TD02 - Kubernetes : Déploiement d'applications

Préparation au TP02

Charles SIEPEN
BUT Informatique - Semestre 5 - Ressource R5.09

2024/2025

Objectifs du TD

Ce TD a pour objectif de vous familiariser avec les concepts théoriques de Kubernetes nécessaires pour réussir le TP02.
Prérequis : Avoir terminé le TD01 sur les conteneurs et Docker.
Vous étudierez :

- La transition de Docker vers Kubernetes
- Les objets Kubernetes fondamentaux (Pods, Deployments, Services)
- Les commandes kubectl essentielles
- L'architecture d'une application sur Kubernetes
- La gestion du stockage avec les PersistentVolumeClaims
- L'exposition des applications avec Ingress

Rappel - TD01

Dans le TD01, vous avez appris à :

- Créer des images Docker avec un Dockerfile
- Déployer des conteneurs individuels
- Gérer le stockage avec des volumes Docker
- Configurer le réseau entre conteneurs

Kubernetes va vous permettre d'orchestrer et de gérer ces conteneurs à grande échelle !

1 De Docker à Kubernetes

1.1 Pourquoi Kubernetes ?

Dans le TD01, vous avez appris à déployer des conteneurs Docker individuellement. Cependant, imaginez que vous devez :

- Déployer une application composée de 10 microservices différents
- Assurer la haute disponibilité avec plusieurs replicas de chaque service
- Gérer automatiquement les redémarrages en cas de panne
- Scaler automatiquement selon la charge
- Mettre à jour vos applications sans interruption de service

C'est là que Kubernetes intervient !

1.2 Kubernetes vs Docker : Complémentarité

Important à comprendre

— **Docker** : Vous permet de créer et exécuter des conteneurs individuels

— **Kubernetes** : Orchestre et gère des centaines/milliers de conteneurs Docker

Kubernetes **n'est pas un remplacement** de Docker, mais un **orchestrateur** qui utilise Docker (ou d'autres runtimes de conteneurs) pour exécuter les conteneurs.

Analogie

Docker = Construire et conduire une voiture
Kubernetes = Gérer une flotte de taxis avec :

— Dispatch automatique des courses

— Remplacement des véhicules en panne

— Ajout de véhicules aux heures de pointe

— Maintenance et mises à jour coordonnées

1.3 Rappel : Architecture Docker (TD01)

Dans le TD01, vous avez vu l'architecture Docker :

— **Client Docker** : Interface en ligne de commande (`docker`)

— **Démon Docker** : Gère les conteneurs, images, réseaux, volumes

— **Registre** : Stocke et distribue les images

1.4 Architecture Kubernetes : Une extension du concept

Kubernetes reprend des concepts similaires mais à plus grande échelle :

Concept	**Docker (TD01)**	**Kubernetes (TD02)**
Conteneur	`docker run`	Pod (ensemble de conteneurs)
Gestion	Démon Docker	Control Plane + Kubelet
CLI	`docker`	`kubectl`
Réseau	Docker network	Service + Ingress
Stockage	Docker volume	PersistentVolume(Claim)
Déploiement	Dockerfile	Deployment manifest (YAML)

2 Introduction à kubectl

2.1 De `docker` à `kubectl`

Vous connaissez déjà les commandes Docker du TD01. Voici leur équivalent en Kubernetes :

Docker (TD01)	**Kubernetes (TD02)**
`docker ps`	`kubectl get pods`
`docker images`	`kubectl get deployments`
`docker run nginx`	`kubectl create deployment nginx -image=nginx`
`docker logs <container>`	`kubectl logs <pod>`
`docker exec -it <container> sh`	`kubectl exec -it <pod> - sh`
`docker rm <container>`	`kubectl delete pod <pod>`
`docker network ls`	`kubectl get services`
`docker volume ls`	`kubectl get pvc`

2.2 Les commandes de base

La commande `kubectl` est l'outil principal pour interagir avec un cluster Kubernetes. Elle utilise une syntaxe de type `kubectl <verbe> <objet>`, similaire à `docker <objet> <commande>` que vous avez vu dans le TD01.

Question 1

Associez chaque verbe kubectl à sa fonction :

a) `get`

b) `describe`

c) `apply`

d) `delete`

e) `logs`

Fonctions :

I. Afficher les détails complets d'une ressource

II. Supprimer une ressource

III. Appliquer une configuration depuis un fichier

IV. Lister les ressources

V. Consulter les journaux d'un conteneur

2.3 Les objets Kubernetes

Question 2

Complétez le tableau suivant avec la description de chaque objet Kubernetes :

Objet	**Description**
Pod	_____
Deployment	_____
Service	_____
Ingress	_____
PersistentVolumeClaim	_____
Secret	_____

2.4 Namespaces

Question 3

1. Qu'est-ce qu'un namespace dans Kubernetes ?
2. Donnez deux exemples de namespaces système standard.
3. Pourquoi ne devrait-on pas déployer ses applications dans le namespace `default` ?
4. Quelle est la différence entre une ressource *namespaced* et une ressource *cluster-scoped* ?
5. Donnez un exemple de ressource cluster-scoped.

