

Kubernetes

Préparation de l'infrastructure (Job 01)

- Objectifs :
 - Créer 3 VM Debian (sans GUI)
 - Installer K3s sur chaque VM
- Noms des VMs :
 - kubes-01.local
 - kubes-02.local
 - kubes-03.local

Mise à jour système :

```
sudo apt update && sudo apt upgrade -y
```

```
sudo apt install curl vim git -y
```

- Installation de K3s :

```
curl -sfL https://get.k3s.io | sh -
```
- Vérification :

```
sudo kubectl get nodes
```

Déploiement d'applications simples (Job 02)

Déployer sur chaque VM :

- nginx (serveur web léger)
- apache (autre serveur web très utilisé)
- mysql ou mariadb (base de données relationnelle)

On va utiliser kubectl et créer un fichier .yaml pour déployer chaque application.

Suivre ces étapes sur chaque VM indépendamment (kubes-01, kubes-02, kubes-03).

1. Créer un fichier de déploiement nginx

Crée un fichier nginx-deployment.yaml :

```
# NGINX DEPLOYMENT
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx
```

```
spec:
```

```
  replicas: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: nginx
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: nginx
```

```
    spec:
```

```
      containers:
```

```
        - name: nginx
```

```
image: nginx:latest  
ports:  
- containerPort: 80
```

```
---
```

NGINX SERVICE

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
name: nginx-service
```

```
spec:
```

```
type: NodePort
```

```
selector:
```

```
app: nginx
```

```
ports:
```

```
- port: 80
```

```
targetPort: 80
```

```
nodePort: 30080
```

```
---
```

APACHE DEPLOYMENT

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: apache
```

```
spec:
```

```
replicas: 1
```

```
selector:
```

```
matchLabels:  
    app: apache  
  
template:  
    metadata:  
        labels:  
            app: apache  
  
spec:  
    containers:  
        - name: apache  
          image: httpd:latest  
    ports:  
        - containerPort: 80  
  
---  
  
# APACHE SERVICE  
  
apiVersion: v1  
  
kind: Service  
  
metadata:  
    name: apache-service  
  
spec:  
    type: NodePort  
  
    selector:  
        app: apache  
  
    ports:  
        - port: 80  
          targetPort: 80  
          nodePort: 30081
```

MARIADB DEPLOYMENT

apiVersion: apps/v1

kind: Deployment

metadata:

name: mariadb

spec:

replicas: 1

selector:

matchLabels:

app: mariadb

template:

metadata:

labels:

app: mariadb

spec:

containers:

- name: mariadb

image: mariadb:latest

env:

- name: MYSQL_ROOT_PASSWORD

value: "password"

ports:

- containerPort: 3306

MARIADB SERVICE

```
apiVersion: v1
kind: Service
metadata:
  name: mariadb-service
spec:
  type: NodePort
  selector:
    app: mariadb
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30306
```

Déployer avec :

```
sudo kubectl apply -f all-apps.yaml
```

```
dome@kubes-01:~$ sudo kubectl apply -f all-apps.yaml
deployment.apps/nginx unchanged
service/nginx-service created
deployment.apps/apache created
service/apache-service created
deployment.apps/mariadb created
service/mariadb-service created
dome@kubes-01:~$ |
```

Vérifier

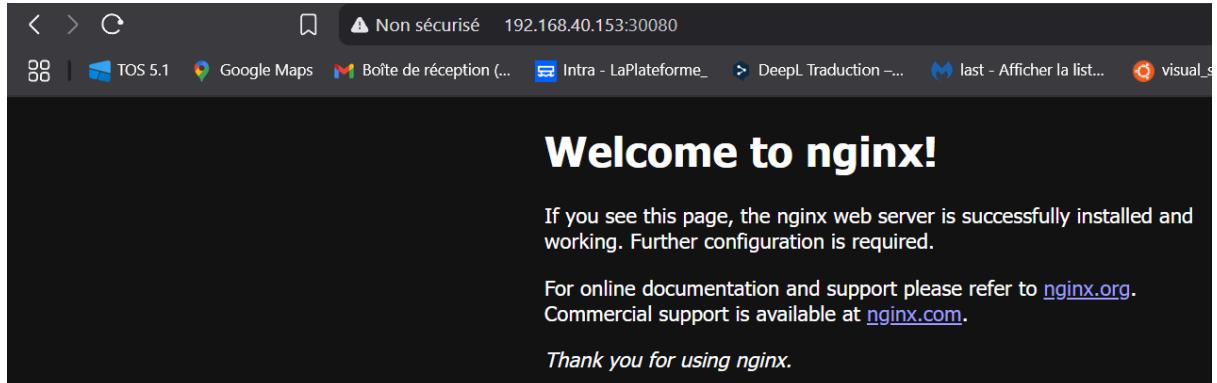
```
kubectl get pods
```

```
kubectl get deployments
```

```
kubectl describe pod <nom_du_pod>
```

l'utilisateur pouvez maintenant accéder à son nginx via l'IP de la VM sur le port 30080

Service avec nodePort: 30080



l'utilisateur pouvez maintenant accéder à son Apache via l'IP de la VM sur le port 30081

Service avec nodePort: 30081

Comment tester si MariaDB fonctionne correctement ?

l'utilisateur dois utiliser un client MySQL/MariaDB, pas un navigateur.

