    bdked7y 192.168.0.13:2377
```

docker swarm join --token <worker_token>

```
[node3] (local) root@192.168.0.11 ~
$ docker swarm join --token SWMTKN-1-68jtlesx375bugjxc2z8ck4n3xqfftcnos22g24w4f0fexke3-8jljz5itxgyz1759tybd
ked7y 192.168.0.13:2377
This node joined a swarm as a manager.
```

Étape 3 : Vérification des nœuds dans le cluster

1. **Lister les nœuds** : Sur le Nœud 1, exécutez la commande suivante pour vérifier les nœuds dans le cluster :

docker node ls

```
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                                HOSTNAME        STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
aiat4o7lp5wj6ebck616ne9kt *     node1          Ready     Active           Leader             27.3.1
gdhja9lmo2adzggndmlcjsrgg       node2          Ready     Active           Reachable          27.3.1
vafvxxvjz2r0numet5k0nhwxj6       node3          Ready     Active           Reachable          27.3.1
```

Vous devriez voir les trois nœuds (Nœud 1, Nœud 2 et Nœud 3) avec leurs rôles respectifs (Manager ou Worker).

Étape 4 : Créer un service dans Docker Swarm

1. **Créer un service Docker Swarm** : Sur le Nœud 1, créez un service nommé testArctic avec les spécifications suivantes :
 - 3 réplicas (instances).
 - Publier le port 5001 de l'hôte sur le port 5000 du conteneur.

- Redémarrage automatique en cas de panne.
- Limiter la mémoire à 100 Mo pour chaque conteneur.

La commande est la suivante :

```
docker service create --name testArctic \  
  
--replicas 3 \  
  
--publish published=5001,target=5000 \  
  
--restart-condition=on-failure \  
  
--limit-memory 100M \  
  
hajdaini/flask:first
```

```
valvkvjz21f0ndmets3konmwxj6      nodes      ready      Active      Reachable      27.3.1  
[node1] (local) root@192.168.0.13 ~  
$ docker service create --name testArctic \  
--replicas 3 \  
--publish published=5001,target=5000 \  
--restart-condition=on-failure \  
--limit-memory 100M \  
hajdaini/flask:first  
5clzrd35359k2cxkp3odqzgkn  
overall progress: 0 out of 3 tasks  
1/3: preparing  
2/3: preparing  
3/3: preparing
```

```
5clzrd35359k2cxkp3odqzgkn  
overall progress: 3 out of 3 tasks  
1/3: running  
2/3: running  
3/3: running  
verify: Service 5clzrd35359k2cxkp3odqzgkn converged
```

Étape 5 : Vérifier l'état du service

1. **Vérifier l'état du service** : Utilisez la commande suivante pour vérifier que le service a bien été créé et que les réplicas sont en cours d'exécution :

docker service ls

```
[node1] (local) root@192.168.0.13 ~  
$ docker service ls  
ID                NAME                MODE                REPLICAS            IMAGE                PORTS  
5clzrd35359k     testArctic          replicated          3/3                 hajdaini/flask:first *:5001->5000/tcp
```

2. **Lister les tâches du service** : Pour vérifier le statut des réplicas, exécutez :

docker service ps testArctic

```
[node1] (local) root@192.168.0.13 ~
$ docker service ps testArctic
```

ID	PORTS	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
cdlfzxp36v		testArctic.1	hajdaini/flask:first	node1	Running	Running 2 minutes ago	
ehc8ihl5lhp		testArctic.2	hajdaini/flask:first	node2	Running	Running 2 minutes ago	
ljd3r3mujech		testArctic.3	hajdaini/flask:first	node3	Running	Running 2 minutes ago	

3. **Scaler le service** : Vous pouvez augmenter le nombre de répliques du service sans redémarrer les conteneurs en utilisant :

docker service scale testArctic=5

```
[node1] (local) root@192.168.0.13 ~
$ docker service scale testArctic=5
testArctic scaled to 5
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service testArctic converged
```

Après cela, vérifiez à nouveau l'état du service avec **docker service ls**.

