

Kubernetes for Teenagers

**CONVIENT AUSSI AUX
ADULTES**



Installation kubectl - minikube



```
● ● ●

# Téléchargement de la dernière version de kubectl
# Remplacez [VERSION] par la version souhaitée, par exemple
'v1.23.0'
curl -LO
"https://dl.k8s.io/release/[VERSION]/bin/linux/amd64/kubectl"

# Rendre le binaire exécutable
chmod +x ./kubectl

# Déplacer le binaire dans un répertoire du PATH
sudo mv ./kubectl /usr/local/bin/kubectl

# Vérification de l'installation
kubectl version --client

# ——————


# Installation de Minikube, un outil qui exécute un cluster
# Kubernetes localement
# ligne de commande standard : curl -Lo minikube
#"https://storage.googleapis.com/minikube/releases/[VERSION]/minikube-linux-amd64" && chmod +x minikube
# Remplacer [VERSION] par la version spécifique de Minikube
curl -Lo minikube
"https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64" && chmod +x minikube

# Démarrage d'un cluster Kubernetes local avec Minikube
# ligne de commande standard : minikube start
minikube start

# Vérification que kubectl est correctement installé et
# configuré
# ligne de commande standard : kubectl version --client
kubectl version --client
```

Configuration kubectl



```
# Configurer kubectl pour utiliser un fichier kubeconfig
# spécifique
# [CHEMIN_DU_FICHIER_KUBECONFIG] est le chemin vers votre
# fichier de configuration
export KUBECONFIG=[CHEMIN_DU_FICHIER_KUBECONFIG]

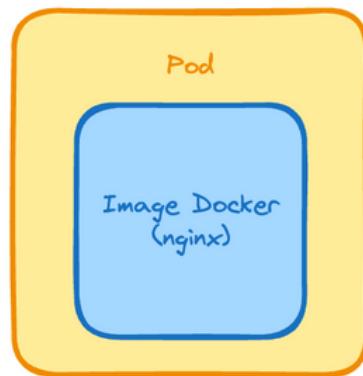
# Vérification de la configuration actuelle
kubectl config view

# Définition du cluster, de l'utilisateur et du contexte par
# défaut
kubectl config set-cluster [NOM_DU_CLUSTER] --server=
[ADRESSE_DU_SERVEUR]
kubectl config set-credentials [NOM_UTILISATEUR] --client-
certificate=[CHEMIN_CERTIFICAT] --client-key=[CHEMIN_CLE]
kubectl config set-context [NOM_DU_CONTEXTE] --cluster=
[NOM_DU_CLUSTER] --user=[NOM_UTILISATEUR]
kubectl config use-context [NOM_DU_CONTEXTE]
```

Pod



Les Pods sont les plus petites unités déployables créées et gérées par Kubernetes. Un Pod est un groupe d'un ou plusieurs conteneurs.



Les Pods peuvent être créés à l'aide de fichiers de configuration YAML, offrant plus de contrôle et de flexibilité.

```
● ● ●  
apiVersion: v1  
kind: Pod  
metadata:  
  name: monpod  
spec:  
  containers:  
    - name: moncontainer  
      image: nginx
```

monpod.yaml

Gestion des Pods



```
● ● ●

# Appliquer le fichier YAML pour créer le Pod
kubectl apply -f monpod.yaml

# Sinon crée un Pod avec un conteneur basé sur l'image nginx
kubectl run monpod --image=nginx

# Vérification de la création du Pod
kubectl get pods

# Afficher tous les Pods avec des détails d'état
kubectl get pods -o wide

# Obtenir des informations détaillées sur un Pod spécifique
kubectl describe pod monpod

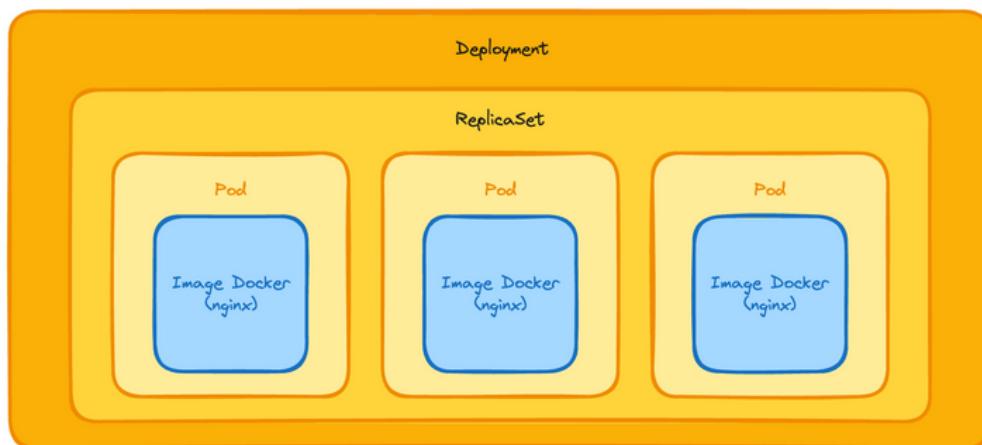
# Afficher les logs d'un Pod
kubectl logs monpod

# Exécuter une commande dans un conteneur d'un Pod
kubectl exec monpod -- [COMMAND]
```