1. Depuis la même VM (kubes-01)

`sudo apt install mariadb-client -y`

Puis essaye une connexion :

`mysql -h 127.0.0.1 -P 30306 -u root -p`

Quand il demande le mot de passe, entre :

`password`

(si l'utilisateur as mis value: "password" dans son YAML)

```
dome@kubes-01:~$ mysql -h 127.0.0.1 -P 30306 -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 24
Server version: 11.8.2-MariaDB-ubuntu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> exit
Bye
dome@kubes-01:~$ |
```

Création du cluster K3S (Job 03)

Objectifs:

- Apprendre à transformer un ensemble de machines en un cluster Kubernetes fonctionnel, avec un nœud maître (master) et plusieurs nœuds travailleurs (workers).
- Vérifier que les applications que l'utilisateur avais installées précédemment sont toujours accessibles après la mise en cluster.

1. Sur le nœud master (kubes-01.local)

```
curl -sfL https://get.k3s.io | sh -
```

- Ce script télécharge et installe K3s.
- À la fin, un message indique que le service k3s tourne.

Vérifier que tout va bien :

```
sudo kubectl get nodes
```

2. Récupérer le token d'accès pour que les workers puissent rejoindre le cluster :

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Copie ce token, sur les machines kubes-02.local et kubes-03.local.

3. Sur chaque nœud worker (kubes-02.local et kubes-03.local)

Remplacer IP_DU_MASTER par l'adresse IP du master (kubes-01.local) et TOKEN par le token copié.

```
curl -sfL https://get.k3s.io | K3S_URL=https://IP_DU_MASTER:6443  
K3S_TOKEN="TOKEN" sh -
```

4. Vérifier sur le master que les 3 nœuds sont bien connectés :

```
sudo kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
kubes-01.local	Ready	control-plane,master	76m	v1.32.6+k3s1
kubes-02.local	Ready	<none>	110s	v1.32.6+k3s1
kubes-03.local	Ready	<none>	9s	v1.32.6+k3s1

Voir les 3 machines listées avec le statut Ready.

Vérification des applications après mise en cluster

Comme on avait déjà installé nginx, apache, mysql/mariadb dans Job 02, vérifier si elles tournent encore :

```
sudo kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	apache-5fd955856f-fpdp4	1/1	Running	0	39m
default	mariadb-6d69b65547-jsc95	1/1	Running	0	39m
default	nginx-96b9d695-cx6cr	1/1	Running	0	47m
kube-system	coredns-5688667fd4-2vzd8	1/1	Running	0	77m
kube-system	helm-install-traefik-crd-7hl2z	0/1	Completed	0	77m
kube-system	helm-install-traefik-drfd7	0/1	Completed	2	77m
kube-system	local-path-provisioner-774c6665dc-bx5dm	1/1	Running	0	77m
kube-system	metrics-server-6f4c6675d5-jzmnb	1/1	Running	0	77m
kube-system	svclb-traefik-4d93dc43-bnq62	2/2	Running	0	79s
kube-system	svclb-traefik-4d93dc43-jz4fl	2/2	Running	0	3m1s
kube-system	svclb-traefik-4d93dc43-pr8tz	2/2	Running	0	76m
kube-system	traefik-c98fdf6fb-hjjlp	1/1	Running	0	76m

Si l'utilisateur ne vois rien : c'est normal ! Les applications que l'utilisateur avais déployées n'étaient pas gérées par Kubernetes mais par docker ou containerd localement.

```
dome@kubes-02:~$ sudo kubectl get pods -A
The connection to the server 127.0.0.1:6443 was refused - did you specify the right host or port?
dome@kubes-02:~$ |
```

```
dome@kubes-03:~$ sudo kubectl get pods -A
The connection to the server 127.0.0.1:6443 was refused - did you specify the right host or port?
dome@kubes-03:~$ |
```

Solution pour utiliser kubectl sur les workers :

Par défaut, la commande kubectl ne fonctionne pas sur les nœuds workers car ils n'hébergent pas le serveur Kubernetes (API).

Si besoin, on peut copier le fichier de configuration /etc/rancher/k3s/k3s.yaml du master et l'adapter (changer 127.0.0.1 en IP du master)

Copier la configuration kubeconfig du master vers les workers.

1. Depuis le master :

```
sudo cat /etc/rancher/k3s/k3s.yaml
```

Copier tout le contenu du fichier.

2. Sur un worker (par exemple kubes-02) passer en mode root:

- Crée le dossier de config :

```
mkdir -p ~/.kube
```

- Colle le contenu dans un fichier :

```
nano ~/.kube/config
```

→ Colle ici ce que l'utilisateur as copié du master.

3. Remplace l'adresse 127.0.0.1 par l'IP du master :

Exemple :

```
server: https://127.0.0.1:6443
```

⬇ devient :

```
server: https://192.168.40.153:6443
```

4. Teste :

```
kubectl get nodes
```

Faut-il configurer kubectl sur les workers ?

Non, ce n'est pas obligatoire pour le fonctionnement du cluster ou la suite du projet.

