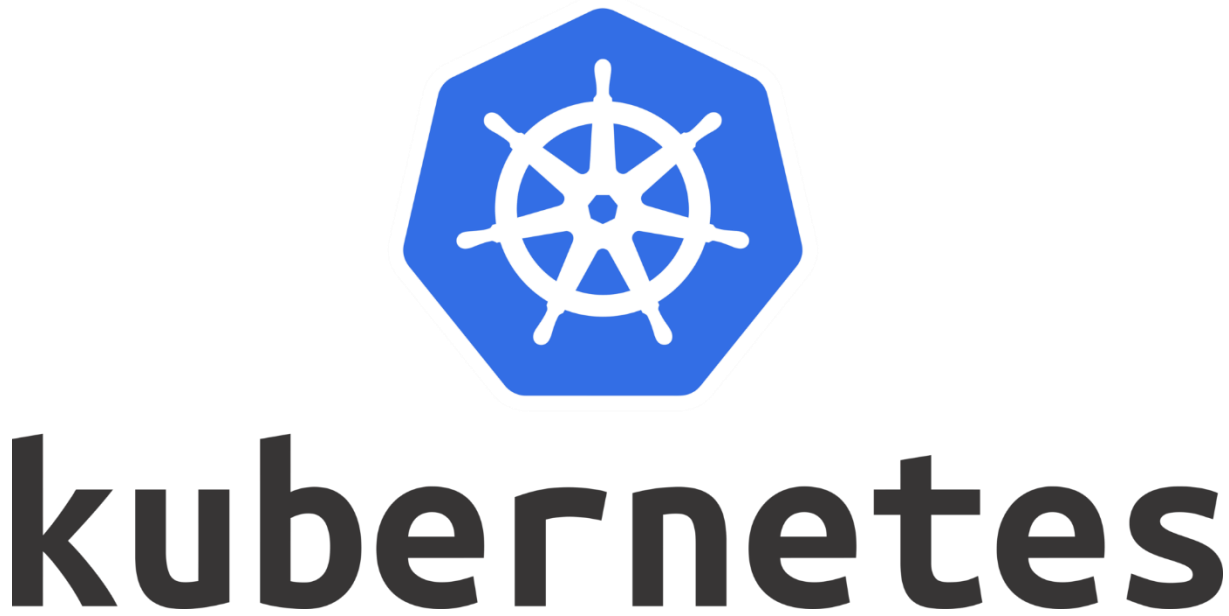

Auteur : Dasek Joiakim

LI : 1 sur 2

Ressources :
https://www.udemy.com/course/la_plateforme_k8s/learn/lecture/9926950#overview

https://www.udemy.com/course/kubernetes-les-bases-indispensables/learn/lecture/22021750#overview

Introduction

Kubernetes est une plateforme open-source conçue pour automatiser le déploiement, la mise à l'échelle et la gestion d'applications conteneurisées. Son objectif principal est de simplifier et d'optimiser le déploiement et la gestion d'applications dans des environnements cloud ou sur site.

L'une des problématiques majeures que Kubernetes résout est la gestion complexe des applications distribuées et des infrastructures à grande échelle. Avant l'avènement de Kubernetes, le déploiement et la gestion d'applications sur des infrastructures complexes nécessitaient souvent une coordination manuelle intensive, ce qui pouvait entraîner des erreurs, des temps d'arrêt et une inefficacité opérationnelle.

Kubernetes fournit une solution en automatisant de nombreux aspects de la gestion des applications, tels que le déploiement, la mise à l'échelle automatique en fonction de la charge, la gestion des ressources, la résilience aux pannes et la mise à jour des applications sans temps d'arrêt. Cela permet aux équipes de développement et d'exploitation de se concentrer davantage sur le développement d'applications et moins sur la gestion des infrastructures sous-jacentes.

