TD Kubernetes — Fiche 1

Prise en main, déploiement et migration depuis Docker Compose

Objectifs

- Démarrer un cluster Kubernetes local avec Docker Desktop
- Comprendre les concepts fondamentaux : Pod, Deployment, Service
- Écrire et appliquer des manifestes YAML déclaratifs
- Gérer le cycle de vie des pods (scaling, rolling update, self-healing)
- Porter une architecture Docker Compose vers Kubernetes

Prérequis : Docker Desktop installé avec Kubernetes activé, bonnes bases Docker (images, Dockerfile, Docker Compose).

1. Prise en main de Kubernetes

a. Démarrer le cluster

Docker Desktop embarque un cluster Kubernetes qu'il suffit d'activer. Contrairement à un cluster de production qui s'étend sur plusieurs machines, ce cluster tourne entièrement en local — c'est parfait pour apprendre.

Démarrage du cluster K8s

1. Ouvrez Docker Desktop, allez dans Settings, cochez Enable Kubernetes puis cliquez sur Apply & Restart.
2. Une fois le cluster lancé, son statut apparaît dans le bandeau bas de Docker Desktop.
3. Ouvrez un terminal et exécutez les commandes suivantes pour vérifier l'installation :

```
kubectl version
kubectl cluster-info
kubectl get nodes
```

kubectl est l'équivalent de la commande docker, mais pour piloter un cluster Kubernetes. Kubernetes (abrégé k8s) orchestre des conteneurs sur plusieurs machines, en gérant le scaling, la répartition de charge et l'auto-réparation.

b. Préparer les images Docker

Avant de déployer quoi que ce soit sur K8s, il faut des images Docker. Nous allons en construire 3 versions successives d'une API Python.

Construction de 3 versions d'une image

1. Créez un fichier app.py contenant une API FastAPI avec au moins deux routes : une route / et une route /version.

Exemple de contenu :

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.get("/version")
async def version():
    return {"version": "0.1.0"}
```

2. Rédigez le Dockerfile nécessaire et construisez l'image en la taguant `api:0.1.0`. Testez-la avec `docker run --rm -p 8000:8000 api:0.1.0`.
3. Modifiez le code (ajoutez une route, changez le numéro de version) et produisez deux nouvelles images : `api:0.2.0` et `api:0.3.0`.

c. Déployer un Pod

Le Pod est l'unité de base de Kubernetes : il encapsule un ou plusieurs conteneurs qui partagent le même réseau et le même stockage local. C'est l'équivalent K8s d'un conteneur Docker, en un peu plus riche.

Comme sur le schéma ci-dessous, un Pod est toujours décrit dans un fichier YAML que l'on soumet à Kubernetes. K8s se charge ensuite de faire converger l'état réel vers l'état déclaré.

Déploiement d'un premier Pod

1. Créez un dossier k8s/, puis un fichier pod.yaml à l'intérieur.
2. Renseignez-y le manifeste suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: api-pod
  labels:
    app: api
    version: 0.1.0
spec:
  containers:
  - name: api
    image: api:0.1.0
    ports:
    - containerPort: 8000
```

3. Soumettez ce fichier au cluster avec `kubectl apply -f pod.yaml`, puis listez les pods actifs.

4. Inspectez le pod en détail avec `kubectl get pod api-pod -o yaml`. Repérez les champs spec (état désiré) et status (état réel).
5. Ouvrez un shell dans le pod avec `kubectl exec -it api-pod -- /bin/bash`, puis testez l'API en interne avec curl.
6. Trouvez et exécutez les commandes pour : afficher les logs du pod, faire un port-forward sur le port 8000 et ouvrir l'API dans le navigateur.
7. Supprimez le pod avec `kubectl delete pod api-pod`. Le pod réapparaît-il tout seul ? Supprimez-le proprement via le fichier YAML.

Plusieurs pods en parallèle

1. Éditez pod.yaml pour déclarer 3 pods (un par version d'image) dans le même fichier, séparés par ---.
2. Appliquez le fichier, listez les pods, puis supprimez manuellement le deuxième. Se recrée-t-il ?
3. Relancez `kubectl apply -f pod.yaml`. Que se passe-t-il pour les pods déjà existants ? Pour le pod supprimé ?
4. Nettoyez tout avec `kubectl delete -f pod.yaml`.

Exercice — Namespaces

Les Namespaces permettent de partitionner logiquement les ressources d'un cluster. Ils servent typiquement à isoler les environnements (dev, qualif, prod) au sein du même cluster.