Mais cela peut être utile si on souhaite exécuter kubectl depuis un nœud worker (par confort ou supervision).

Dans un usage pro, on utilise souvent kubectl uniquement depuis le master, ou depuis un poste admin avec un accès réseau sécurisé.

Haute Disponibilité (Job 04)

Objectifs:

- Supprimer toutes les applications déployées jusque-là.
- Les réinstaller avec de la haute disponibilité (HA) → donc réplicas = plusieurs copies.
- Tester la HA → en éteignant un nœud worker pour vérifier que les applis sont reprogrammées automatiquement ailleurs.

Concepts à comprendre avant de commencer

► Qu'est-ce que la haute disponibilité (HA) ?

C'est le fait d'assurer que les applications restent disponibles même si une partie du système (ici un worker) tombe en panne.

► Réplica (replicaSet ou replicas dans un deployment)

Un replica est une copie d'un pod (son application). Plus l'utilisateur a de replicas, plus l'utilisateur a de copies prêtes à prendre le relais.

1. Supprimer les applications existantes depuis le master :

kubectl delete deployment nginx apache mariadb

Vérifie aussi s'il reste des pods :

kubectl get pods

```
dome@kubes-01:~$ sudo kubectl delete deployment nginx apache mariadb
[sudo] Mot de passe de dome :
deployment.apps "nginx" deleted
deployment.apps "apache" deleted
deployment.apps "mariadb" deleted
dome@kubes-01:~$ |
```

2. Réinstaller les applications avec des replicas

Exemple : fichier apps-ha.yaml

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

```
kind: Deployment
metadata:
  name: mariadb
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      containers:
        - name: mariadb
          image: mariadb:latest
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "password"
          ports:
            - containerPort: 3306
```

Déploiement

```
kubectl apply -f apps-ha.yaml
```

```
dome@kubes-01:~$ sudo kubectl apply -f apps-ha.yaml
deployment.apps/nginx created
deployment.apps/apache created
deployment.apps/mariadb created
dome@kubes-01:~$ |
```

Faut-il supprimer les applications sur les trois machines ?

Non, pas manuellement sur chaque machine.

L'utilisateur est maintenant dans un cluster K3s avec 1 master et 2 workers (Job 03 terminé).
Donc toutes les commandes kubectl s'exécutent depuis le master (ou n'importe quel noeud avec kubectl configuré).

Exemple d'un Service pour accéder à NGINX (NodePort)

Types de services

1. ClusterIP (par défaut)

- Accès interne au cluster.
- Utilisé pour les communications entre les applications (ex: frontend ↔ backend).
- L'utilisateur ne peut pas y accéder depuis son PC ou depuis l'extérieur.

2. NodePort

- Ouvre un port sur chaque nœud (worker ou master).
- L'utilisateur peut y accéder depuis l'extérieur :
`http://<IP_NODE>:<NodePort>`

3. LoadBalancer (pour les clouds uniquement)

Ignore pour l'instant, sauf si l'utilisateur est dans un cloud (AWS, GCP...).

Modifier le fichier `apps-ha.yaml` :

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - protocol: TCP
      port: 80      # le port dans le cluster
      targetPort: 80 # le port du conteneur nginx
      nodePort: 30080 # le port exposé sur les machines du cluster
```

Sauvegarde ce code dans un fichier apps-ha.yaml

Depuis le master :

```
kubectl apply -f apps-ha.yaml
```

1. Vérifie les pods :

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

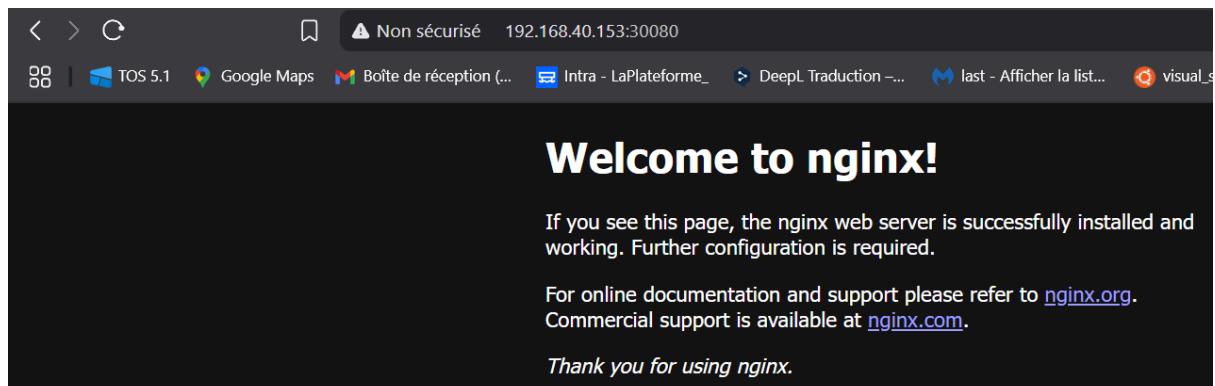
2. Accède aux applis :

- http://IP_NODE:30080 → nginx
- http://IP_NODE:30081 → apache
- Connexion MariaDB sur IP_NODE:30306

Résultat :

l'utilisateur peut accéder à son Nginx sur :

```
http://<IP_de_ton_node>:30080
```



Tester la HA : éteins un worker et vérifie si les pods sont redéployés.

