Version - **2024.01**

Dernière mise-à-jour : 2024/12/15 06:51

DOF304 - Travailler avec des Pods et des Conteneurs

Contenu du Module

Ressources

Lab #1

- https://www.dropbox.com/scl/fi/7hkyea9v3lc949b9ar5hl/myconfigmap.yaml?rlkey=kv5x17lirugxppbyzgk0yhbhh&dl=0
- https://www.dropbox.com/scl/fi/o752fqb1gc5shocih9zc7/mysecret.yaml?rlkey=mfof15llfnanksi0ztmzdy7tp&dl=0
- https://www.dropbox.com/scl/fi/70g1jb7p4ighdbkk33mre/envpod.yaml?rlkey=31muxz3g7a7k91nd98bjxhkjz&dl=0
- https://www.dropbox.com/scl/fi/td43bvv8aphqqbwc59j4l/volumepod.yaml?rlkey=sti941svagvli2qbi6jkljaoy&dl=0

Lab #2

- https://www.dropbox.com/scl/fi/n1147jb572h0dnadwjamm/bigrequestpod.yaml?rlkey=08fpyndzpg720or0h6zkm9vxz&dl=0
- https://www.dropbox.com/scl/fi/3lo335z508wo4sutr8zwk/resourcepod.yaml?rlkey=ezycaxxvyf74u7xdtawhnssje&dl=0

Lab #3

- https://www.dropbox.com/scl/fi/9igcin5jo18z1bpjjx9vx/livenesspod.yaml?rlkey=23f17olf3jo8l12h972noijve&dl=0
- https://www.dropbox.com/scl/fi/tqno3tjsif093kpxo0jrg/livenesspodhttp.yaml?rlkey=lsn5q2d9goe619jnkpz3p6ok2&dl=0
- https://www.dropbox.com/scl/fi/s4pst2ezp0qpylu6m8frx/startuppod.yaml?rlkey=xbaenkztscopqzuq8u4dxxcx8&dl=0
- https://www.dropbox.com/scl/fi/a0hdk8shspxsi23hkf7vi/readinesspod.yaml?rlkey=w230asyme4ywxitfzgzy4ehsw&dl=0

Lab #4

- https://www.dropbox.com/scl/fi/y8bu7cryzv5wfkln2r6wc/alwayspod.yaml?rlkey=n5rmuhmy4o1gojvez1yz3w1ys&dl=0
- https://www.dropbox.com/scl/fi/m6wy0x16vdsd87vuriyl9/onfailure.yaml?rlkey=ox8nfznllrjui1mal4idtzx3u&dl=0
- https://www.dropbox.com/scl/fi/7oyo26ackzdxjm78ipjvg/never.yaml?rlkey=hqf5f07kvmiuhdehyjc9r6mni&dl=0

Lab #5

- https://www.dropbox.com/scl/fi/4j0nnzgt8ammsfzpqm3ul/multicontainerpod.yaml?rlkey=n08saexw65stxvy4twd9x2npr&dl=0
- https://www.dropbox.com/scl/fi/x8fy28yiiyq7rrb5x7gse/helper.yaml?rlkey=9hhvly431j39x2vmfeopk9tk1&dl=0

Lab #6

- https://www.dropbox.com/scl/fi/llvkk1jija3pk227u6w8v/initpod.yaml?rlkey=krtkq8qhc8dalr84jw0p4jwdh&dl=0

Lab #7

- https://www.dropbox.com/scl/fi/qdnl21iip9shwjqc93rpy/nodeselector.yaml?rlkey=x5eumxvmgkeh9vctrwd9rmuwi&dl=0
- https://www.dropbox.com/scl/fi/46npmxik2heh8z3wiw6ah/nodename.yaml?rlkey=blck3kzwgqzm21ttsjxph965k&dl=0

Lab #8

- https://www.dropbox.com/scl/fi/fqxcxm7ia69ne9keruqg2/daemonset.yaml?rlkey=r7hn65en4beq3zvza5jxfysd5&dl=0

Lab #9

- https://www.dropbox.com/scl/fi/pwvwbsant7onw0hwikmp4/mystaticpod.yaml?rlkey=l6jzgtgcss3atx9emqk3h7qz6&dl=0

LAB #1 - Application Configuration

1.1 - Présentation

La gestion de la configuration d'application ou *Application Configuration* est le processus de passage de valeurs dynamiques aux applications au moment du runtime.

Il y a deux façons de stocker des informations dans K8s :

- ConfigMaps,
- Secrets.

Les données stockées dans des ConfigMaps et des Secrets peuvent être passées aux conteneurs en utilisant des :

- Variables d'environnement,
- Volumes de configuration.

