# TP02 | Kubernetes - Déploiement

Déploiement d'applications sur Kubernetes

Cours IUT Ifs - R509

Dernière mise à jour : 24 octobre 2025

**Tags :** `docker` | `container` | `kubernetes` | `tp02`

## 1 Déroulement du TP

L'objectif du TP est de vous donner tous les éléments nécessaires afin de déployer votre application sur Kubernetes.

Il y a des questions lors de ce TP affichées par le symbole **?**.

Vous **DEVEZ** faire un compte-rendu et l'envoyer sur l'adresse mail de l'enseignant encadrant votre séance de TP.

⚠ Important

**LISEZ-BIEN TOUT LE TP**

## 2 Prérequis

- ✅ Avoir terminé le TP01
- ✅ Accès au proxy IUT
- ✅ Votre VM utilisée au TP01 en lui ajoutant 2 cœurs
- ✅ Cluster kind avec 2 control-plane + 1 worker :
  - Forward le port 80 :80 et 443 :443
  - Mettre un `node-labels: "ingress-ready=true"` sur le nœud où il y a le port forward
  - Déployer un nginx sur votre cluster → Fin de TP01
  - ⚠ Pensez bien au proxy...

## 3 Focus sur les commandes kubectl

L'outil en ligne de commande de Kubernetes, `kubectl`, vous permet d'exécuter des commandes dans les clusters Kubernetes. Vous pouvez utiliser `kubectl` pour déployer des applications, inspecter et gérer les ressources du cluster et consulter les logs.

### 3.1 Après l'installation de votre cluster

```
kubectl get nodes
```

Sortie attendue :

```
NAME                 STATUS   ROLES           AGE     VERSION
kind-control-plane   Ready    control-plane   3m8s    v1.25.3
kind-control-plane2  Ready    control-plane   2m43s   v1.25.3
kind-worker          Ready    <none>          113s    v1.25.3
```

Décomposons la commande que nous avons lancée pour avoir une idée de ce qui se passe :

— `get` - c'est l'une des nombreuses commandes ou "verbes" de kubectl

— `nodes` - l'objet cible de la commande get

La sortie que nous avons reçue de Kubernetes nous indique qu'il y a 3 nœuds, qu'ils sont tous "prêts" (le kubelet sur chaque nœud est en place et connecté à l'API Kubernetes), quels sont leurs rôles (pas tout à fait important pour la plupart des utilisateurs), quel est leur âge et quelle est la version de Kubernetes qu'ils exécutent.

### 3.2 Verbes kubectl courants

Voici quelques-uns des verbes les plus courants que vous utiliserez avec `kubectl` :

— `get` - Affiche une ressource

— `describe` - Affiche des détails spécifiques sur une ou plusieurs ressources

— `create` - Crée une ressource (ou à partir d'un fichier avec `-f`)

— `apply` - Applique un manifeste

— `delete` - Supprime une ressource (ou un fichier avec `-f`)

## 4 Informations à propos du cluster

Maintenant que les verbes les plus courants sont connus, voyons ce que nous pouvons trouver d'autre dans le cluster.

**Astuce**

```
kubectl get all
```

**Question**

1. Afficher la liste des namespaces du cluster
   — Combien de namespaces contient le cluster ?
2. Afficher la liste de la "plupart" des objets dans le namespace `kube-system`
   — Quels objets vous afficher dans le namespace ?
   — Quelle adresse IP le service `kubernetes` a-t-il ?

## 5 Les Objets Kubernetes

Tous les objets Kubernetes peuvent être visualisés de la même manière et de différentes façons. Examinons un objet, le Service kubernetes dans le namespace par `default`.

**Note**

Le namespace `default` est exactement ce qu'il semble être. S'il n'y a pas de contexte défini, kubectl ciblera toujours le namespace par défaut. Ce n'est pas une bonne pratique de mettre vos applications ici, mais il y a beaucoup de guides sur Internet qui vous "enseignent" de mettre vos applications ici. S'il vous plaît, ne faites pas cela et utilisez les namespaces comme ils sont là pour vous sauver. Si vous avez envie de supprimer votre namespace avec un `kubectl delete all`, cela supprimera également le service kubernetes, ce qui pourrait avoir des effets secondaires peu souhaitables.