Deployment



Un Deployment gère un ensemble de répliques de votre application, assurant ainsi son déploiement et sa mise à l'échelle.



Les Deployments sont souvent définis et configurés via des fichiers YAML.

```
● ● ●

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mondeployment
spec:
  replicas: 3  # Nombre de répliques du Pod
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
```

Gestion des Deployments



```
# Applique le fichier YAML pour créer le Deployment
kubectl apply -f mondeployment.yaml

# Sinon Crée un Deployment nommé 'mondeployment' utilisant
l'image nginx
kubectl create deployment mondeployment --image=nginx

# Vérifier le Deployment créé
kubectl get deployments

# Mise à l'échelle du Deployment pour avoir 5 réplicas
kubectl scale deployment mondeployment --replicas=5

# Vérifier la mise à l'échelle
kubectl get deployment mondeployment

# Mettre à jour l'image du conteneur dans le Deployment
kubectl set image deployment/mondeployment nginx=nginx:1.16.1

# Vérifier le déploiement mis à jour
kubectl rollout status deployment/mondeployment

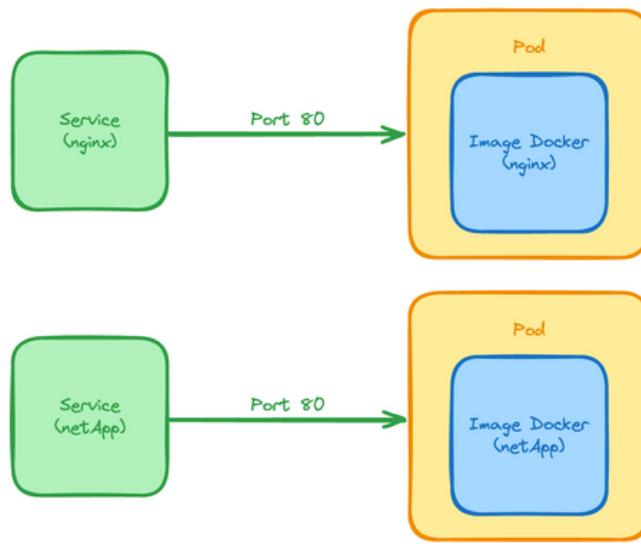
# Annuler la dernière mise à jour du Deployment
kubectl rollout undo deployment/mondeployment

# Supprimer un Deployment par son nom
kubectl delete deployment mondeployment
```

Service



Un Service dans Kubernetes est une abstraction qui définit un ensemble logique de Pods et une politique permettant d'y accéder.



Les Services peuvent être configurés de manière plus détaillée via des fichiers YAML, notamment pour définir différents types de Services comme ClusterIP, NodePort, ou LoadBalancer.

```

● ● ●

apiVersion: v1
kind: Service
metadata:
  name: monservice
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  
```

Gestion des Services



```
# Appliquer le fichier YAML pour créer le Service
kubectl apply -f monservice.yaml

# Sinon créer un Service de type ClusterIP (par défaut) pour
# exposer le Deployment
kubectl expose deployment mondeployment --port=80 --
type=ClusterIP

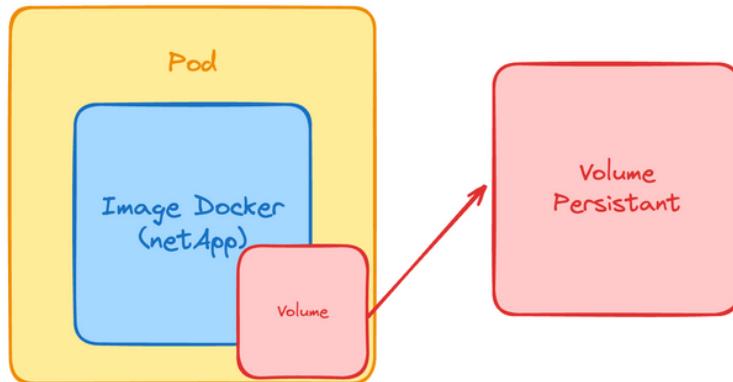
# Vérifier le Service créé
kubectl get services

# Supprimer un Service par son nom
kubectl delete service monservice
```