1.2 - Création d'une ConfigMap

Pour commencer, créez le fichier **myconfigmap.yaml** :

```
root@kubemaster:~# vi myconfigmap.yaml
root@kubemaster:~# cat myconfigmap.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: my-configmap
data:
  key1: Hello, world!
  key2: |
    Test
    multiple lines
    more lines
```

Important : Notez que les données sont stockées dans des **Key-values**. La première donnée dans la section **data** est **key1: Hello, world!** tandis que la deuxième, **key2**, est en plusieurs lignes.

Créez maintenant la ConfigMap :

```
root@kubemaster:~# kubectl create -f myconfigmap.yaml
configmap/my-configmap created
```

Pour consulter le contenu de la ConfigMap, utilisez la commande **kubectl describe** :

```
root@kubemaster:~# kubectl describe configmap my-configmap
Name:         my-configmap
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
key1:
----
```

```
Hello, world!
key2:
----
Test
multiple lines
more lines


BinaryData
====

Events:  <none>
```

1.3 - Création d'un Secret

Créez maintenant le fichier **mysecret.yaml** :

```
root@kubemaster:~# vi mysecret.yaml
root@kubemaster:~# cat mysecret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  secretkey1:
  secretkey2:
```

Important : Notez que les clefs secrets n'ont pas encore été définies.

Cryptez maintenant les deux clefs en utilisant **base64** :

```
root@kubemaster:~# echo -n 'secret' | base64
c2VjcmV0

root@kubemaster:~# echo -n 'anothersecret' | base64
YW5vdGhlcnNlY3JldA==
```

Copiez et collez les chaînes base64 dans le fichier mysecret.yaml :

```
root@kubemaster:~# vi mysecret.yaml
root@kubemaster:~# cat mysecret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  secretkey1: c2VjcmV0
  secretkey2: YW5vdGhlcnNlY3JldA==
```

Important : Remplacez les chaînes par celles que VOUS avez créé.

Créez maintenant le Secret :

```
root@kubemaster:~# kubectl create -f mysecret.yaml
secret/my-secret created
```

1.4 - Utilisation de ConfigMaps et de Secret

Utilisation des Variables d'environnement

Créez le fichier **envpod.yaml** :

```
root@kubemaster:~# vi envpod.yaml
root@kubemaster:~# cat envpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: envpod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'echo "configmap: $CONFIGMAPVAR secret: $SECRETVAR"']
    env:
    - name: CONFIGMAPVAR
      valueFrom:
        configMapKeyRef:
          name: my-configmap
          key: key1
    - name: SECRETVAR
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: secretkey1
```

Important : Notez que la variable **$CONFIGMAPVAR** contiendra la valeur de **key1** de la **ConfigMap** et que la variable **$SECRETVAR** contindra la valeur de **secretkey1** du

Secret.

Créez maintenant le pod :

```
root@kubemaster:~# kubectl create -f envpod.yaml
pod/envpod created
```

Consultez maintenant les logs du pod :

```
root@kubemaster:~# kubectl logs envpod
configmap: Hello, world! secret: secret
```

Important : Notez que le conteneur dans le pod voit bien les valeurs des deux variables.

Utilisation des Volumes de Configuration

Créez le fichier **volumepod.yaml** :

```
root@kubemaster:~# vi volumepod.yaml
root@kubemaster:~# cat volumepod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: volumepod
spec:
  containers:
  - name: busybox
    image: busybox
```

```
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    volumeMounts:
    - name: configmap-volume
      mountPath: /etc/config/configmap
    - name: secret-volume
      mountPath: /etc/config/secret
  volumes:
  - name: configmap-volume
    configMap:
      name: my-configmap
  - name: secret-volume
    secret:
      secretName: my-secret
```

Créez maintenant le pod :

```
root@kubemaster:~# kubectl create -f volumepod.yaml
pod/volumepod created
```

Utilisez maintenant la commande **kubectl exec** pour consulter les config data files dans le conteneur :

```
root@kubemaster:~# kubectl exec volumepod -- ls /etc/config/configmap
key1
key2

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/configmap/key1
Hello, world!root@kubemaster:~# [Enter]

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/configmap/key2
Test
multiple lines
more lines

root@kubemaster:~# kubectl exec volumepod -- ls /etc/config/secret
```

```
secretkey1
secretkey2

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/secret/secretkey1
secretroot@kubemaster:~# [Enter]

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/secret/secretkey2
anothersecretroot@kubemaster:~# [Enter]

root@kubemaster:~#
```

Dernièrement, supprimez les pods **envpod** et **volumepod** :

```
root@kubemaster:~# kubectl delete pod envpod volumepod
pod "envpod" deleted
pod "volumepod" deleted
```

LAB #2 - Gestion des Ressources des Conteneurs

2.1 - Présentation

Deux aspects importants de la gestion des ressources des conteneurs sont :

- **Resource Requests**,
  - Une Resource Request permet de définir des ressources telles le CPU et la mémoire au moment du **scheduling**. Autrement dit, si la Resource Request est de 5Go, le scheduleur des pods cherchera une noeud ayant 5 Go de RAM disponible. Une Resource Request n'est pas une limite car le pod peut utiliser plus ou moins de mémoire.
- **Resource Limits**,
  - Une Resource Limit permet de définir des limites des ressources telles le CPU et la mémoire. Différents Container Runtimes réagissent de manières différentes devant une Resource Limit. Par exemple, certains vont arrêter le processus du conteneur en cas de dépassement de la limite. Dans le cas de Docker, si la limite du CPU est dépassé, Docker va limiter l'utilisation du CPU. Par contre dans le cas d'un dépassement de la limite de la mémoire, Docker va tuer le processus du conteneur.