**Astuce**

```
get ... -o yaml
```

**Question**

Afficher le service kubernetes dans le namespace par défaut en yaml

1. Quelle est la version de l'API de l'objet kubernetes ?
2. Quel est le type d'objet ?
3. Quelles sont les labels de cet objet ?

## 6 Où vivent les objets ?

Nous avons donc examiné les objets Kubernetes et la façon de les obtenir et de les décrire, mais nous n'avons pas vraiment discuté de l'endroit où ils vivent.

Jusqu'à présent, chaque objet était un objet au niveau du namespace, mais il y a des objets dans un cluster Kubernetes qui ne sont pas au niveau du namespace et qui vivent au niveau du cluster.

Ces objets ont un flag dans leur définition qui indique `namespaced: false`.

Pour obtenir une liste des objets Kubernetes dans votre cluster qui peuvent être créés/appliqués, vous pouvez exécuter la commande `kubectl api-resources -verbs=list` pour les identifier. Vous pouvez également les séparer en `namespaced=true` ou `namespaced=false` pour montrer quels objets sont destinés à quelle zone d'un cluster.

**Question**

Question :

— Quelle est la différence entre une api-resources namespaced true/false. Citer un exemple.

## 7 Déploiement d'une première application

Nous connaissons donc les objets Kubernetes et savons comment examiner leurs spécifications pour déterminer ce qu'ils sont et ce qu'ils font, mais nous n'avons rien vu d'autre que ce dont Kubernetes lui-même a besoin.

Comme mentionné précédemment, Kubernetes peut facilement être décomposé en 3 composants principaux d'un "cloud" (Compute, Storage, Networking), alors regardons les manifestes qui génèrent ces composants.

Au lieu de plonger directement dans le déploiement d'un serveur Minecraft, nous allons examiner un serveur VS Code pour nous familiariser avec d'autres composants Kubernetes. En règle générale, vous devriez nommer vos fichiers manifestes en fonction de ce qu'ils sont, mais pour comparer les trois principaux composants, je les ai nommés en fonction de ce qu'ils "font".

Il s'agit simplement de VS Code dans un navigateur (un projet vraiment cool) fonctionnant dans un conteneur.

## 7.1 Compute Manifest

Dans le fichier `compute.yaml`, nous avons un seul objet qui est notre Déploiement. Un `Deployment` est responsable du déploiement d'un Pod qui peut contenir n'importe quel nombre de conteneurs.

Cet exemple particulier n'a qu'un seul conteneur qui utilise l'image d'un serveur VS Code qui peut être exécuté dans un navigateur.

Cette image est hébergée sur Quay.io et lorsque le Pod va démarrer, le kubelet sur le nœud où le Pod a été planifié va essentiellement exécuter un `docker pull` pour obtenir l'image afin qu'il puisse ensuite démarrer le conteneur. Quelques éléments à noter dans ce manifeste sont les sections `volumeMounts`, `volumes`, et `env`.

*Enregistrer le fichier `compute.yaml` dans un dossier nommé `vs_code`*

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: code-server
  name: code-server
spec:
  selector:
    matchLabels:
      app: code-server
  replicas: 1
  template:
    metadata:
      labels:
        app: code-server
    spec:
      containers:
      - env:
        - name: PASSWORD
          value: CHANGEME
        image: codercom/code-server:latest
        imagePullPolicy: Always
        name: code-server
        ports:
        - name: code-server
          containerPort: 8080
          protocol: TCP
        volumeMounts:
        - mountPath: /home/coder
          name: coder
      initContainers:
      - name: pvc-permission-fix
        image: busybox
        command: ["/bin/chmod","-R","777", "/home/coder"]
        volumeMounts:
        - name: coder
```

```
38          mountPath: /home/coder
39      volumes:
40      - name: coder
41        persistentVolumeClaim:
42          claimName: code-server
```

Listing 1 – compute.yaml

**? Question**

Questions :

— À quoi sert la section `env` ?

— À quoi sert la section `volume` et `volumemount` ?

### 7.2 Storage Manifest

Notre manifeste de stockage est assez simple comme vous pouvez le voir dans `storage.yaml`. Il y a un seul objet appelé `PersistentVolumeClaim` qui, selon sa spécification, demandera 5Gi de stockage.

Le "comment" est propre à chaque plateforme, mais un `PersistentVolumeClaim` demandera au cluster un PersistentVolume de la même taille que le Claim demandé et, s'il n'en existe pas, il se tournera vers la `StorageClass` pour en créer un.

Une `StorageClass` est un provisionneur qui réside dans un cluster pour atteindre une plateforme de stockage et créer un partage/volume/disque qui est ensuite présent dans le cluster en tant que PersistentVolume.

Un cluster peut avoir plus d'une `StorageClass` et l'une d'entre elles peut être définie comme la valeur par défaut à partir de laquelle tous les volumes persistants sont créés, à moins que vous ne spécifiiez dans le `PersistentVolumeClaim` la `StorageClass` à utiliser.

S'il n'y a pas de `StorageClass` dans un cluster, vous aurez un `PersistentVolumeClaim` perpétuellement bloqué dans l'état Pending.

*Enregistrer le fichier `storage.yaml` dans votre dossier `vs_code`*