```
[node1] (local) root@192.168.0.13 ~
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
5clzrd35359k	testArctic	replicated	5/5	hajdaini/flask:first	*:5001->5000/tcp

Étape 6 : Re-planifier les tâches (Drain un nœud)

1. **Mettre un nœud en mode "Drain"** : Si vous souhaitez évacuer toutes les tâches du Nœud 3 sans l'arrêter, vous pouvez mettre ce nœud en mode **Drain** :

docker node update --availability drain <node_id>

Cela déplace toutes les tâches du Nœud 3 vers d'autres nœuds actifs.

```
gdhja9lmo2adzggndmlcjsrgg node2 Ready Active 27.3.1
vafvxvjz2r0numet5k0nhwxj6 node3 Ready Active Reachable 27.3.1
[node1] (local) root@192.168.0.13 ~
$ docker node update --availability drain vafvxvjz2r0numet5k0nhwxj6
vafvxvjz2r0numet5k0nhwxj6
[node1] (local) root@192.168.0.13 ~
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
aiat4o7lp5wj6ebck616ne9kt *	node1	Ready	Active	Leader	27.3.1
gdhja9lmo2adzggndmlcjsrgg	node2	Ready	Active		27.3.1
vafvxvjz2r0numet5k0nhwxj6	node3	Ready	Drain	Reachable	27.3.1

2. **Vérifier l'état des nœuds après le mode "Drain"** :

docker node ls

docker service ps testArctic

euyljw6429dl	testArctic.3	hajdaini/flask:first	node1	Running	Running about a minute ag
ljd3r3mujech	_ testArctic.3	hajdaini/flask:first	node3	Shutdown	Shutdown about a minute a
go					
xda7laopphj6	testArctic.4	hajdaini/flask:first	node1	Running	Running 3 minutes ago
vo52cjmwhk26	testArctic.5	hajdaini/flask:first	node2	Running	Running 3 minutes ago

3. **Réactiver un nœud** : Pour rendre un nœud disponible à nouveau, exécutez la commande suivante :

docker node update --availability active <node_id>

```
[node1] (local) root@192.168.0.13 ~
$ docker node update --availability active vafvxvjz2r0numet5k0nhwxj6
vafvxvjz2r0numet5k0nhwxj6
[node1] (local) root@192.168.0.13 ~
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
aiat4o7lp5wj6ebck616ne9kt *	node1	Ready	Active	Leader	27.3.1
gdhja9lmo2adzggndmlcjsrgg	node2	Ready	Active		27.3.1
vafvxvjz2r0numet5k0nhwxj6	node3	Ready	Active	Reachable	27.3.1

Remarque :

Lorsque on redéfinit la disponibilité du nœud sur Active, il peut recevoir de nouvelles tâches:

- Lors d'une mise à jour de service pour un scale up
- Lors d'une mise à jour continue Lorsqu'on définit un autre nœud sur la disponibilité Drain
- Lorsqu'une tâche échoue sur un autre nœud actif

Étape 7 : Mettre à jour l'application sans interruption de service

1. **Mettre à jour l'application avec une nouvelle image** : Pour déployer une nouvelle version de votre service avec une image différente, exécutez la commande suivante sur le Nœud 1 :

docker service update --image hajdaini/flask:second testArctic

```
[node1] (local) root@192.168.0.13 ~
$ docker service update --image hajdaini/flask:second testArctic
testArctic
overall progress: 0 out of 5 tasks
1/5: ready
2/5:
3/5:
4/5:
5/5:
[]
```

2. **Vérifier le statut des tâches** : Après la mise à jour, vous pouvez vérifier les tâches en cours d'exécution avec :

docker service ps testArctic

docker service ps testArctic --filter "node=<node_id>"

```
[node1] (local) root@192.168.0.13 ~
$ docker service ps testArctic --filter "node=node3"
ID                NAME                IMAGE                NODE                DESIRED STATE        CURRENT STATE        ER
ROR              PORTS
p4x6qzj5e5g0     testArctic.2        hajdaini/flask:second node3                Running              Running 7 minutes ago
ljd3r3mujech     testArctic.3        hajdaini/flask:first  node3                Shutdown             Shutdown 13 minutes ago
```

Vous devriez voir que les anciens conteneurs sont arrêtés et que les nouveaux conteneurs avec l'image flask:second sont en cours d'exécution.