Volume



Dans Kubernetes, un volume est une unité de stockage attachée à un Pod, qui existe tant que le Pod existe. Un volume persistant (PersistentVolume, PV), en revanche, est une ressource de stockage dans le cluster qui reste indépendante de la durée de vie des Pods individuels. Les PersistentVolumeClaims (PVC) sont des demandes de stockage par les utilisateurs qui peuvent être liées à des PV pour fournir un stockage persistant.



```

● ● ●

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

Gestion des Volumes



```
● ● ●

apiVersion: v1
kind: Pod
metadata:
  name: monpod
spec:
  containers:
  - name: moncontainer
    image: netApp
    volumeMounts:
    - mountPath: "/var/www/html"
      name: monvolume
  volumes:
  - name: monvolume
    persistentVolumeClaim:
      claimName: monpvc
```

```
● ● ●

# Appliquer le fichier YAML pour créer le PVC
kubectl apply -f monpvc.yaml

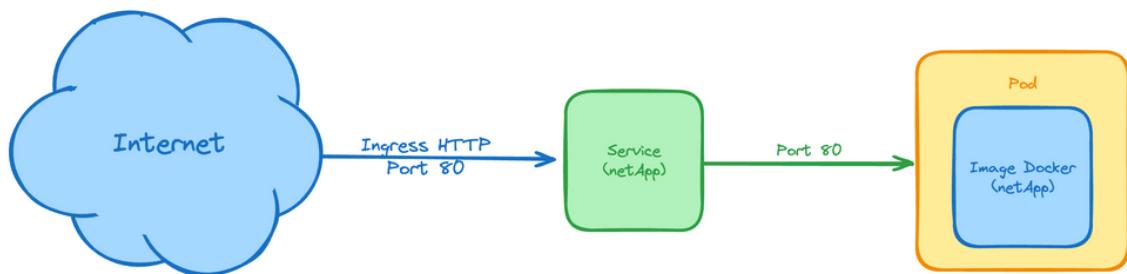
# Appliquer le fichier YAML pour créer le Pod avec le volume
kubectl apply -f monpod.yaml

# Supprimer un PVC par son nom
kubectl delete pvc monpvc
```

Ingress



Dans Kubernetes, les réseaux permettent la communication entre les différents composants, tels que les Pods, les Services, et l'extérieur du cluster.



Ingress est un objet Kubernetes qui gère l'accès externe aux services dans un cluster, typiquement HTTP.

```
● ● ●

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: moningress
spec:
  rules:
  - host: monapp.exemple.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: monservice
            port:
              number: 80
```

Gestion des Ingress



```
● ● ●

# Appliquer le fichier YAML pour créer l'Ingress
kubectl apply -f moningress.yaml

# Lister tous les Ingress dans le namespace courant
kubectl get ingress

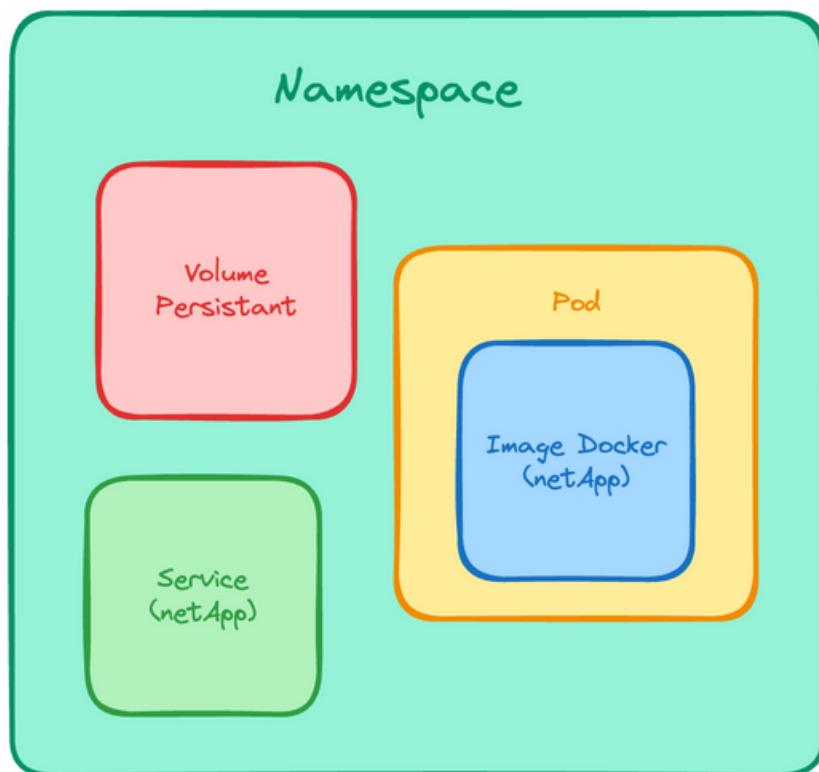
# Ou pour lister les Ingress dans tous les namespaces
kubectl get ingress --all-namespaces