- Renseignez-vous sur le concept de Namespace Kubernetes.
- Créez un fichier namespace.yaml déclarant trois namespaces : dev, qualif et prod.
- Déployez api:0.1.0 dans prod, api:0.2.0 dans qualif et api:0.3.0 dans dev. Chaque pod doit s'appeler api-pod et porter un label environment correspondant.
- Basculez entre les namespaces et vérifiez que chaque pod n'est visible que dans le bon contexte.
- Filtrez tous les pods portant le label environment=dev.
- Nettoyez avec kubectl delete -f pod.yaml puis kubectl delete -f namespace.yaml.

d. Scaler avec un Deployment

Un Pod seul ne se répare pas et ne se réplique pas. Le Deployment est la ressource K8s qui gère cela : il maintient un nombre souhaité de répliques, remplace automatiquement un pod tombé et permet les mises à jour progressives.

10 répliques avec auto-réparation

1. Créez un fichier deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-deploy
spec:
  replicas: 10
  selector:
```

```
    matchLabels:
      app: api
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
      - name: api-pod
        image: api:0.1.0
        ports:
        - containerPort: 8000
```

2. Appliquez ce fichier et vérifiez que 10 pods sont bien en cours d'exécution.
3. Dans un second terminal, lancez `kubectl get deploy api-deploy --watch` pour observer le cluster en temps réel.
4. Supprimez un pod manuellement. Que se passe-t-il ? Le Deployment maintient-il bien 10 répliques ?
5. Modifiez replicas dans le YAML et relancez kubectl apply. Observez la montée/descente en charge.

e. Exposer les pods avec un Service

Les pods ont des adresses IP éphémères qui changent à chaque redémarrage. Un Service fournit un point d'accès stable (nom DNS + IP fixe) qui redirige le trafic vers un ensemble de pods via leurs labels.

Création d'un Service NodePort

1. Vérifiez que votre deployment de 10 pods est toujours actif.
2. Créez un fichier service.yaml :

```
apiVersion: v1
kind: Service
metadata:
  name: api-svc
  labels:
    app: api
spec:
  type: NodePort
  ports:
  - port: 8000
    nodePort: 30001
    protocol: TCP
  selector:
    app: api
```

3. Appliquez le service et ouvrez `http://localhost:30001/docs` dans le navigateur. Chaque requête est redirigée vers un pod différent.

> *La combinaison Deployment → Répliques de Pods → Service est le socle minimal pour déployer une application sur Kubernetes.*

2. De Docker Compose vers Kubernetes

Dans le TD Docker Compose, vous avez déployé un projet fullstack composé d'un frontend client et d'un backend de prédiction Iris. L'objectif ici est de porter cette même architecture sur Kubernetes, en remplaçant docker-compose.yml par des manifestes YAML K8s.

Docker Compose	Kubernetes
`Service`	`Deployment + Service`
`scale: N`	`replicas: N`
`networks`	`Labels + selectors`
`volumes`	`PersistentVolumeClaim`
`environment`	`ConfigMap / Secret`

Exercice — Application fullstack Iris sur Kubernetes

Vous allez reconstruire l'intégralité de l'architecture de prédiction Iris, cette fois sur K8s.

1. Reconstruisez l'image `mlops-client:latest` correspondant au frontend.
2. Entraînez 3 modèles de prédiction Iris différents et empaquetez-les dans 3 images Docker distinctes : `mlops-server:0.1.0`, `mlops-server:0.2.0` et `mlops-server:0.3.0`. Chaque image doit exposer un endpoint /version indiquant sa version.
3. Créez un Deployment et un Service pour le frontend (mlops-client:latest).
4. Créez un Deployment avec 3 répliques et un Service pour le backend (mlops-server:0.1.0).
5. Connectez le frontend au backend en utilisant le nom du Service comme URL dans le code Python. Par exemple, si votre service backend s'appelle `mlops-api-service`, l'URL à utiliser sera `http://mlops-api-service:8000`. Kubernetes résout ce nom DNS automatiquement.
6. Vérifiez que le frontend est accessible depuis le navigateur et que les prédictions fonctionnent.

> *Conservez ce déploiement Iris actif : il servira de base pour la Fiche 2, qui portera sur les stratégies de mise à jour (Rolling Update, Blue/Green, Canary).*

Récapitulatif des commandes essentielles

`kubectl apply -f fichier.yaml`	*Créer ou mettre à jour une ressource*
`kubectl delete -f fichier.yaml`	*Supprimer les ressources déclarées*
`kubectl get pods`	*Lister les pods du namespace courant*
`kubectl get pods -l app=api`	*Filtrer les pods par label*
`kubectl describe pod <nom>`	*Détails complets d'un pod*
`kubectl logs <pod>`	*Afficher les logs d'un pod*
`kubectl exec -it <pod> -- /bin/bash`	*Ouvrir un shell dans un pod*
`kubectl port-forward <pod> 8000:8000`	*Exposer un pod en local*
`kubectl get deploy <nom> --watch`	*Observer un deployment en temps réel*
`kubectl rollout undo deployment/<nom>`	*Annuler la dernière mise à jour*