3 Architecture d'une application sur Kubernetes

Kubernetes organise les applications selon trois composants principaux : Compute, Storage, et Networking.

3.1 Compute : Les Deployments

Voici un exemple de Deployment simplifié :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-app
  labels:
    app: mon-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mon-app
  template:
    metadata:
      labels:
        app: mon-app
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Question 4

En analysant le manifest ci-dessus :

1. Quelle version de l'API Kubernetes est utilisée ?
2. Combien de replicas (copies) du Pod seront créés ?
3. Quel est le rôle du champ `selector.matchLabels` ?
4. Sur quel port le conteneur écoute-t-il ?
5. Que se passerait-il si vous supprimiez manuellement un des Pods créés par ce Deployment ?

3.2 Storage : Les PersistentVolumeClaims

Rappel TD01 - Volumes Docker

Dans le TD01 (exercice 8), vous avez appris que :

- Les données dans un conteneur sont **éphémères** par défaut
- Les **volumes Docker** permettent de persister les données
- Un volume survit à la suppression du conteneur

Kubernetes utilise le même principe, mais avec plus de sophistication !

Docker Volume (TD01)	**Kubernetes PVC (TD02)**
Créé par le démon Docker	Provisionné dynamiquement
Lié à un hôte spécifique	Peut migrer entre nœuds
`docker volume create`	Manifest PersistentVolumeClaim
Montage : `-v nom:/path`	`volumeMounts` dans le Pod

Question 5

1. Expliquez la différence entre :
   - PersistentVolume (PV)
   - PersistentVolumeClaim (PVC)
   - StorageClass
2. Pourquoi utilise-t-on un PVC plutôt que de stocker directement les données dans le conteneur ? (Pensez à l'exercice 8 du TD01)
3. Que signifie le mode d'accès `ReadWriteOnce` ?
4. Citez deux autres modes d'accès possibles pour un volume.

3.3 Networking : Services et Ingress

Rappel TD01 - Réseau Docker

Dans le TD01 (section 9), vous avez appris que :

- Docker crée un réseau `bridge` par défaut
- Les conteneurs communiquent via leurs adresses IP
- Le mapping de ports expose les conteneurs (`-p 8080:80`)
- Les conteneurs sur le même réseau peuvent se découvrir

Kubernetes ajoute des abstractions puissantes avec les Services et Ingress !

Docker Network (TD01)	**Kubernetes (TD02)**
Réseau bridge	Service ClusterIP
Port mapping `-p 8080:80`	Service NodePort
Accès HTTP externe	Ingress
Découverte par nom DNS	Service DNS automatique

Question 6

1. Expliquez pourquoi on a besoin d'un Service pour accéder à des Pods.
2. Quels sont les trois principaux types de Services dans Kubernetes ?
3. Quelle est la différence entre un Service de type `ClusterIP` et `NodePort` ?
4. À quoi sert un objet Ingress ? (Comparez avec le port mapping Docker du TD01)
5. Qu'est-ce qu'un Ingress Controller ? Donnez un exemple.

3.4 Focus sur l'Ingress

L'Ingress est un objet crucial pour exposer vos applications au monde extérieur de manière professionnelle.

Rappel : Limites du port mapping Docker

Dans le TD01, pour exposer un conteneur, vous utilisiez :

```
docker run -p 8080:80 nginx
```

Limitations :

— Un seul conteneur par port de l'hôte
— Pas de routage intelligent (toujours vers le même conteneur)
— Pas de gestion HTTPS/TLS
— Pas de virtual hosting (plusieurs domaines)

Exemple d'Ingress pour une application web :

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mon-app-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: mon-app.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

Question 6 bis - Analyse d'Ingress

En analysant le manifest Ingress ci-dessus :

1. Quelle est la version de l'API utilisée pour l'Ingress ?
2. Quel hostname est configuré pour accéder à l'application ?
3. Vers quel Service le trafic sera-t-il routé ?
4. Que signifie `pathType: Prefix` ?
5. Comment pourriez-vous ajouter le support HTTPS à cet Ingress ?

Exemple avancé : Ingress avec plusieurs chemins

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: multi-path-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 3000
      - path: /admin
        pathType: Prefix
        backend:
          service:
            name: admin-service
            port:
              number: 8080
```

Question 6 ter - Routage

1. Un utilisateur accède à `http://myapp.example.com/api/users`. Vers quel Service sera-t-il routé ?
2. Que se passe-t-il si l'utilisateur accède à `http://myapp.example.com/home` ?
3. Proposez un cas d'usage réel pour ce type de configuration multi-chemins.

4 Les variables d'environnement et les Secrets

Rappel TD01 - Images et configuration

Dans le TD01 (exercice 6), vous avez déployé un service web Java dans un conteneur en utilisant :

- Une image de base (`eclipse-temurin:11-jre`)
- L'instruction `COPY` pour ajouter votre JAR
- L'instruction `CMD` ou `ENTRYPOINT` pour lancer l'application

Dans Kubernetes, le principe est similaire, mais la configuration est externalisée !

4.1 Variables d'environnement

Les variables d'environnement permettent de passer des configurations aux conteneurs, exactement comme dans Docker.