# Supprimer un Ingress
kubectl delete ingress moningress
```

Namespace



Les namespaces Kubernetes offrent une manière de diviser les ressources du cluster entre plusieurs utilisateurs et projets. Ils sont utiles pour créer des environnements isolés au sein d'un même cluster.



Gestion des Namespaces



```
● ● ●  
  
# Créer un namespace nommé 'monnamespace'  
kubectl create namespace monnamespace  
  
# Afficher tous les namespaces dans le cluster  
kubectl get namespaces  
  
# Créer un Pod dans un namespace spécifique  
kubectl run monpod --image=nginx --namespace=monnamespace  
  
# Lister tous les Pods dans 'monnamespace'  
kubectl get pods --namespace=monnamespace  
  
# Lister toutes les ressources dans un namespace donné  
kubectl get all --namespace=monnamespace  
  
# Supprimer un namespace et toutes ses ressources  
kubectl delete namespace monnamespace
```

Authentification



La sécurité dans Kubernetes repose fortement sur l'utilisation de tokens pour l'authentification des utilisateurs et des processus. Les tokens peuvent être des jetons d'API, des jetons de compte de service, ou d'autres formes d'identifiants.

Kubernetes utilise RBAC (Role-Based Access Control) pour gérer les autorisations des utilisateurs et des comptes de service.

```
● ● ●

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: monrole
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
● ● ●

apiVersion: v1
kind: ServiceAccount
metadata:
  name: moncompteservice
  namespace: default
```

Gestion des comptes



```
● ● ●

# Créer un compte de service :
kubectl create serviceaccount moncompte

# Récupérer le token du compte de service :
kubectl get secret $(kubectl get serviceaccount moncompte -o jsonpath='{.secrets[0].name}') -o jsonpath='{.data.token}' | base64 --decode

# Configurer kubectl avec le token :
kubectl config set-credentials moncompte --token=[TOKEN]
kubectl config set-context --current --user=moncompte

# Appliquer le rôle à partir d'un fichier YAML
kubectl apply -f monrole.yaml

# Attribuez des rôles aux comptes de service ou aux utilisateurs.
kubectl create rolebinding monrolebinding --role=monrole --serviceaccount=default:moncompte

# Lister tous les rôles dans un namespace
kubectl get roles --namespace=default

# Lister tous les rolebindings dans un namespace
kubectl get rolebindings --namespace=default

# Supprimer un rôle
kubectl delete role monrole --namespace=default

# Supprimer un rolebinding
kubectl delete rolebinding monrolebinding --namespace=default
```

Network Policies



Les Network Policies dans Kubernetes permettent de contrôler comment les Pods peuvent communiquer entre eux et avec d'autres points de terminaison réseau.

Les politiques réseau sont définies à l'aide de fichiers YAML qui spécifient les règles de trafic entrant (ingress) et sortant (egress).

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ma-policy
spec:
  podSelector:
    matchLabels:
      app: monapp
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: monapp
      ports:
        - protocol: TCP
          port: 80
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
```

Gestion des Network Policies



```
● ● ●

# Appliquer la Network Policy
kubectl apply -f ma-policy.yaml

# Lister toutes les Network Policies dans un namespace
kubectl get networkpolicies --namespace=default

# Supprimer une Network Policy
kubectl delete networkpolicy ma-policy
```

ConfigMap



Les ConfigMaps permettent de stocker des données de configuration externes aux Pods, ce qui aide à la gestion et au déploiement d'applications.