```

dome@kubes-01:~$ sudo kubectl apply -f apps-ha.yaml
deployment.apps/nginx unchanged
service/nginx-service unchanged
deployment.apps/apache unchanged
service/apache-service unchanged
deployment.apps/mariadb unchanged
service/mariadb-service unchanged
dome@kubes-01:~$ sudo kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE      NOMINATED NODE   READINESS G
ATES
apache-5fd955856f-8gdpr  1/1    Running   0          13m    10.42.0.13  kubes-01.local  <none>        <none>
apache-5fd955856f-dfcxj  1/1    Running   0          13m    10.42.1.2  kubes-02.local  <none>        <none>
apache-5fd955856f-smkhj  1/1    Running   0          13m    10.42.2.3  kubes-03.local  <none>        <none>
mariadb-6d69b65547-4tbrp 1/1    Running   0          13m    10.42.1.3  kubes-02.local  <none>        <none>
mariadb-6d69b65547-ccmf 1/1    Running   0          13m    10.42.2.4  kubes-03.local  <none>        <none>
mariadb-6d69b65547-jdxlh 1/1    Running   0          13m    10.42.0.12  kubes-01.local  <none>        <none>
nginx-96b9d695-cdf9z    1/1    Running   0          13m    10.42.0.14  kubes-01.local  <none>        <none>
nginx-96b9d695-fx4zg    1/1    Running   0          13m    10.42.1.4  kubes-02.local  <none>        <none>
nginx-96b9d695-wtgbv    1/1    Running   0          13m    10.42.2.2  kubes-03.local  <none>        <none>
dome@kubes-01:~$ |

```

Stockage persistant (Job 05)

Concepts de base à comprendre

- Pod : Le plus petit objet exécutable dans Kubernetes (ex. un conteneur).
- Volume : Espace de stockage qu'un pod peut utiliser.
- PersistentVolume (PV) : Espace de stockage provisionné dans le cluster.
- PersistentVolumeClaim (PVC) : Demande d'un pod pour un volume spécifique.

Étapes pour mettre en place le stockage persistant

On va faire deux déploiements :

- Un pour nginx
- Un pour mariadb

NGINX – Exemple avec volume

1. Créer un volume local sur un nœud

Sur chaque VM, crée un dossier partagé :

```

sudo mkdir -p /data/nginx
sudo chmod 777 /data/nginx

```

l'utilisateur peux adapter /data/nginx selon sa structure.

2. Créer un PersistentVolume (PV), sur la vm Master.

Crée un fichier nginx-pv.yaml :

yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/nginx"
```

3. Créer un PersistentVolumeClaim (PVC)

Crée un fichier nginx-pvc.yaml :

yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

4. Créer le déploiement nginx

Crée nginx-deployment.yaml :

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
```

```

selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: nginx-storage
  volumes:
    - name: nginx-storage
      persistentVolumeClaim:
        claimName: nginx-pvc

```

Mariadb – Exemple avec volume

1. Créer un volume sur chaque VM :

```

sudo mkdir -p /data/mariadb
sudo chmod 777 /data/mariadb

```

2. Créer un PersistentVolume (PV), sur la vm Master.

Créer mariadb-pv.yaml :

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: mariadb-pv
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/mariadb"

```

3. Créer mariadb-pvc.yaml :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

4. Créer mariadb-deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      containers:
        - name: mariadb
          image: mariadb
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: rootpass
          volumeMounts:
            - name: mariadb-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mariadb-storage
          persistentVolumeClaim:
            claimName: mariadb-pvc
```

```
dome@kubes-01:~$ sudo nano mariadb-pv.yaml
dome@kubes-01:~$ sudo nano mariadb-pvc.yaml
dome@kubes-01:~$ sudo nano mariadb-deployment.yaml
dome@kubes-01:~$ sudo nano nginx-pv.yaml
dome@kubes-01:~$ sudo nano nginx-pvc.yaml
dome@kubes-01:~$ sudo nano nginx-deployment.yaml
dome@kubes-01:~$ ls
all-apps.yaml  apps-ha.yaml  mariadb-deployment.yaml  mariadb-pvc.yaml  nginx-deployment.yaml  nginx-pvc.yaml  nginx-pv.yaml
```

Commandes à exécuter

Voici la séquence à suivre :

```
kubectl apply -f nginx-pv.yaml
kubectl apply -f nginx-pvc.yaml
kubectl apply -f nginx-deployment.yaml
```

```
kubectl apply -f mariadb-pv.yaml
kubectl apply -f mariadb-pvc.yaml
kubectl apply -f mariadb-deployment.yaml
```

```
dome@kubes-01:~$ sudo kubectl apply -f nginx-pv.yaml
persistentvolume/nginx-pv created
dome@kubes-01:~$ sudo kubectl apply -f nginx-pvc.yaml
persistentvolumeclaim/nginx-pvc created
dome@kubes-01:~$ sudo kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx configured
dome@kubes-01:~$ sudo kubectl apply -f mariadb-pv.yaml
persistentvolume/mariadb-pv created
dome@kubes-01:~$ sudo kubectl apply -f mariadb-pvc.yaml
persistentvolumeclaim/mariadb-pvc created
dome@kubes-01:~$ sudo kubectl apply -f mariadb-deployment.yaml
deployment.apps/mariadb configured
dome@kubes-01:~$
```

Vérification

