

Formation Kubernetes

-

Pratique



kubernetes

Atelier 3

-

Création et manipulation des Namespaces et Pods

Table des matières

| | |
|---|----|
| 1. Namespaces | 3 |
| 1.1. Création de namespaces | 3 |
| 2. Pods | 5 |
| 2.1. Pod mono-container | 5 |
| 2.1.1. Création d'un pod | 5 |
| 2.2. Pod multi-container | 10 |
| 2.2.1. Création du pod | 11 |
| 2.2.2. Manipulations des containers | 12 |
| 2.2.3. Arrêt d'un pod | 13 |

1. Namespaces

Un namespace **Kubernetes** (à ne pas confondre avec un namespace **Linux**) peut être vu comme un cluster virtuel.

Il est notamment possible de lui associer des quotas (limite maximale de ressources utilisables), et des droits définis pour chaque utilisateur (ex : droit de création de certaines ressources, droits de lecture, etc...).

Un namespace peut donc être utilisé pour :

- Séparer de manière logique des environnements
 - Par exemple, un namespace pour le développement, un pour la production etc...
 - Cependant, sans configuration supplémentaire, un namespace ne permet pas de créer une réelle isolation des ressources. En effet, par défaut un pod/service d'un namespace A peut communiquer avec un pod/service d'un namespace B !
- Répartir les ressources disponibles au sein du cluster
 - Par le biais de création de quotas

Les ressources **Kubernetes** se trouvant dans un namespace doivent avoir un nom unique. Cependant, il est possible d'avoir des ressources avec le même nom dans deux namespaces différents.

1.1. Création de namespaces

Pour commencer, créez un dossier « *atelier3/* » dans lequel vous stockerez l'ensemble des ressources utilisées pour cet atelier.

1) L'objectif va être de créer un nouveau namespace grâce au fichier **json** suivant.

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "formation"
  }
}
```

Note : Ici, le format utilisé est **json** mais par la suite le format qui sera utilisé est **yaml**, car celui est plus courant. Sachez simplement que les 2 formats sont acceptés par l'API server.

2) Ensuite, créez le namespace :

```
$ kubectl create -f <fichier.json>
```

Remarque :

- Il est également possible d'utiliser la commande **kubectl create namespace lab**
 - Avec la CLI, aucune option supplémentaire autre que le nom du namespace ne peut être ajoutée.

3) Visionnez le namespace :

```
$ kubectl get namespace
NAME          STATUS    AGE
default       Active   1d
formation     Active   8s
kube-public   Active   1d
kube-system   Active   1d
```

Pour rappel, **Kubernetes** démarre avec les trois namespaces suivants :

- **default** : le namespace par défaut pour les objets sans namespace défini
- **kube-system** : le namespace dédié pour les objets créés par le système **Kubernetes**
- **kube-public** : ce namespace est accessible en lecture par tous les utilisateurs (même ceux non authentifiés).

Comme vu lors de l'atelier 2, l'option « **-n** » permet de sélectionner un namespace particulier. Si l'option n'est pas spécifiée, la commande s'exécutera pour le namespace **default**.

Par exemple si vous souhaitez lister les pods se trouvant dans le namespace **kube-system** il vous suffit de lancer la commande suivante :

```
$ kubectl get pods -n kube-system
```

Vous noterez que dans le cas où de multiples namespaces sont créés au sein de votre cluster, la précision de l'option **-n** pour chaque commande peut devenir contraignant.

C'est pourquoi il est possible de fixer un namespace particulier pour l'ensemble des commandes **kubectl** qui suivront dans votre contexte actuel :

```
$ kubectl config set-context $(kubectl config current-context) --namespace=kube-system
```

Remarque :

- La notion de contexte se comprend par rapport à l'utilisateur et au cluster sur lequel kubectl interagit.

4) Exécutez la commande suivante :

```
$ kubectl get pods
```

Que remarquez-vous ?

5) Fixez le contexte sur le namespace **formation** précédemment créé :