Ici c'est un exemple de node2

```
$ docker service ps testArctic --filter "node=node2"
ID                NAME                IMAGE                NODE                DESIRED STATE        CURRENT STATE        ER
ROR              PORTS
ehc8ihl5lhp      testArctic.2        hajdaini/flask:first  node2                Shutdown             Shutdown 9 minutes ago
5sgbxokx3srv     testArctic.3        hajdaini/flask:second node2                Running              Running 9 minutes ago
uslkr9fvm850     testArctic.5        hajdaini/flask:second node2                Running              Running 9 minutes ago
vo52cjmwhk26     \_ testArctic.5     hajdaini/flask:first  node2                Shutdown             Shutdown 9 minutes ago
```

Promouvoir un worker en manager dans ce cas, veut dire transformer un Worker en Manager

docker node promote <nom_du_worker/ID>

```
[node1] (local) root@192.168.0.13 ~
$ docker node promote node2
Node node2 promoted to a manager in the swarm.
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                HOSTNAME            STATUS             AVAILABILITY        MANAGER STATUS        ENGINE VERSION
aiat4o7lp5wj6ebck616ne9kt * node1                Ready              Active               Leader                 27.3.1
gdhja9lmo2adzggndmlcjsrgg node2                Ready              Active               Reachable              27.3.1
vafvxvjz2r0numet5k0nhwxj6 node3                Ready              Active               Reachable              27.3.1
```

Rétrograder un Manager en Worker:

Contrairement au promotion d'un worker en manager, on peut aussi **rétrograder un manager en worker** en exécutant la commande suivante:

docker node demote <nom_du_manager>

```
gdhja9lmo2adzggndmlcjsrgg    node2    Ready    Active    Reachable    27.3.1
vafvxvjz2r0numet5k0nhwxj6    node3    Ready    Active    Reachable    27.3.1
[node1] (local) root@192.168.0.13 ~
$ docker node demote node3
Manager node3 demoted in the swarm.
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                                HOSTNAME    STATUS    AVAILABILITY    MANAGER STATUS    ENGINE VERSION
aiat4o7lp5wj6ebck6l6ne9kt *    node1    Ready    Active          Leader            27.3.1
gdhja9lmo2adzggndmlcjsrgg    node2    Ready    Active          Reachable         27.3.1
vafvxvjz2r0numet5k0nhwxj6    node3    Ready    Active          Reachable         27.3.1
[node1] (local) root@192.168.0.13 ~
```

Inspecter un nœud spécifique:

Pour Inspecter un nœud spécifique on utilise cette commande:

docker node inspect <nom_du_noeud/ID>

```
[node1] (local) root@192.168.0.13 ~
$ docker node inspect node2
[
  {
    "ID": "gdhja9lmo2adzggndmlcjsrgg",
    "Version": {
      "Index": 101
    },
    "CreatedAt": "2025-02-11T21:23:25.343033126Z",
    "UpdatedAt": "2025-02-11T21:53:48.0686295Z",
    "Spec": {
```

```
    },
    "Status": {
      "State": "ready",
      "Addr": "192.168.0.12"
    },
    "ManagerStatus": {
      "Reachability": "reachable",
      "Addr": "192.168.0.12:2377"
    }
  }
]
```

Ajouter un label au nœud:

docker node update --label-add region=africa <ID/nom>

```
[node1] (local) root@192.168.0.13 ~  
$ docker node update --label-add region=africa node1  
node1
```

Inspector node1 :

```
updatedat: 2023-02-11T22:04:20.101211932Z  
"Spec": {  
  "Labels": {  
    "region": "africa"  
  },  
  "Role": "manager",  
  "Availability": "active"  
}
```