Pour les deux types, les demandes et les limites de la mémoire sont généralement exprimées en **Mi**, tandis que les les demandes et les limites du CPU sont exprimées en 1/1000 d'un processeur. Par exemple le chiffre 250m représente 250/1000 d'un CPU ou 1/4.

2.2 - Resource Requests

Créez le fichier **bigrequestpod.yaml** :

```
root@kubemaster:~# vi bigrequestpod.yaml
root@kubemaster:~# cat bigrequestpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: bigrequestpod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    resources:
      requests:
        cpu: "10000m"
        memory: "128Mi"
```

Créez le pod :

```
root@kubemaster:~# kubectl create -f bigrequestpod.yaml
pod/bigrequestpod created
```

Consultez maintenant le statut du pod créé :

```
root@kubemaster:~# kubectl get pod bigrequestpod
NAME            READY   STATUS    RESTARTS   AGE
```

```
bigrequestpod   0/1     Pending   0          92s
```

Important : Notez que le statut du pod est en **pending**. Le pod restera en pending car ni **kubenode1**, ni **kubenode2** sont capables de satisfaire la demande de **10000m**.

2.3 - Resource Limits

Créez le fichier **resourcepod.yaml** :

```
root@kubemaster:~# vi resourcepod.yaml
root@kubemaster:~# cat resourcepod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: resourcepod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    resources:
      requests:
        cpu: "250m"
        memory: "128Mi"
      limits:
        cpu: "500m"
        memory: "256Mi"
```

Créez le pod :

```
root@kubemaster:~# kubectl create -f resourcepod.yaml
pod/resourcepod created
```

Consultez le statut des pods :

```
root@kubemaster:~# kubectl get pods
NAME                              READY   STATUS              RESTARTS       AGE
bigrequestpod                     0/1     Pending             0              20m
my-deployment-67b5d4bf57-6wcrq    1/1     Running             0              22h
myapp-deployment-689f9d59-c25f9   1/1     Running             0              7d
myapp-deployment-689f9d59-nn9sw   1/1     Running             0              7d
myapp-deployment-689f9d59-rnc4r   1/1     Running             0              7d
resourcepod                       1/1     Running             0              5m49s
```

Important : Notez que le statut du pod **bigrequestpod** est **toujours** en **pending**.

LAB #3 - Supervision des Conteneurs

3.1 - Présentation

La supervision des conteneurs concerne la surveillance de la santé des conteneurs afin d'assurer des applications et des solutions robustes en redémarrant des conteneurs cassés. Pour accomplir cette tâche, K8s utilise des sondes ou *probes* en anglais.

Il existe plusieurs types de sondes :

- **Liveness Probes**,
  - Par défaut K8s considère un conteneur HS uniquement quand le conteneur en question s'arrête,
  - Liveness probes permettent une configuration plus sophistiquée de ce mécanisme.
- **Startup Probes**,

- Similaires aux Liveness Probes, les Startup Probes n'interviennent uniquement au démarrage du conteneur et s'arrêtent quand l'application a démarré.

- **Readiness Probes**,
  - Similaires aux Startup Probes car ils n'interviennet qu'au démarrage du pod, les Readiness Probes sont responsables du blocage du trafic vers les pods tant que tous les conteneurs du pod n'ont pas réussi les Readiness Probes.

3.2 - Liveness Probes

Le Probe exec

Créez le fichier **livenesspod.yaml** :

```
root@kubemaster:~# vi livenesspod.yaml
root@kubemaster:~# cat livenesspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: livenesspod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    livenessProbe:
      exec:
        command: ["echo", "Hello, world!"]
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important : Dans le fichier ci-dessus, si la commande **echo “Hello, World!** retourne un code de retour de 0, le conteneur sera considéré en bonne santé. Le Liveness Probe

démarrera 5 secondes après le démarrage du conteneur grâce à la directive **initialDelaySeconds**. Ensuite le probe s'exécutera tous les 5 secondes grâce à la directive **periodSeconds**.

Créez le pod :

```
root@kubemaster:~# kubectl create -f livenesspod.yaml
pod/livenesspod created
```

Consultez le statut du pod :