```
$ kubectl config set-context $(kubectl config current-context) --namespace=formation
```

Désormais, l'ensemble des manipulations suivantes seront appliquées à ce namespace « formation ».

2. Pods

2.1.Pod mono-container

2.1.1. Création d'un pod

1) Créez un pod grâce à la commande « **kubectl run** » :

```
$ kubectl run nginx --image=nginx  
deployment.apps "nginx" created
```

Vous pouvez également créer un simple pod à partir d'un fichier **yaml** de ce type :

```
apiVersion: v1 # spécification de la version de l'api utilisée  
kind: Pod # type de ressource à créer, ici un pod  
metadata:  
  name: apache # nom de la ressource, celui-ci doit être unique dans  
  un même namespace  
  namespace: default # namespace dans lequel déployer la ressource
```

```
spec: # définition des spécifications du pod
  containers:
    - name: apache # nom du container visible dans Kubernetes
      image: httpd # image du container à utiliser
      ports:
        - containerPort: 80 # port du container à exposer
```

2) Affichez la liste des pods pour valider le bon démarrage de celui-ci :

```
$ kubectl get pods
```

| NAME | READY | STATUS | RESTARTS | AGE |
|----------------------|-------|---------|----------|-----|
| nginx-8586cf59-rq87c | 1/1 | Running | 0 | 10s |

3) Maintenant, exécutez la commande suivante pour obtenir une description complète du pod au format **yaml** :

```
$ kubectl get pod nginx-8586cf59-rq87c -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/limit-ranger: 'LimitRanger plugin set: cpu request
for container
  nginx'
  creationTimestamp: 2018-10-09T13:55:58Z
  generateName: nginx-8586cf59-
  labels:
    pod-template-hash: "41427915"
    run: nginx
  name: nginx-8586cf59-rq87c
  namespace: default
  ownerReferences:
    - apiVersion: extensions/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ReplicaSet
      name: nginx-8586cf59
```

```
uid: 0c6062a5-cbcb-11e8-a3f3-42010a8402c3
resourceVersion: "29116"
selfLink: /api/v1/namespaces/default/pods/nginx-8586cf59-rq87c
uid: 0c6323cb-cbcb-11e8-a3f3-42010a8402c3
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
    resources:
      requests:
        cpu: 100m
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-vh72b
      readOnly: true
  dnsPolicy: ClusterFirst
  nodeName: gke-k8s-lab-default-pool-4fd48f3f-r0w2
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
```

```
volumes:
- name: default-token-vh72b
  secret:
    defaultMode: 420
    secretName: default-token-vh72b
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2018-10-09T13:55:58Z
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: 2018-10-09T13:55:59Z
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: 2018-10-09T13:55:58Z
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: docker://3a4fc45cfcad82482db1241c4de781071f7619c0f38c68072462362888f1b94e
    image: nginx:latest
    imageID: docker-pullable://nginx@sha256:204a9a8e65061b10b92ad361dd6f406248404fe60efd5d6a8f2595f18bb37aad
    lastState: {}
    name: nginx
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2018-10-09T13:55:59Z
    hostIP: 10.132.0.2
    phase: Running
```



```
podIP: 10.28.2.7
qosClass: Burstable
startTime: 2018-10-09T13:55:58Z
```

Ci-dessous l'explication des trois principales sections de la définition YAML :

- **Metadata**

- Contient notamment les métadonnées suivantes à propos du pod : la date de création, le nom, le namespace dans lequel celui-ci est déployé, l'uid de la ressource...

- **Spec**

- Contient notamment les informations suivantes à propos du pod : informations concernant l'image des containers, les limitations/réservations de ressources associées, le nom de la node sur laquelle il est déployé...

- **Status**

- Contient notamment les informations suivantes à propos du pod : statut du ou des container(s), adresse IP du nœud qui héberge le pod, adresse IP du pod, statut du pod...

- 4) Maintenant que vous avez validé que le pod est démarré et fonctionnel, connectez-vous maintenant au container grâce à la commande suivante :