```
kubectl get pv
kubectl get pvc
kubectl get pods
```

L'utilisateur peut même redémarrer un pod et vérifier que les fichiers ne sont pas perdus.

ConfigMaps : Gérer la configuration des applications (Job 06)

Qu'est-ce qu'un ConfigMap ?

Un ConfigMap permet de séparer la configuration de l'application du code. C'est un objet Kubernetes utilisé pour injecter des variables d'environnement, des fichiers de configuration, ou des arguments de ligne de commande dans ses pods.

Pourquoi l'utiliser ?

- Modifier la configuration sans reconstruire l'image du conteneur.
- Réutilisable entre plusieurs applications.
- Lecture plus claire et meilleure maintenance.

Exemple de création :

1. Créer un fichier nginx-config.yaml :

```
yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  index.html: |
    <h1>Bonjour depuis ConfigMap !</h1>
```

2. Appliquer :

```
kubectl apply -f nginx-config.yaml
```

```
dome@kubes-01:~$ sudo kubectl apply -f nginx-config.yaml
configmap/nginx-config created
dome@kubes-01:~$ |
```

3. Monter dans un pod :

Dans le Deployment de Nginx :

```
yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
      volumeMounts:
        - name: nginx-html
          mountPath: /usr/share/nginx/html/index.html
          subPath: index.html
  volumes:
```

```
- name: nginx-html  
  configMap:  
    name: nginx-config
```

Test : Utilisation d'un ConfigMap dans un déploiement Nginx

Objectif :

Vérifier que la page HTML personnalisée définie dans le ConfigMap est bien utilisée par le serveur Nginx déployé dans Kubernetes.

Étapes du test :

1. Vérifie que le ConfigMap existe :

```
kubectl get configmap nginx-config
```

Résultat attendu :

```
dome@kubes-01:~$ sudo kubectl get configmap nginx-config  
NAME        DATA   AGE  
nginx-config 1      10h  
dome@kubes-01:~$ |
```

2. Vérifie que le déploiement Nginx est en place :

```
kubectl get deployments
```

l'utilisateur doit voir une ligne comme :

```
dome@kubes-01:~$ sudo kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
apache        3/3     3            3           3d14h
mariadb       1/1     1            1           3d14h
nginx         1/1     1            1           3d14h
nginx-deploy  1/1     1            1           12m
dome@kubes-01:~$ |
```

Et que le pod tourne :

```
kubectl get pods -l app=nginx
```

```
dome@kubes-01:~$ sudo kubectl get pods -l app=nginx
NAME          READY   STATUS    RESTARTS   AGE
nginx-5b484d9bfb-cbpn6   1/1     Running   1 (22m ago)   11h
nginx-deploy-566cc56d7b-c2dm4  1/1     Running   0           13m
dome@kubes-01:~$ |
```

3. Accède au pod Nginx et vérifie le fichier HTML :

```
sudo kubectl exec -it $(sudo kubectl get pods -l app=nginx -o
jsonpath=".items[1].metadata.name") -- cat /usr/share/nginx/html/index.htm
```

Résultat attendu :

- ✓ Cela prouve que Nginx utilise bien la page HTML venant du ConfigMap.

Secrets : Gérer des données sensibles (Job 07)

Qu'est-ce qu'un Secret ?

Un Secret est similaire à un ConfigMap, mais conçu pour des données sensibles (mots de passe, clés API, etc.). Les données sont encodées en base64.

À savoir :

- Ne remplace pas un vrai coffre-fort sécurisé (mais mieux que ConfigMap pour les infos sensibles).
- Kubernetes peut monter un Secret comme variable d'environnement ou fichier.

Objectif :

l'utilisateur vas protéger un mot de passe (par exemple celui de MariaDB) en l'injectant dans un conteneur, mais sans l'écrire directement dans le fichier YAML.

C'est ce qu'on appelle un Secret dans Kubernetes.

Test : Pour MariaDB

Créer un secret :

```
kubectl create secret generic mariadb-secret \
--from-literal=MARIADB_ROOT_PASSWORD=motdepasse
```

Cela crée un secret contenant une clé (MARIADB_ROOT_PASSWORD) et une valeur (motdepasse).

Utiliser dans un déploiement :

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: mariadb
```

```
spec:
```

```
  replicas: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: mariadb
```

```
template:  
  metadata:  
    labels:  
      app: mariadb  
  spec:  
    containers:  
      - name: mariadb  
        image: mariadb  
        ports:  
          - containerPort: 3306  
        env:  
          - name: MARIADB_ROOT_PASSWORD  
            valueFrom:  
              secretKeyRef:  
                name: mariadb-secret  
                key: MARIADB_ROOT_PASSWORD
```

Ce déploiement va lancer un conteneur mariadb, et lui passer le mot de passe motdepasse via la variable d'environnement MARIADB_ROOT_PASSWORD, sans l'écrire en clair