```
root@kubemaster:~# kubectl get pod livenesspod
NAME           READY   STATUS    RESTARTS   AGE
livenesspod    1/1     Running   0          90s
```

Important : Notez que le pod est en bonne santé et à un statut de **running**.

Le Probe httpGet

Créez le fichier **livenesspodhttp.yaml** :

```
root@kubemaster:~# vi livenesspodhttp.yaml
root@kubemaster:~# cat livenesspodhttp.yaml
apiVersion: v1
kind: Pod
metadata:
  name: livenesspodhttp
spec:
```

```
  containers:
  - name: nginx
    image: nginx:1.19.1
    livenessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important : Dans le fichier ci-dessus, si la commande **GET /** s'exécute sans erreur, le conteneur sera considéré en bonne santé. Le Liveness Probe démarrera 5 secondes après le démarrage du conteneur grâce à la directive **initialDelaySeconds**. Ensuite le probe s'exécutera tous les 5 secondes grâce à la directive **periodSeconds**.

Créez le pod :

```
root@kubemaster:~# kubectl create -f livenesspodhttp.yaml
pod/livenesspodhttp created
```

Consultez le statut du pod :

```
root@kubemaster:~# kubectl get pod livenesspodhttp
NAME              READY   STATUS    RESTARTS   AGE
livenesspodhttp   1/1     Running   0          52s
```

Important : Notez que le pod est en bonne santé et à un statut de **running**.

3.3 - Startup Probes

Créez le fichier **startuppod.yaml** :

```
root@kubemaster:~# vi startuppod.yaml
root@kubemaster:~# cat startuppod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: startuppod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
    startupProbe:
      httpGet:
        path: /
        port: 80
      failureThreshold: 30
      periodSeconds: 10
```

Important : Dans le fichier ci-dessus, le Startup Probe va attendre un maximum de 30 secondes pour que l'application démarre grâce à la directive **failureThreshold**. Le probe s'exécutera tous les 10 secondes grâce à la directive **periodSeconds**.

Créez le pod :

```
root@kubemaster:~# kubectl create -f startuppod.yaml
pod/startuppod created
```

Consultez le statut du pod :

```
root@kubemaster:~# kubectl get pod startuppod
NAME           READY   STATUS    RESTARTS   AGE
livenesspod    1/1     Running   0          90s
```

Important : Notez que le pod est en bonne santé et à un statut de **running**.

3.4 - Readiness Probes

Créez le fichier **readinesspod.yaml** :

```
root@kubemaster:~# vi readinesspod.yaml
root@kubemaster:~# cat readinesspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: readinesspod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important : Dans le fichier ci-dessus, si la commande **GET /** s'exécute sans erreur, le conteneur sera considéré dasn un état de READY. Le Readiness Probe démarrera 5 secondes après le démarrage du conteneur grâce à la directive **initialDelaySeconds**. Ensuite le probe s'exécutera tous les 5 secondes grâce à la directive **periodSeconds**.

Créez le pod et consultez son statut :

```
root@kubemaster:~# kubectl create -f readinesspod.yaml;kubectl get pod readinesspod;sleep 1;kubectl get pod
readinesspod;sleep 3;kubectl get pod readinesspod;sleep 3;kubectl get pod readinesspod;sleep 3;kubectl get pod
readinesspod;sleep 3;kubectl get pod readinesspod
pod/readinesspod created
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Pending   0          0s
NAME           READY   STATUS              RESTARTS   AGE
readinesspod   0/1     ContainerCreating   0          1s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          4s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          7s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          10s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   1/1     Running   0          13s
```

Important : Notez que le pod est a un statut de Running 4 secondes après son démarrage. Par contre, le pod ne passe qu'en READY au bout de 13 secondes quand le Readiness Probe a réussi.

LAB #4 - Gestion des Politiques de Redémarrage

4.1 - Présentation

K8s peut redémarrer des conteneurs en cas de problèmes. Il y a trois politiques de redémarrage :

- **Always**,
  - Always est la politique par défaut,
  - Always redémarre **toujours** un conteneur quelque soit le code retour quand le conteneur est arrêté.
- **OnFailure**,
  - OnFailure ne redémarre un conteneur que dans ls cas où celui-ci sort avec un code retour autre que 0 ou dans le cas où un Liveness Probe a rapporté la mauvaise santé du conteneur. Dans le cas contraire, où le conteneur a terminé sa tâche et sort avec un code retour de 0, la politique ne le redémarre pas.
- **Never**,
  - Never est l'opposé de Always. Le conteneur n'est jamais redémarré en cas d'arrêt, quelque soit la cause de cette dernière.

4.2 - Always

Créez le fichier **alwayspod.yaml** :

```
root@kubemaster:~# vi alwayspod.yaml
root@kubemaster:~# cat alwayspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alwayspod
spec:
  restartPolicy: Always
  containers:
  - name: busybox
    image: busybox
```

```
    command: ['sh', '-c', 'sleep 10']
```

Créez le pod et consultez son statut :