```
1  ---
2  apiVersion: v1
3  kind: PersistentVolumeClaim
4  metadata:
5    name: code-server
6  spec:
7    accessModes:
8    - ReadWriteOnce
9    resources:
10     requests:
11       storage: 5Gi
```

Listing 2 – storage.yaml

**? Question**

Questions :

— Pourquoi créer un PVC ?

### 7.3 Network Manifest

Le fichier `networking.yaml` contient 2 objets, un Service et un Ingress.

Ce manifeste est divisé par la syntaxe YAML de `--` qui crée une "pause" et indique au moteur de rendu (dans notre cas `kubectl`) où commence l'objet suivant.

*Enregistrer le fichier* ***`network.yaml`*** *dans votre dossier* ***`vs_code`***

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: code-server
  labels:
    app: code-server
spec:
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
  selector:
    app: code-server
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: code-server
  labels:
    app: code-server
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: "mon-app.local"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: code-server
            port:
              number: 8080
```

Listing 3 – network.yaml

**? Question**

✅ Déployer les manifests dans le dossier `vs_code` sur votre cluster
✅ Faites en sorte d'accéder à votre application sur votre navigateur
Questions :

- Comment accédez-vous à l'application `mon-app.local` ?
- Comment affichez-vous les logs des requêtes entrantes sur votre application ?
- Quand vous supprimez le pod que se passe-t-il ?

## 8 Secret dans Kubernetes

Une fois votre déploiement valide, modifier le `compute.yaml` afin de pointer la variable d'environnement vers un secret existant. Il faut créer ce manifest `secret.yaml` dans votre dossier

`vs_code`

**Astuce**

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: coder-password
type: Opaque
stringData:
  password: xxxx
```

Listing 4 – secret.yaml

Modifier la section `env` :

```yaml
valueFrom:
  secretKeyRef:
    name: nom
    key: nom-de-la-key
```

**Question**

Questions :

— Comment faire en sorte que ce secret ne soit en clair dans nos manifests (base64 n'est pas un algo de chiffrement...) ?

# 9 Pour les plus rapides

## 9.1 Déploiement d'une application PHP avec Redis

L'application Guestbook est un frontend PHP qui utilise redis pour stocker ses données. Redis est un système de gestion de base de données clé-valeur.

Avec les indications ci-dessous, créer votre application Guestbook dans un dossier `guestbook-php`

## 9.2 Création de la base de données Redis

Le fichier manifest, inclus ci-dessous, spécifie un contrôleur de déploiement qui exécute un replica unique du pod Redis.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
```

```
      labels:
        app: redis
        role: leader
        tier: backend
    spec:
      containers:
      - name: leader
        image: "docker.io/redis:6.0.5"
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

Listing 5 – redis-leader-deployment.yaml

1. Interroger la liste des Pods pour vérifier que le Pod Redis est en cours d'exécution

```
kubectl get pods
```

La réponse devrait être similaire à :

```
NAME                           READY   STATUS    RESTARTS   AGE
redis-leader-bv76de5-hvp0      1/1     Running   0          3s
```

2. Exécutez la commande suivante pour afficher les logs du pod leader Redis :

```
kubectl logs -f deployment/redis-leader
```

### 9.3 Création du service Redis leader

L'application Guestbook a besoin de communiquer avec le leader redis pour écrire ses données. Pour ce faire, il faut créer un service pour proxifier le trafic vers le pod Redis.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: leader
    tier: backend
```

Listing 6 – redis-leader-service.yaml

1. Interroger la liste des services pour vérifier que le service Redis est en cours d'exécution

```
kubectl get service
```

La réponse devrait être similaire à :

```
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes       ClusterIP   10.10.0.1       <none>        443/TCP    20m
redis-leader     ClusterIP   10.154.88.17    <none>        6379/TCP   4m
```

2. Exécutez la commande suivante pour afficher les logs du pod leader Redis :

```
kubectl logs -f deployment/redis-leader
```

**Note**

Ce manifest déploie un service nommé `redis-leader` avec des labels qui match les labels définis dans le `Deployment`, donc le Service route le trafic vers le pod Redis.