Test final

Déployer MariaDB :

```
kubectl apply -f mariadb-deployment.yaml
```

```
SECRET/mariadb-secret created  
dome@kubes-01:~$ sudo kubectl apply -f mariadb-deployment.yaml  
deployment.apps/mariadb configured  
dome@kubes-01:~$ |
```

Vérifie si le pod tourne bien :

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
apache-5fd955856f-8gdpr	1/1	Running	1 (3h27m ago)	3d4h
apache-5fd955856f-hbqsv	1/1	Running	1 (142m ago)	154m
apache-5fd955856f-smkhj	1/1	Running	2 (142m ago)	3d4h
mariadb-cb5f99946-txxql	1/1	Running	0	26s
nginx-5b484d9bfb-cbpn6	1/1	Running	0	66m

- ✓ Le secret mariadb-secret a bien été créé.
- ✓ Le déploiement mariadb a été appliqué correctement.
- ✓ Le pod mariadb-cb5f99946-txxql est bien en cours d'exécution (Running), sans redémarrage.

RBAC : Contrôle d'accès (Job 08)

Qu'est-ce que RBAC ?

Role-Based Access Control permet de définir qui peut faire quoi dans son cluster Kubernetes.

- Role : définit les permissions (ex. : lire les pods).
- RoleBinding : attache ce rôle à un utilisateur/groupe.

1. Créer un rôle (lecture des pods) role-pod-reader.yaml :

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods-binding
  namespace: default
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Appliquer :

```
kubectl apply -f role-pod-reader.yaml
```

2. Créer un ServiceAccount : serviceaccount-demo.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: lecteur-pod
  namespace: default
Appliquer :
kubectl apply -f serviceaccount-demo.yaml
```

3. Lier le rôle au ServiceAccount, attacher à un utilisateur rolebinding-pod-reader.yaml :

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods-binding
  namespace: default
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Appliquer :

```
kubectl apply -f rolebinding-reader.yaml
```

Tester le ServiceAccount avec un curl vers l'API

1. Récupère le token :

Depuis Kubernetes 1.24, les tokens ne sont plus automatiquement stockés dans des Secrets.

Il faut utiliser la commande officielle suivante :

```
sudo kubectl -n default create token lecteur-pod
```

Cette commande :

- Génère un token d'accès temporaire pour son ServiceAccount.
- Fonctionne dans tous les clusters récents (K3s, K8s >=1.24).
- Est plus sûre et plus simple à utiliser que les anciens secrets.

```
dome@kubes-01:~$sudo kubectl -n default create token lecteur-podd
[sudo] Mot de passe de dome :
eyJhbGciOiJSUzI1NiIsImtpZCI6InpGZTl00F9zNFdLbUk0TElsdFlmU2ROTVp1M1BHbTZWLFliaHhyZXlmTz
QifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcmt5ldGVzLmRlZmF1bHQuc3ZjLmNsdxN0ZXIubG9jYWwiLCJrM3MiX
SwizXhwIjoxNzUzMdg4MjcxLCJpYXQiOjE3NTMwODQ2NzEsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRLcy5kZWZ
ndWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwanRpIjoiMTkxY2NjMDMtMTA0NS000Dk1LTgzYTItZmJmYWE2NDYwY
zNhTiwia3ViZXJuZXRLcyI6eyJuYW1lc3BhY2UiOijkZWZhdWx0Iiwick2VydmljZWfjY291bnQiOnsibmF
tZSI6ImxLY3RldXItcG9kIiwidWlkIjoiMjdjMTliYWEtMjk20S00NzM4LWEzNDgtNzQ2NWI5NTM0MTU4In19L
CJuYmYi0jE3NTMwODQ2NzEsInN1YiI6InN5c3RlbTpzZXJ2aWNlyWNjb3VudDpkZWZhdWx0Omxy3RldXItcG9
xIn0.RyYFvLL4g0N5qkAFCCbeTxF9171CqRUnzVUJ7P5scLABT0YEWCoBLIWU7MWSn4D-HQcjkIgpujUe8dGpH
/n209ipYmw5BIPizC3wv8pCHe46DMTpixvrES2xxIoAZ2ak29qT0FI1HhRgLlRYXab0wfsvJuIo0gI0K80NrB9S
Kpb-YyadTw6HK21NoJmdnxzm0mc6_tLdLPntvr_xbTsQYb68Z0Ei2jTkFBekAneVtM0qpnkFCgwkNSdkJzPyVT
St0k10QqzYED00kao3U1V9SFkUjsNFpETCEzIx0o4K366VTYZHJoZ0kDjKE7Khppc449p7eatVwN46HHl03gvK
61w
```

Garde ce token JWT, c'est l'identifiant du ServiceAccount.

2. Récupère l'adresse du cluster (et certificat si nécessaire)

```
kubectl config view --minify -o jsonpath="{.clusters[0].cluster.server}"
```

Note aussi ce lien (souvent ou autre).

3. Lancer la requête test :

```
curl -k -H "Authorization: Bearer <TOKEN>" \
https://<CLUSTER_IP>:6443/api/v1/namespaces/default/pods
```