```
root@kubemaster:~# kubectl create -f alwayspod.yaml;kubectl get pod alwayspod;sleep 9;kubectl get pod
alwayspod;sleep 9;kubectl get pod alwayspod;sleep 9;kubectl get pod alwayspod
pod/alwayspod created
NAME        READY   STATUS              RESTARTS   AGE
alwayspod   0/1     ContainerCreating   0          0s
NAME        READY   STATUS    RESTARTS   AGE
alwayspod   1/1     Running   0          9s
NAME        READY   STATUS    RESTARTS     AGE
alwayspod   1/1     Running   1 (6s ago)   19s
NAME        READY   STATUS      RESTARTS      AGE
alwayspod   0/1     Completed   1 (15s ago)   28s
```

Important : Notez que le pod a été redémarré.

4.3 - OnFailure

Créez le fichier **onfailure.yaml** :

```
root@kubemaster:~# vi onfailure.yaml
root@kubemaster:~# cat onfailure.yaml
apiVersion: v1
kind: Pod
metadata:
  name: onfailure
spec:
  restartPolicy: OnFailure
```

```
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10']
```

Créez le pod et consultez son statut :

```
root@kubemaster:~# kubectl create -f onfailure.yaml;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure
pod/onfailure created
NAME        READY   STATUS    RESTARTS   AGE
onfailure   0/1     Pending   0          0s
NAME        READY   STATUS    RESTARTS   AGE
onfailure   1/1     Running   0          9s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          19s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          28s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          37s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          46s
```

Important : Notez que le pod n'a pas été redémarré.

Supprimez maintenant le pod onfailure :

```
root@kubemaster:~# kubectl delete pod onfailure
pod "onfailure" deleted
```

Modifiez ensuite le fichier **onfailure.yaml** en ajoutant la chaîne **this is a bad command** :

```
root@kubemaster:~# vi onfailure.yaml
root@kubemaster:~# cat onfailure.yaml
apiVersion: v1
kind: Pod
metadata:
  name: onfailure
spec:
  restartPolicy: OnFailure
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10;this is a bad command']
```

Créez le pod et consultez son statut :

```
root@kubemaster:~# kubectl create -f onfailure.yaml;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure
pod/onfailure created
NAME        READY   STATUS    RESTARTS   AGE
onfailure   0/1     Pending   0          0s
NAME        READY   STATUS    RESTARTS   AGE
onfailure   1/1     Running   0          9s
NAME        READY   STATUS    RESTARTS     AGE
onfailure   1/1     Running   1 (5s ago)   18s
NAME        READY   STATUS   RESTARTS      AGE
onfailure   0/1     Error    1 (14s ago)   27s
NAME        READY   STATUS   RESTARTS      AGE
onfailure   0/1     Error    1 (23s ago)   36s
NAME        READY   STATUS    RESTARTS      AGE
onfailure   1/1     Running   2 (21s ago)   46s
```

Important : Notez que le pod a été redémarré à cause de l'erreur.

4.4 - Never

Créez le fichier **never.yaml** :

```
root@kubemaster:~# vi never.yaml
root@kubemaster:~# cat never.yaml
apiVersion: v1
kind: Pod
metadata:
  name: never
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10;this is a bad command']
```

Créez le pod et consultez son statut :

```
root@kubemaster:~# kubectl create -f never.yaml;kubectl get pod never;sleep 9;kubectl get pod never;sleep
9;kubectl get pod never;sleep 9;kubectl get pod never;sleep 9;kubectl get pod never;sleep 9;kubectl get pod never
pod/never created
NAME    READY   STATUS              RESTARTS   AGE
never   0/1     ContainerCreating   0          0s
NAME    READY   STATUS    RESTARTS   AGE
never   1/1     Running   0          9s
NAME    READY   STATUS   RESTARTS   AGE
```

```
never   0/1     Error    0          18s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          27s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          36s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          45s
```

Important : Notez que le pod n'a pas été redémarré.

LAB #5 - Création de Pods Multi-conteneurs

5.1 - Présentation

Il est toujours préférable de ne mettre qu'un seul conteneur dans un pod. L'exception à cette règle est quand deux ou plus de pods ont besoin d'interagir afin de remplir leur rôles respectifs. Le autres conteneur s'appellent des **sidecars** (*side-cars* en français) ou des **helpers** (*assistants* en français). L'interaction s'appelle **Cross-Container Interaction**.

Cette interaction prend la forme de partager :

- le même espace réseau,
  - les conteneurs peuvent se communiquer sur tous les ports, même si les ports ne sont pas exposés au cluster,
- le même espace de stockage,
  - les conteneurs peuvent partager les mêmes volumes.

5.2 - Mise en Place

Commencez par créer le fichier **multicontainerpod.yaml** :