Rappel

Docker est une plateforme open-source qui permet de développer, de déployer et d'exécuter des applications dans des conteneurs. Voici quelques rappels et notions importantes sur Docker :

- **Processus de conteneurisation** : Docker utilise la technologie de conteneurisation pour encapsuler une application et ses dépendances dans un conteneur léger et autonome. Chaque conteneur exécute un processus isolé, ce qui permet d'exécuter plusieurs applications sur une même machine hôte sans conflit.
- **Visibilité sur la machine hôte** : Contrairement aux machines virtuelles, les conteneurs Docker partagent le même noyau avec la machine hôte. Cela signifie que les conteneurs sont visibles et gérables directement depuis la machine hôte, ce qui simplifie leur gestion et leur surveillance.
- **Limite dans les ressources** : Docker permet de définir des limites de ressources pour chaque conteneur, telles que la mémoire, le CPU et le stockage, afin de garantir un partage équitable des ressources système entre les différents conteneurs exécutés sur la même machine.
- **Combinaison de deux primitives Linux** : Docker repose sur deux fonctionnalités du noyau Linux : les espaces de noms (namespaces) et les groupes de contrôle (cgroups). Les espaces de noms permettent l'isolement des processus, des réseaux et des systèmes de fichiers, tandis que les cgroups permettent de limiter et de surveiller l'utilisation des ressources par les processus.
- **Portabilité et reproductibilité** : Les conteneurs Docker sont portables et reproductibles, ce qui signifie qu'ils peuvent être exécutés de manière cohérente sur n'importe quelle plateforme

compatible avec Docker, sans modification supplémentaire. Cela facilite le déploiement des applications dans des environnements de développement, de test et de production.

Architecture

L'architecture monolithique est un style traditionnel de conception d'applications où toutes les fonctionnalités sont regroupées en un seul et même bloc logiciel, généralement déployé comme une seule unité. Dans ce modèle, toutes les parties de l'application sont développées, déployées et gérées ensemble. Cela signifie que toute modification ou mise à jour nécessite souvent le déploiement de l'ensemble de l'application, ce qui peut être complexe et potentiellement risqué. Bien que l'architecture monolithique soit simple à mettre en œuvre et à déployer au début, elle peut devenir difficile à maintenir et à faire évoluer à mesure que l'application grossit.

D'autre part, l'architecture de microservices est un style de conception où une application est décomposée en un ensemble de services autonomes, chacun responsable d'une fonctionnalité spécifique. Ces services communiquent entre eux généralement via des API, et chaque service peut être développé, déployé et mis à l'échelle indépendamment des autres. Cette approche favorise la modularité, la scalabilité et la flexibilité de l'application. Les microservices permettent également une plus grande résilience, car une défaillance dans un service n'entraîne pas nécessairement l'arrêt de l'ensemble de l'application.

La plateforme

Le projet est aussi nommé « k8s » a plusieurs fonctionnalités :

- Gestion d’applications tournant dans des containers.
- Self-healing, capacité de k8s de détecter des failure et réparer automatiquement.
- Service discovery, permet de dévouvrir et communiquer avec d’autres services au sein du même cluster.
- Gestion des secrets et des configurations
- Long-running process et batch jobs
- Role Based Access Control (RBAC)
- Orchestration du stockage

Les concepts de base

Un cluster est une infrastructure de calcul distribué composée de plusieurs machines interconnectées, appelées "nodes" ou nœuds, qui coopèrent pour exécuter des applications ou des services. Chaque node peut jouer un rôle spécifique dans le cluster, notamment en tant que master/control plane ou slave/worker.

1. `Node` :

- Un node est une machine physique ou virtuelle faisant partie du cluster. Chaque node exécute un ensemble de services qui permettent au cluster de fonctionner et de gérer les charges de travail.
- Il existe deux types principaux de nodes :
    - `Master/Control Plane Node` : Ce type de node est responsable de la gestion globale du cluster. Il héberge les composants de contrôle qui orchestrent et supervisent les opérations du cluster, tels que la planification des tâches, la gestion des nœuds, et la communication avec les autres nodes.
    - `Slave/Worker Node` : Ces nodes exécutent les charges de travail réelles du cluster, telles que les conteneurs ou les applications. Ils sont gérés par les nodes master/control plane et reçoivent des instructions sur les tâches à exécuter.