### 9.4 Création des redis followers

Comme le Redis Leader est un seul Pod, vous pouvez faire en sorte de le rendre hautement disponible et ainsi matcher les demandes de trafic réseau en ajoutant des redis followers, ou bien des replicas.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: redis-follower
5    labels:
6      app: redis
7      role: follower
8      tier: backend
9  spec:
10   replicas: 2
11   selector:
12     matchLabels:
13       app: redis
14   template:
15     metadata:
16       labels:
17         app: redis
18         role: follower
19         tier: backend
20     spec:
21       containers:
22       - name: follower
23         image: gcr.io/google_samples/gb-redis-follower:v2
24         resources:
25           requests:
26             cpu: 100m
27             memory: 100Mi
28         ports:
29         - containerPort: 6379
```

Listing 7 – redis-follower-deployment.yaml

1. Interroger la liste des Pods pour vérifier que les 2 Pod Redis followers sont en cours d'exécution

```
kubectl get pods
```

La réponse devrait être similaire à :

```
NAME                              READY   STATUS    RESTARTS   AGE
redis-leader-bv76de5-hvp0         1/1     Running   0          11m
redis-follower-dddfbdcc9-82sfr    1/1     Running   0          37s
redis-follower-dddfbdcc9-qrt5k    1/1     Running   0          38s
```

### 9.5 Création du service Redis followers

L'application Guestbook a besoin de communiquer avec les followers redis afin de lire les données. Pour ce faire il faut créer un autre service.

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: redis-follower
5    labels:
6      app: redis
7      role: follower
8      tier: backend
9  spec:
10   ports:
11   # the port that this service should serve on
12   - port: 6379
13   selector:
14     app: redis
15     role: follower
16     tier: backend
```

Listing 8 – redis-follower-service.yaml

1. Interroger la liste des services pour vérifier que le service Redis est en cours d'exécution

```
kubectl get service
```

La réponse devrait être similaire à :

```
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
kubernetes       ClusterIP   10.10.0.1        <none>        443/TCP     20m
redis-follower   ClusterIP   10.165.17.42     <none>        6379/TCP    9s
redis-leader     ClusterIP   10.154.88.17     <none>        6379/TCP    5m
```

### 9.6 Création de l'application Guestbook

Maintenant que le stockage Redis de votre application Guestbook est opérationnel, démarrez les serveurs web Guestbook. Comme les Redis followers, le frontend est déployé à l'aide d'un déploiement Kubernetes.

L'application Guestbook utilise un frontend PHP. Elle est configurée pour communiquer avec les services Redis follower ou leader, selon que la requête est une lecture ou une écriture. Le frontend expose une interface JSON.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: frontend
5  spec:
6    replicas: 3
7    selector:
```

```
    matchLabels:
      app: guestbook
      tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v5
        env:
        - name: GET_HOSTS_FROM
          value: "dns"
        resources:
          requests:
            cpu: 10m
            memory: 10Mi
        ports:
        - containerPort: 80
```

Listing 9 – frontend-deployment.yaml

1. Interroger la liste des services pour vérifier que les replicas frontend sont en cours d'exécution

```
kubectl get pods -l app=guestbook -l tier=frontend
```

La réponse devrait être similaire à :

```
NAME                        READY   STATUS    RESTARTS   AGE
frontend-85595f5bf9-5tqhb   1/1     Running   0          47s
frontend-85595f5bf9-qbzwm   1/1     Running   0          47s
frontend-85595f5bf9-zchwc   1/1     Running   0          47s
```

## 9.7 Création du service frontend

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  ports:
  # the port that this service should serve on
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

Listing 10 – frontend-service.yaml

### 9.8 Création de l'ingress

À vous de créer l'ingress pour accéder dans votre navigateur à l'application Guestbook
Si vous avez réussi, un cadeau vous attend ☻

## Merci de votre attention