```
root@kubemaster:~# vi multicontainerpod.yaml
root@kubemaster:~# cat multicontainerpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: multicontainerpod
spec:
  containers:
  - name: nginx
    image: nginx
  - name: redis
    image: redis
  - name: couchbase
    image: couchbase
```

Important : Notez que le fichier créera trois conteneurs - **nginx**, **redis** et **couchbase**.

Créez ensuite le pod :

```
root@kubemaster:~# kubectl create -f multicontainerpod.yaml
pod/multicontainerpod created
```

Consultez l'état du pod :

```
root@kubemaster:~# kubectl get pod multicontainerpod
NAME                READY   STATUS              RESTARTS   AGE
multicontainerpod   0/3     ContainerCreating   0          65s
```

Important : Notez qu'il y a actuellement 0 de 3 pods dans un état de READY.

Attendez quelques minutes et constatez de nouveau l'état du pod :

```
root@kubemaster:~# kubectl get pod multicontainerpod
NAME                READY   STATUS    RESTARTS   AGE
multicontainerpod   3/3     Running   0          16m
```

Important : Notez qu'il y a actuellement 3 de 3 pods dans un état de READY.

Créez maintenant le fichier **helper.yaml** :

```
root@kubemaster:~# vi helper.yaml
root@kubemaster:~# cat helper.yaml
apiVersion: v1
kind: Pod
metadata:
  name: helperpod
spec:
  containers:
  - name: busybox1
    image: busybox
    command: ['sh', '-c', 'while true; do echo logs data > /output/output.log; sleep 5; done']
    volumeMounts:
      - name: sharedvol
        mountPath: /output
  - name: helper
    image: busybox
    command: ['sh', '-c', 'tail -f /input/output.log']
    volumeMounts:
      - name: sharedvol
        mountPath: /input
  volumes:
```

```
  - name: sharedvol
    emptyDir: {}
```

Important : Notez que ce fichier créera un pod contenant deux conteneurs - **busybox1** et **helper**. Chaque conteneur partage un volume identique qui s'appelle **sharedvol**. Dans le conteneur ***busybox1** ce volume est monté sur **/output** tandis que dans le conteneur **helper**, le même volume est monté sur **/input**.

Créez le pod **helper** :

```
root@kubemaster:~# kubectl create -f helper.yaml
pod/helperpod created
```

Consultez les logs du conteneur **helper** dans le pod **helperpod** :

```
root@kubemaster:~# kubectl logs helperpod -c helper
logs data
```

Important : Notez que le conteneur **busybox1** a écrit la chaîne **logs data** dans le fichier **/output/output.log** tous les 5 secondes grâce à l'exécution de la commande **while true; do echo logs data > /output/output.log; sleep 5; done**. Le conteneur **helper** exécute la commande **tail -f /input/output.log**. Le log du conteneur **helper** contient donc la chaîne **logs data** issu du fichier **output.log** car ce fichier est partagé entre les deux conteneurs.

LAB #6 - Conteneurs Init

6.1 - Présentation

Un Conteneur Init est un conteneur qui ne s'exécute qu'une seule fois au démarrage du pod. S'il existe plusieurs conteneurs Init, ceux-ci s'exécutent dans l'ordre. Un conteneur Init doit terminer son exécution avant que le conteneur Init suivant, ou l'application si le conteneur Init concerné est le dernier, peut s'exécuter. Le but d'un conteneur Init est d'exécuter du code qui n'a pas besoin de se trouver dans les conteneurs de l'application afin rendre plus légers ces derniers, par exemple :

- isoler, d'une manière sécurisée, des données sensibles tels des mots de passe afin d'éviter à ce que celles-ci soient compromises si un conteneur de l'application est compromis,
- injecter des données dans un volume partagé,
- faire patienter un pod en attendant que d'autres ressources de K8s soient créées.

6.2 - Mise en Place

Commencez par créer le fichier **initpod.yaml** :

```
root@kubemaster:~# vi initpod.yaml
root@kubemaster:~# cat initpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: initpod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
  initContainers:
  - name: delay
    image: busybox
```

```
    command: ['sleep', '30']
```

Important : Notez que le conteneur **delay** va faire patienter la création du conteneur **nginx** pendant 30 secondes.

Créez le pod **initpod** :

```
root@kubemaster:~# kubectl create -f initpod.yaml
pod/initpod created
```

Consultez l'état du pod :

```
root@kubemaster:~# kubectl get pod initpod
NAME      READY   STATUS     RESTARTS   AGE
initpod   0/1     Init:0/1   0          6s
```

Important : Notez que le **STATUS** du pod est **Init:0/1**.

Patientez au moins 30 secondes puis exécutez la dernière commande de nouveau :