```
dome@kubes-01:~$ curl -k -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6InpGZTl00F9zNFdLbUk0TElsdFlmU2ROTvp1M1BhbTZwZlfiaHhyZXlmTzQifQ .eyJhdWQiolsiaHR0cHM6Ly9rdWJlcml5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiLCJrM3MiXSwiZXhwIjoxNzUzMdg4MjcxLCJpYXQiOjE3NTMwODQ2NzEsImlzcyI6Imh0dBz0i8va3ViZXJuZXRLcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwianRpIjoimTlxY2NjMDMtMTA0NS00Dk1LTgzYTItZmJmYWE2NDYwYzNhIiwia3ViZXJuZXRLcy5pbvI6eyJuYW1lc3BhY2UiOjKzWZhdWx0Iiwic2VydmljZWFjY291bnQiOnsibmFtZSI6Imxly3RldXItcG9kIiwidWlkIjoiMjdjMTliYWEtMjk20S00NzM4LWEzNDgtNzQ2NWI5NTM0MTU4In19LCJuYmYi0jE3NTMwODQ2NzEsInN1YiI6InN5c3RlbTpzZXJ2aWNLYWNjb3VudDpkZWZhdWx00mxly3RldXItcG9kIn0 .RyYFvLL4g0N5qkAFCCbeTxF9171CqRUnzVUJ7P5scLABT0YEWCoBLIWU7MWSn4D-HQcjkIgpjue8dGpHVn209ipYmw5BIPizC3wv8pCHe46DMTpvrES2xxIoAZ2ak29qT0FI1HhRgLlRYXab0wfnnJuIo0gI0K80NrB9SXpb-YyadTw6HK21NoJmdnxzm0mc6_tLdLPntrv_xbTsQYb68Z0Ei2jTkFBekAnevtM0qpnkFCgwkNSdKJzPyVTGt0k10QqzYED00kao3U1V9SFkUjsNFpETCEzIx0o4K366VTYZHJoZ0kDjKE7Khppc449p7eatVwN46HHl03gvK61w" https://192.168.40.153:6443/api/v1/namespaces/default/pods
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "16294"
  },
  "items": [
    {
      "metadata": {
        "name": "apache-5fd955856f-8gdpr",
        "generateName": "apache-5fd955856f-",
        "namespace": "default",
      }
    }
  ]
}
```

L'utilisateur doit voir une liste des pods (succès = RBAC fonctionne en lecture).

4. Essaye une commande non autorisée :

```
curl -k -X DELETE -H "Authorization: Bearer <TOKEN>" \
https://<CLUSTER_IP>:6443/api/v1/namespaces/default/pods/<UN POD>
```

L'utilisateur devrait obtenir une erreur "Forbidden" : preuve que son rôle est bien restreint.

```
dome@kubes-01:~$ curl -k -X DELETE \ "Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6InpGZTl00F9zNFdLbUk0TElsdFlmU2ROTvp1M1BhbTZwZlfiaHhyZXlmTzQifQ .eyJhdWQiolsiaHR0cHM6Ly9rdWJlcml5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiLCJrM3MiXSwiZXhwIjoxNzUzMdg4MjcxLCJpYXQiOjE3NTMwODQ2NzEsImlzcyI6Imh0dBz0i8va3ViZXJuZXRLcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwianRpIjoimTlxY2NjMDMtMTA0NS00Dk1LTgzYTItZmJmYWE2NDYwYzNhIiwia3ViZXJuZXRLcy5pbvI6eyJuYW1lc3BhY2UiOjKzWZhdWx0Iiwic2VydmljZWFjY291bnQiOnsibmFtZSI6Imxly3RldXItcG9kIiwidWlkIjoiMjdjMTliYWEtMjk20S00NzM4LWEzNDgtNzQ2NWI5NTM0MTU4In19LCJuYmYi0jE3NTMwODQ2NzEsInN1YiI6InN5c3RlbTpzZXJ2aWNLYWNjb3VudDpkZWZhdWx00mxly3RldXItcG9kIn0 .RyYFvLL4g0N5qkAFCCbeTxF9171CqRxf9171CqRUnzVUJ7P5scLABT0YEWCoBLIWU7MWSn4D-HQcjkIgpjue8dGpHVn209ipYmw5BIPizC3wv8pCHe46DMTpvrES2xxIoAZ2ak29qT0FI1HhRgLlRYXab0wfnnJuIo0gI0K80NrB9SXpb-YyadTw6HK21NoJmdnxzm0mc6_tLdLPntrv_xbTsQYb68Z0Ei2jTkFBekAnevtM0qpnkFCgwkNSdKJzPyVTGt0k10QqzYED00kao3U1V9SFkUjsNFpETCEzIx0o4K366VTYZHJoZ0kDjKE7Khppc449p7eatVwN46HHl03gvK61w" https://192.168.40.153:6443/api/v1/namespaces/default/pods
curl: (3) URL using bad/illegal format or missing URL
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}dome@kubes-01:~$ |
```

Gestion avancée avec Helm (Job 09)

Objectifs

1. Installer et configurer Helm.
2. Rechercher, installer, personnaliser, mettre à jour et désinstaller des applications à l'aide de Helm.
3. Apprendre à utiliser des charts Helm avec des valeurs personnalisées.

Comprendre Helm

Helm est un gestionnaire de packages pour Kubernetes (comme apt ou yum pour Linux). Il permet :

- d'automatiser le déploiement d'applications sur K8s
- de les mettre à jour facilement
- de partager les déploiements sous forme de "chart" (modèle YAML + valeurs personnalisables)

Un chart est comme une "recette" pour déployer une application avec tous les fichiers nécessaires (Deployment, Service, Ingress, etc.).

Étapes détaillées (avec commandes)

Étape 1 : Installer Helm

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