```
$ kubectl exec -it nginx-8586cf59-rq87c -- /bin/bash
root@nginx-8586cf59-rq87c:/#
root@nginx-8586cf59-rq87c:/# hostname
nginx-8586cf59-rq87c
```

- 5) Vous êtes désormais à l'intérieur du container et pouvez donc exécuter les commandes habituelles.

Par exemple, vous allez lister les processus qui s'exécutent dans le container avec la commande **ps** :

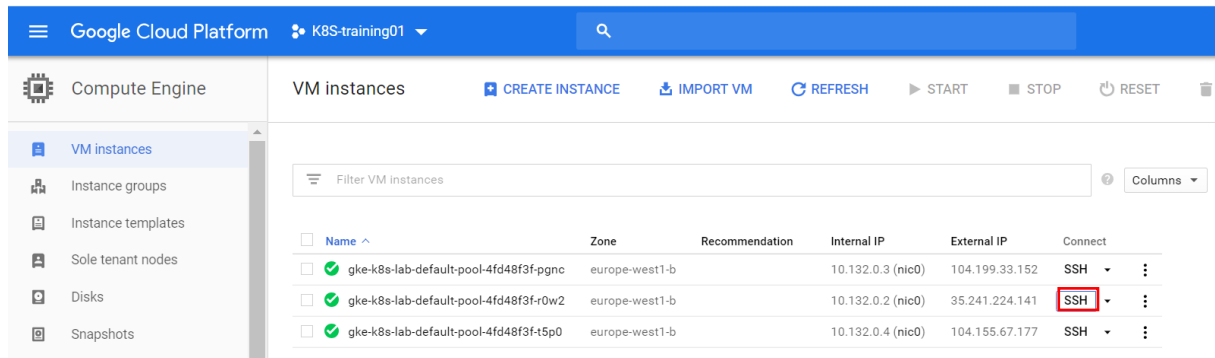
```
root@nginx-8586cf59-rq87c:/# apt-get update
...
root@nginx-8586cf59-rq87c:/# apt-get install -y procps
...
root@nginx-8586cf59-rq87c:/# ps aux
```

| USER | PID | %CPU | %MEM | VSZ | RSS | TTY | STAT | START | TIME |
|--|-----|------|------|-------|------|-----|------|-------|---------|
| COMMAND | | | | | | | | | |
| root | 1 | 0.0 | 0.1 | 32624 | 5156 | ? | Ss | 13:55 | 0:00 |
| nginx: master process nginx -g daemon off; | | | | | | | | | |
| nginx | 5 | 0.0 | 0.0 | 33076 | 2352 | ? | S | 13:55 | 0:00 |
| nginx: worker process | | | | | | | | | |
| root | 6 | 0.0 | 0.0 | 4292 | 720 | ? | Ss+ | 14:00 | 0:00 |
| /bin/sh | | | | | | | | | |
| root | 10 | 0.0 | 0.0 | 18136 | 3216 | ? | Ss | 14:00 | 0:00 |
| /bin/bash | | | | | | | | | |
| root | 257 | 0.0 | 0.0 | 36640 | 2812 | ? | R+ | 14:14 | 0:00 ps |
| aux | | | | | | | | | |

6) Repérez la node sur laquelle votre pod s'exécute, pour ensuite s'y connecter :

```
$ kubectl get pods -o wide
```

7) Retournez sur la console GCP et connectez-vous à l'instance correspondant au nœud en cliquant sur « **SSH** » :



| Name | Zone | Recommendation | Internal IP | External IP | Connect |
|--|----------------|----------------|-------------------|----------------|---------|
| gke-k8s-lab-default-pool-4fd48f3f-pgnc | europa-west1-b | | 10.132.0.3 (nic0) | 104.199.33.152 | SSH |
| gke-k8s-lab-default-pool-4fd48f3f-r0w2 | europa-west1-b | | 10.132.0.2 (nic0) | 35.241.224.141 | SSH |
| gke-k8s-lab-default-pool-4fd48f3f-t5p0 | europa-west1-b | | 10.132.0.4 (nic0) | 104.155.67.177 | SSH |

8) Une fois la console SSH lancée, exécutée la commande suivante :