```
root@kubemaster:~# kubectl get pod initpod
NAME      READY   STATUS    RESTARTS   AGE
initpod   1/1     Running   0          79s
```

Important : Notez que le **STATUS** du pod est **Running**.

LAB #7 - Scheduling

7.1 - Présentation

Scheduling est le processus d'attribution de pods aux noeuds. Ce processus est accompli par le **Scheduler**, un composant du **Control Plane**.

Le Scheduler prend sa décision en fonction d'un un contrôle :

- des ressources disponibles sur les neouds en fonction des **Resource Resquests**,
- des configurations des **nodeSelectors** qui utilisent des **Node Labels**,
- des instructions de type **nodeName** qui forcent le choix d'un noeud par rapport à un autre.

7.2 - Mise en Place

Commencez par visualiser les noeuds du cluster :

```
root@kubemaster:~# kubectl get nodes
NAME                        STATUS   ROLES           AGE   VERSION
kubemaster.ittraining.loc   Ready    control-plane   11d   v1.25.0
kubenode1.ittraining.loc    Ready    <none>          11d   v1.25.0
kubenode2.ittraining.loc    Ready    <none>          11d   v1.25.0
```

nodeSelector

Attribuez l'étiquette **mylabel=thisone** au noeud **kubenode1.ittraining.loc** :

```
root@kubemaster:~# kubectl label nodes kubenode1.ittraining.loc mylabel=thisone
node/kubenode1.ittraining.loc labeled
```

Créez maintenant le fichier **nodeselector.yaml** :

```
root@kubemaster:~# vi nodeselector.yaml
root@kubemaster:~# cat nodeselector.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeselector
spec:
  nodeSelector:
    mylabel: "thisone"
  containers:
  - name: nginx
    image: nginx:1.19.1
```

Important : Notez l'entrée **nodeSelector**.

Créez le pod **nodeselector** :

```
root@kubemaster:~# kubectl create -f nodeselector.yaml
pod/nodeselector created
```

Constatez l'emplacement du pod **nodeselector** :

```
root@kubemaster:~# kubectl get pod nodeselector -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP               NODE                        NOMINATED NODE
READINESS GATES
nodeselector   1/1     Running   0          66s   192.168.239.21   kubenode1.ittraining.loc   <none>
<none>
```

Important : Notez que le pod nodeselector a été schedulé sur le noeud **kubenode1**.

nodeName

Créez maintenant le fichier **nodename.yaml** :

```
root@kubemaster:~# vi nodename.yaml
root@kubemaster:~# cat nodename.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodename
spec:
  nodeName: kubenode2.ittraining.loc
  containers:
    - name: nginx
      image: nginx:1.19.1
```

Important : Notez que le pod va être schedulé sur **kubenode2.ittraining.loc** grâce à l'utilisation de **nodeName**.

Créez le pod **nodename** :

```
root@kubemaster:~# kubectl create -f nodename.yaml
pod/nodename created
```

Constatez l'emplacement du pod **nodename** :

```
pod/nodename created
root@kubemaster:~# kubectl get pod nodename -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP              NODE                      NOMINATED NODE
READINESS GATES
```

```
nodename   1/1     Running   0          67s   192.168.150.25   kubenode2.ittraining.loc   <none>           <none>
```

Important : Notez que le pod a été schedulé sur **kubenode2.ittraining.loc** grâce à l'utilisation de **nodeName**.

LAB #8 - DaemonSets

8.1 - Présentation

Un DaemonSet :

- crée une copie d'un pod sur tous les noeuds disponibles,
- crée une copie d'un pod sur tout nouveau noeud ajouté au cluster,
- respecte les contraintes de Node Labels.

8.2 - Mise en Place

Commencez par nettoyer le cluster :

```
root@kubemaster:~# kubectl delete --all pods --namespace=default
pod "alwayspod" deleted
pod "bigrequestpod" deleted
pod "helperpod" deleted
pod "initpod" deleted
pod "liveness-pod" deleted
pod "livenesspodhttp" deleted
pod "multicontainerpod" deleted
pod "my-deployment-67b5d4bf57-6wcrq" deleted
```

```
pod "myapp-deployment-689f9d59-c25f9" deleted
pod "myapp-deployment-689f9d59-nn9sw" deleted
pod "myapp-deployment-689f9d59-rnc4r" deleted
pod "never" deleted
pod "nodename" deleted
pod "nodeselector" deleted
pod "onfailure" deleted
pod "readinesspod" deleted
pod "resourcepod" deleted
pod "startuppod" deleted

root@kubemaster:~# kubectl delete --all deployments --namespace=default
deployment.apps "my-deployment" deleted
deployment.apps "myapp-deployment" deleted
```

Créez ensuite le fichier **daemonset.yaml** :

```
root@kubemaster:~# vi daemonset.yaml
root@kubemaster:~# cat daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mydaemonset
spec:
  selector:
    matchLabels:
      app: mydaemonset
  template:
    metadata:
      labels:
        app: mydaemonset
    spec:
      containers:
      - name: nginx
```