2. `Cluster` :

- Un cluster est un ensemble de nodes qui travaillent ensemble pour fournir des capacités de calcul et de stockage à une application ou un service. Les clusters peuvent être configurés pour fournir une haute disponibilité, une redondance et une scalabilité horizontale.
- Les clusters permettent de distribuer les charges de travail sur plusieurs machines, ce qui améliore les performances et la résilience de l'infrastructure. Ils facilitent également la gestion centralisée des ressources et des applications.

Plusieurs manières d'accéder à un cluster :

L'accès et le contrôle d'un cluster peuvent se faire de différentes manières, en fonction de la configuration spécifique du cluster et des outils disponibles. Voici quelques méthodes courantes pour accéder et contrôler un cluster :

- **Interface en ligne de commande (CLI)** : De nombreux systèmes de gestion de clusters, tels que Kubernetes, fournissent des interfaces en ligne de commande (CLI) pour interagir avec le cluster. Par exemple, pour Kubernetes, vous pouvez utiliser l'outil en ligne de commande kubectl pour exécuter des commandes et des opérations sur le cluster.
- **Tableau de bord web** : Certains clusters, comme Kubernetes, offrent également un tableau de bord web convivial qui permet de visualiser et de gérer les ressources du cluster via une interface utilisateur graphique (GUI). Vous pouvez accéder à ce tableau de bord à l'aide d'un navigateur web.
- **API programmable** : De nombreux clusters exposent une API RESTful ou d'autres interfaces de programmation pour permettre un contrôle et une automatisation avancés. Vous pouvez utiliser des bibliothèques cliente dans différents langages de programmation pour interagir avec l'API du cluster et effectuer des opérations programmables.
- **Accès SSH** : Dans certains cas, vous pouvez accéder directement aux machines du cluster via Secure Shell (SSH) pour effectuer des tâches de maintenance, de débogage ou de configuration.

- **Outils de gestion tiers** : Il existe également des outils de gestion tiers qui peuvent fournir une interface utilisateur ou une API pour contrôler et gérer un cluster. Par exemple, des plates-formes de gestion de clusters cloud comme AWS EKS, Google Kubernetes Engine (GKE) ou Azure Kubernetes Service (AKS) fournissent des outils intégrés pour gérer les clusters Kubernetes sur leurs infrastructures cloud respectives.

Cluster de développement

Dans l'écosystème de Kubernetes, la mise en place d'un environnement de développement efficace est essentielle pour les développeurs. Minikube et Kind (Kubernetes in Docker) sont deux outils populaires pour créer des clusters Kubernetes locaux, fournissant ainsi un environnement de développement isolé et reproductible.

Minikube

Minikube est une solution légère pour exécuter un cluster Kubernetes mono-noeud sur une machine locale. Il utilise une machine virtuelle (comme VirtualBox ou Docker) pour créer un environnement Kubernetes complet. Les développeurs peuvent rapidement déployer des applications et des services sur Minikube pour tester leurs déploiements localement avant de les déployer sur des clusters de production. Minikube offre une grande flexibilité et prend en charge diverses configurations, ce qui en fait un choix populaire pour le développement et les tests.

Kind

Kind est une autre option pour créer des clusters Kubernetes locaux, mais il diffère de Minikube en utilisant des conteneurs Docker comme nœuds du cluster. Cela rend Kind plus rapide à démarrer et plus léger en ressources que Minikube. Les clusters créés avec Kind sont idéaux pour les tests d'intégration et la validation de configurations Kubernetes spécifiques. Les développeurs peuvent facilement déployer des clusters Kind pour simuler des environnements complexes et réaliser des tests approfondis sans avoir besoin de ressources matérielles importantes.