```
$ ps -aux |grep nginx
```

Que remarquez-vous ?

2.2.Pod multi-container

Lors de l'atelier précédent, vous avez vu comment créer un pod grâce à la commande « **kubectl run** ». Cependant, l'utilisation de cette commande ne permet pas d'accéder à l'ensemble des paramètres de configuration.

De plus, en utilisant des fichiers **yaml**, il est possible de stocker et conserver différentes versions des déploiements.

Pour rappel, les conteneurs au sein d'un pod partagent le stockage, un même linux namespace, un même cgroups et ont la même adresse IP.

2.2.1. Création du pod

Pour cet exercice, vous allez donc utiliser un pod contenant deux containers, dont la définition est présente ci-dessous :

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    name: fortune
spec:
  containers:
  - name: web
    image: ilkiinformation/webapache:v1
    ports:
      - containerPort: 80
      - containerPort: 22
  - name: generator
    image: ilkiinformation/fortunegen:v1
```

Comme expliqué précédemment, ce pod contient deux containers et fonctionne de la manière suivante :

- Un container nommé « web », qui affiche une page HTML,
- Un container nommé « generator », qui génère une page HTML toutes les 5 secondes et qui l'upload sur le serveur « web ».

1) Créez le fichier **yaml** correspondant au pod et créez la ressource

```
kubectl create -f pod-fortune.yaml
```

2) Vérifiez la bonne création du pod et le bon lancement des containers

```
kubectl get pod -o wide
```

Vous devriez donc voir dans la deuxième colonne la valeur « 2/2 ».

- 3) Repérez l'adresse IP du pod et la node sur laquelle il est déployé. Connectez-vous sur la node et essayez d'effectuer une requête HTTP sur l'adresse IP du pod.

```
curl http://10.28.1.6
```

Vous devriez apercevoir une phrase/citation en réponse à cette requête.

- 4) Testez la même opération depuis une autre node du cluster.
Que remarquez-vous ?

2.2.2. Manipulations des containers

- 1) Retournez sur la node hébergeant le pod, puis repérez les containers associés.

```
docker ps |grep fortune
```

- 2) Connectez-vous au container « web »

```
docker exec -it 49bfc971673f /bin/bash
```

- 3) Une fois dans le container, effectuez les commandes « *cat /etc/hosts* ». Que remarquez-vous ?
- 4) Répétez l'opération sur le container « generator ». Que remarquez-vous ?
- 5) Listez les processus s'exécutant dans les deux containers. Que remarquez-vous ?
- 6) Enfin, listez les processus démarrés sur la node. Retrouvez-vous l'ensemble des processus des deux containers ?

Vous remarquez donc que par défaut, les containers n'ont la visibilité que sur leurs propres processus, et non ceux des autres containers dans le même pod.

Depuis la version **Kubernetes** 1.10, il est possible de partager les processus entre les containers d'un même pod. La fonctionnalité se nomme « *ShareProcessNamespace* » et est encore en version beta à ce jour.

<https://kubernetes.io/docs/tasks/configure-pod-container/share-process-namespace/#configure-a-pod>

2.2.3. Arrêt d'un pod

L'objectif est d'observer le comportement de **Kubernetes** lors d'un arrêt de containers ou pods.

- 1) Commencez par lancer dans un terminal la commande suivante :

```
kubectl get po --watch
```

- 2) Puis connectez-vous sur la node hébergeant votre pod et forcez la suppression du container « web ». Continuez d'observer en parallèle l'autre terminal.

```
docker rm -f 94f9dc210c49
```

Que remarquez-vous ?

Pensez à observer les derniers évènements associés aux pods avec la commande :

```
kubectl describe pod fortune
```

- 3) Vous avez remarqué que **Kubernetes** se charge de redémarrer les containers présents dans un pod en cas de défaillance. Maintenant, testez la suppression d'un pod et observez le comportement :

```
kubectl delete pod fortune
```

Dans ce cas, vous pouvez voir que le pod n'est pas redémarré.

Cependant, **Kubernetes** possède des fonctionnalités permettant de redémarrer automatiquement les pods en cas d'arrêt ou défaillance, gérées par les *ReplicaSets* que nous verrons dans un prochain atelier.