```
        image: nginx:1.19.1
```

Créez le DaemonSet **mydaemonset** :

```
root@kubemaster:~# kubectl create -f daemonset.yaml
daemonset.apps/mydaemonset created
```

Constatez le statut du **DaemonSet** :

```
root@kubemaster:~# kubectl get daemonset
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
mydaemonset   2         2         2       2            2           <none>          37s
```

Constatez maintenant qu'il a un pod sur chaque noeud :

```
root@kubemaster:~# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP               NODE                        NOMINATED NODE
READINESS GATES
mydaemonset-hmdhp   1/1     Running   0          38s   192.168.239.26   kubenode1.ittraining.loc   <none>
<none>
mydaemonset-kmf4z   1/1     Running   0          38s   192.168.150.30   kubenode2.ittraining.loc   <none>
<none>
```

Important : Notez qu'il n'y ait pas de pod sur **kubemaster**. En effet, le kubemaster a le drapeau **no taint** fixé qui empêche la création de pods sur lui.

LAB #9 - Pods Statiques

9.1 - Présentation

Un Static Pod (*Pod Statique*) est :

- un pod qui est contrôlé par le **kubelet** sur le noeud concerné au lieu d'être contrôler par l'API de K8s,
  - ce type de pod peut être créé même s'il n y'ait pas de Control Plane,
  - si le Control Plane existe, un **Mirror Pod** (*Pod Miroir*) est créé dans le Control Plane pour représenter le pod statique afin de faciliter la consultation son statut. Par contre, le pod ne peut ni être changé, ni être géré à partir du Control Plane,
- un pod créé en utilisant un fichier yaml situé dans un chemin **spécifique** sur le noeud concerné,
  - pour un cluster installé avec **kubeadm**, le chemin “spécifique” par défaut dans chaque **worker** est **/etc/kubernetes/manifests**. Notez qu'il est possible de modifier ce chemin.

9.2 - Mise en Place

Connectez-vous à kubenode1 et devenez l'utilisateur **root** :

```
root@kubemaster:~# ssh -l trainee 192.168.56.3
trainee@192.168.56.3's password: trainee
Linux kubenode1.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep  4 13:01:18 2022 from 192.168.56.2
trainee@kubenode1:~$ su -
Mot de passe : fenestros
root@kubenode1:~#
```

Créez le fichier **/etc/kubernetes/manifests/mystaticpod.yaml** :

```
/etc/kubernetes/manifests
```

Créez le pod **mystaticpod** :

```
root@kubenode1:~# vi /etc/kubernetes/manifests/mystaticpod.yaml
root@kubenode1:~# cat /etc/kubernetes/manifests/mystaticpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mystaticpod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
```

Important : Notez que kubelet va voir que le fichier a été créé et ensuite pouruivra avec la création du pod.

Re-démarrez le service **kubelet** pour démarrer le pod statique **immédiatement** sans attendre :

```
root@kubenode1:~# systemctl restart kubelet
```

Retournez au **kubemaster** et constatez la présence d'un pod miroir :

```
root@kubenode1:~# exit
déconnexion
trainee@kubenode1:~$ exit
déconnexion
Connection to 192.168.56.3 closed.

root@kubemaster:~# kubectl get pods
```

```
NAME                                READY   STATUS    RESTARTS   AGE
mydaemonset-hmdhp                   1/1     Running   0          32m
mydaemonset-kmf4z                   1/1     Running   0          32m
mystaticpod-kubenode1.ittraining.loc   1/1     Running   0          3m40s
```

Supprimez maintenant le pod statique :

```
root@kubemaster:~# kubectl delete pod mystaticpod-kubenode1.ittraining.loc
pod "mystaticpod-kubenode1.ittraining.loc" deleted
```

Important : Notez que la suppression semble avoir réussi.

Constatez les pods en cours d'exécution :

```
root@kubemaster:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mydaemonset-hmdhp                   1/1     Running   0          45m
mydaemonset-kmf4z                   1/1     Running   0          45m
mystaticpod-kubenode1.ittraining.loc   1/1     Running   0          19s
```

Important : Notez que le pod **mystaticpod-kubenode1.ittraining.loc** est revenu. En effet, la suppression précédente n'a supprimé que le miroir qui a ensuite ête regénéré.

Pour supprimer le pod statique, connectez-vous à **kubenode1** :

```
root@kubemaster:~# ssh -l trainee kubenode1
trainee@kubenode1's password: trainee
Linux kubenode1.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 15 17:51:03 2022 from 192.168.56.2

trainee@kubenode1:~$ su -
Mot de passe : fenestros

root@kubenode1:~# rm -f /etc/kubernetes/manifests/mystaticpod.yaml

root@kubenode1:~# systemctl restart kubelet

root@kubenode1:~# exit
déconnexion

trainee@kubenode1:~$ exit
déconnexion
Connection to kubenode1 closed.

root@kubemaster:~#
```