Les ConfigMaps peuvent être utilisées dans les Pods en tant que variables d'environnement, arguments de commande, ou comme des fichiers de configuration dans un volume.

```
● ● ●

apiVersion: v1
kind: ConfigMap
metadata:
  name: maconfigmap
data:
  clef1: valeur1
  clef2: valeur2
```

ConfigMap



```
● ● ●

apiVersion: v1
kind: Pod
metadata:
  name: monpod
spec:
  containers:
  - name: moncontainer
    image: nginx
    env:
      - name: CONFIG_KEY
        valueFrom:
          configMapKeyRef:
            name: maconfigmap
            key: clef1
```

Gestion des ConfigMaps



```
# Créer une ConfigMap à partir d'un YAML
kubectl apply -f maconfigmap.yaml

# Créer une ConfigMap avec des valeurs clé-valeur spécifiées
kubectl create configmap maconfigmap --from-
literal=clef1=valeur1 --from-literal=clef2=valeur2

# Créer une ConfigMap à partir d'un fichier de configuration
kubectl create configmap maconfigmap --from-
file=path/to/configfile

# Modifier une ConfigMap
kubectl edit configmap maconfigmap

# Ou recréer une ConfigMap avec de nouvelles données
kubectl create configmap maconfigmap --from-
file=path/to/newconfigfile --dry-run=client -o yaml | kubectl
apply -f -

# Supprimer une ConfigMap
kubectl delete configmap maconfigmap
```

Secret



Les secrets Kubernetes sont utilisés pour stocker et gérer des informations sensibles, telles que les mots de passe, les jetons OAuth et les clés SSH. Ils permettent de séparer les informations sensibles des fichiers de configuration ou des images de conteneurs.

```
● ● ●

apiVersion: v1
kind: Secret
metadata:
  name: monsecret
type: Opaque
data:
  clef1: dmFsdWVyMQ== # La valeur en base64 pour "valeur1"
  clef2: dmFsdWVyMg== # La valeur en base64 pour "valeur2"
```

Secret



```
apiVersion: v1
kind: Pod
metadata:
  name: monpod
spec:
  containers:
  - name: moncontainer
    image: nginx
    env:
      - name: SECRET_KEY
        valueFrom:
          secretKeyRef:
            name: monsecret
            key: clef1
```

Gestion des Secrets



```
● ● ●

# Créer un fichier à partir d'un fichier YAML
kubectl apply -f monsecret.yaml

# Créer un secret à partir d'un fichier
kubectl create secret generic monsecret --from-
file=path/to/bar

# Créer un secret à partir de paires clé-valeur
kubectl create secret generic monsecret --from-
literal=clef1=valeur1 --from-literal=clef2=valeur2

# Recréer un secret
kubectl create secret generic monsecret --from-
literal=clef1=nouvelleValeur --dry-run=client -o yaml |
kubectl apply -f -

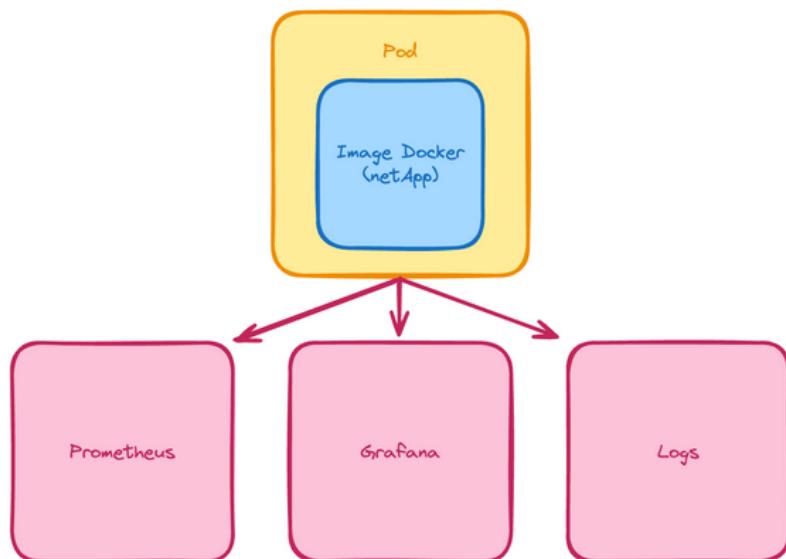
# Supprimer un secret
kubectl delete secret monsecret
```

Gestion des logs



Le monitoring est crucial pour maintenir la santé et les performances de votre cluster Kubernetes. Il implique la surveillance des ressources, des performances et de la santé des Pods, des nœuds et d'autres composants.

Kubernetes peut être intégré avec divers outils de monitoring tels que Prometheus, Grafana, etc.



```
# Afficher les logs d'un Pod spécifique  
kubectl logs monpod
```