```
dome@kubes-01:~$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-hel
m-3 | bash
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
          Dload  Upload   Total   Spent    Left  Speed
100 11913  100 11913    0      0  160k      0 --:--:-- --:--:-- --:--:--  166k
Downloaded https://get.helm.sh/helm-v3.18.4-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
[sudo] Mot de passe de dome :
helm installed into /usr/local/bin/helm
```

Vérification :

```
helm version
```

```
dome@kubes-01:~$ helm version
version.BuildInfo{Version:"v3.18.4", GitCommit:"d80839cf37d860c8aa9a0503fe463278f26cd5
e2", GitTreeState:"clean", GoVersion:"go1.24.4"}
```

L'utilisateur doit voir une version comme v3.x.x.

Étape 2 : Ajouter un dépôt Helm (ex : Bitnami)

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

```
helm repo update
```

```
dome@kubes-01:~$ helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
"bitnami" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⚡Happy Helming!⚡
```

Étape 3 : Rechercher une application

Par exemple, on va chercher WordPress :

```
helm search repo wordpress
```

```
dome@kubes-01:~$ helm search repo wordpress
NAME          CHART VERSION   APP VERSION   DESCRIPTION
bitnami/wordpress    25.0.3        6.8.2        WordPress is the world's most
popular blogging ...
bitnami/wordpress-intel 2.1.31      6.1.1        DEPRECATED WordPress for Intel
is the most popu...
```

Étape 4 : Installer l'application (ex : WordPress)

```
helm install monwordpress bitnami/wordpress
```

```

dome@kubes-01:~$ helm install monwordpress bitnami/wordpress
NAME: monwordpress
LAST DEPLOYED: Mon Jul 21 11:41:41 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
  CHART NAME: wordpress
  CHART VERSION: 25.0.3
  APP VERSION: 6.8.2

  Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure
  software supply chain features, unlimited pulls from Docker, LTS support, or applicatio
  n customization, see Bitnami Premium or Tanzu Application Catalog. See https://www.arro
  w.com/globalecs/na/vendors/bitnami for more information.

  ** Please be patient while the chart is being deployed **

  Your WordPress site can be accessed through the following DNS name from within your clu
  ster:

    monwordpress.default.svc.cluster.local (port 80)

  To access your WordPress site from outside the cluster follow the steps below:

  1. Get the WordPress URL by running these commands:

    NOTE: It may take a few minutes for the LoadBalancer IP to be available.
          Watch the status with: 'kubectl get svc --namespace default -w monwordpress'

    export SERVICE_IP=$(kubectl get svc --namespace default monwordpress --template "{{range .status.loadBalancer.ingress 0 }}{{.}}{{end }}")

```

Cela déploie WordPress avec les valeurs par défaut.

Étape 5 : Personnaliser les valeurs

l'utilisateur peut créer un fichier valeurs.yaml :

```

wordpressUsername: admin
wordpressPassword: monmotdepasse
service:
  type: LoadBalancer

```

Et ensuite :

```
helm install monwordpress-personnalise bitnami/wordpress -f valeurs.yaml
```

```

dome@kubes-01:~$ sudo nano valeurs.yaml
dome@kubes-01:~$ helm install monwordpress-personnalise bitnami/wordpress -f valeurs.ya
ml
NAME: monwordpress-personnalise
LAST DEPLOYED: Mon Jul 21 11:46:28 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
  CHART NAME: wordpress
  CHART VERSION: 25.0.3
  APP VERSION: 6.8.2

```

Qu'est-ce que ça veut dire « Personnaliser les valeurs » dans Helm ?

Quand l'utilisateur fais :

```
helm install monwordpress bitnami/wordpress
```

l'utilisateur déploie WordPress avec des valeurs par défaut.

Mais parfois l'utilisateur veux changer le mot de passe admin, le type de service, le nom de la base de données, etc.

Helm lui permet de personnaliser ces réglages via un fichier YAML ou avec des options --set.

Exemple en ligne avec --set

Alternative rapide :

```
helm install wp-rapide bitnami/wordpress \
--set wordpressUsername=admin \
--set wordpressPassword=motdepasse \
--set service.type=NodePort
```

Mais ce n'est pas lisible ni pratique si l'utilisateur as beaucoup de paramètres.

Étape 6 : Mettre à jour une application

```
helm upgrade monwordpress bitnami/wordpress --set wordpressPassword=nouveaupass
```

Étape 7 : Supprimer une application

```
helm uninstall monwordpress
```

```
dome@kubes-01:~$ helm uninstall monwordpress
release "monwordpress" uninstalled
dome@kubes-01:~$ |
```