```
env:
- name: DATABASE_URL
  value: "postgres://db:5432/mydb"
- name: PASSWORD
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: password
```

Question 7

1. Quelle est la différence entre définir une valeur directement avec `value` et utiliser `valueFrom` ?
2. Dans quel cas devriez-vous utiliser un Secret plutôt qu'une valeur en clair ?
3. Le Secret Kubernetes utilise l'encodage base64. Est-ce un chiffrement sécurisé ? Pourquoi ?
4. Proposez une solution pour améliorer la sécurité des Secrets dans Kubernetes.

5 Volumes et montages

```
volumeMounts:
- mountPath: /data
  name: storage-volume
volumes:
- name: storage-volume
  persistentVolumeClaim:
    claimName: my-pvc
```

Question 8

1. Expliquez le rôle de la section `volumes`.
2. Expliquez le rôle de la section `volumeMounts`.
3. Que se passe-t-il si le PVC `my-pvc` n'existe pas ?
4. Pourquoi les données stockées dans `/data` persistent-elles alors que celles stockées ailleurs dans le conteneur sont perdues ?

6 Labels et Selectors

Les labels sont des paires clé-valeur attachées aux objets Kubernetes. Les selectors permettent de filtrer les objets par leurs labels.

Question 9

Considérez le Deployment et le Service suivants :

Deployment :

```yaml
metadata:
  labels:
    app: frontend
    tier: web
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
        tier: web
```

Service :

```yaml
spec:
  selector:
    app: frontend
  ports:
  - port: 80
```

Questions :

1. Le Service pourra-t-il router le trafic vers les Pods créés par ce Deployment ? Pourquoi ?
2. Que se passerait-il si le Service utilisait le selector `tier: web` au lieu de `app: frontend` ?
3. Proposez une commande kubectl pour lister uniquement les Pods ayant le label `app=frontend`.

7 Exercice de synthèse

Exercice pratique sur papier

Vous devez déployer une application web simple composée de :

- Un frontend Nginx (3 replicas)
- Un backend Node.js (2 replicas)
- Une base de données PostgreSQL (1 replica avec stockage persistant)

Travail à faire :

1. Dessinez un schéma représentant l'architecture de cette application sur Kubernetes.
2. Listez tous les objets Kubernetes nécessaires (types et noms).
3. Pour chaque objet, indiquez :
   - Son type (Deployment, Service, PVC, etc.)
   - Son rôle
   - Avec quels autres objets il interagit
4. Expliquez le flux de données depuis un utilisateur externe jusqu'à la base de données.

8 Questions de réflexion pour le TP

Préparation au TP02

Avant le TP, réfléchissez aux questions suivantes :

1. Comment vérifier que tous les Pods d'un Deployment sont en état `Running` ?
2. Quelle commande utiliseriez-vous pour consulter les logs d'un Pod qui redémarre en boucle ?
3. Comment accéder à une application depuis votre navigateur si elle est exposée via un Ingress avec le hostname `mon-app.local` ?
4. Si vous supprimez un Pod, que devient son PersistentVolume ?
5. Comment modifier le nombre de replicas d'un Deployment déjà déployé ?
6. Quelle est la différence entre `kubectl apply` et `kubectl create` ?

9 Commandes kubectl à mémoriser

Antisèche kubectl

```
# Lister les ressources
kubectl get nodes
kubectl get pods
kubectl get deployments
kubectl get services
kubectl get pvc
kubectl get namespaces

# Obtenir des details
kubectl describe pod <nom-pod>
kubectl describe service <nom-service>

# Appliquer des manifests
kubectl apply -f fichier.yaml
kubectl apply -f dossier/

# Consulter les logs
kubectl logs <nom-pod>
kubectl logs -f <nom-pod>  # mode suivi

# Supprimer des ressources
kubectl delete pod <nom-pod>
kubectl delete -f fichier.yaml

# Executer une commande dans un pod
kubectl exec -it <nom-pod> -- /bin/bash

# Afficher en YAML
kubectl get service <nom> -o yaml

# Filtrer par labels
kubectl get pods -l app=frontend
```

10 Pour aller plus loin

Ressources complémentaires

- Documentation officielle Kubernetes : `https://kubernetes.io/fr/docs/home/`
- Interactive tutorial : `https://kubernetes.io/docs/tutorials/`
- Kubernetes CheatSheet : `https://kubernetes.io/docs/reference/kubectl/cheatsheet/`
- Play with Kubernetes : `https://labs.play-with-k8s.com/`

Bon courage pour le TP !