Avantages des Clusters de Développement :

- **Isolation** : Les clusters de développement fournissent un environnement isolé où les développeurs peuvent expérimenter sans craindre d'impact sur d'autres services ou environnements.
- **Réplicabilité** : Les environnements de développement créés avec Minikube et Kind peuvent être reproduits facilement, assurant ainsi une cohérence entre les environnements de développement des différents membres de l'équipe.
- **Rapidité** : La mise en place rapide de clusters locaux permet aux développeurs de tester et de déployer leurs applications plus rapidement, accélérant ainsi le cycle de développement.

Cluster de production

Pour les déploiements de production à grande échelle sur Kubernetes, les services gérés tels qu'EKS, AKS et GKE offrent une solution robuste et évolutive pour exécuter des clusters Kubernetes dans le cloud.

Amazon Elastic Kubernetes Service (EKS) :

EKS est un service entièrement géré par AWS qui permet aux utilisateurs de déployer des clusters Kubernetes sur l'infrastructure AWS. En utilisant EKS, les entreprises bénéficient de l'intégration étroite avec les services AWS, ce qui simplifie le déploiement, la gestion et l'évolutivité des applications conteneurisées. EKS offre des fonctionnalités avancées telles que la haute disponibilité, la sécurité renforcée et l'intégration transparente avec d'autres services AWS.

Azure Kubernetes Service (AKS) :

AKS est le service Kubernetes géré par Microsoft Azure, offrant une plateforme native pour le déploiement et la gestion de clusters Kubernetes dans le cloud Azure. AKS permet aux entreprises de tirer parti de l'écosystème Azure, y compris les services de surveillance, de sécurité et de mise en réseau, pour créer des solutions cloud-native robustes. AKS propose des intégrations avec d'autres services Azure tels que Azure Monitor, Azure Active Directory et Azure Policy pour simplifier les opérations de gestion des clusters Kubernetes.

Google Kubernetes Engine (GKE) :

GKE est le service Kubernetes géré par Google Cloud, offrant une plateforme hautement évolutive et sécurisée pour exécuter des charges de travail conteneurisées. GKE s'appuie sur l'expertise de Google en matière d'orchestration de conteneurs et offre des fonctionnalités avancées telles que la mise à l'échelle automatique, la gestion des mises à jour du système et la sécurité intégrée. GKE permet aux entreprises de se concentrer sur le développement d'applications sans se soucier de la gestion de l'infrastructure sous-jacente.

DigitalOcean Kubernetes (DOKS) :

DOKS est conçu pour simplifier le processus de déploiement et de gestion des clusters Kubernetes sur l'infrastructure cloud de DigitalOcean. Il offre une intégration transparente avec d'autres services DigitalOcean tels que les espaces de stockage object, les volumes bloquants et les services de base de données managés, ce qui permet aux développeurs de créer des applications cloud-native complètes. DOKS est également connu pour sa simplicité d'utilisation et sa tarification transparente, ce qui en fait une option attrayante pour les startups et les petites entreprises.

Avantages des Clusters de Production Gérés :

- **Fiabilité** : Les clusters gérés EKS, AKS et GKE sont conçus pour offrir une haute disponibilité et une fiabilité maximale, garantissant ainsi des performances constantes pour les charges de travail de production.
- **Sécurité** : Ces services intègrent des fonctionnalités de sécurité avancées pour protéger les clusters Kubernetes contre les menaces potentielles, assurant ainsi la conformité et la confidentialité des données.
- **Évolutivité** : Grâce à la mise à l'échelle automatique et à la gestion simplifiée des ressources, les clusters gérés peuvent s'adapter de manière transparente à la croissance des charges de travail, garantissant ainsi des performances optimales à tout moment.

Les composants de Kubernetes

Les objets : Pod

Un Pod est l'unité de base dans Kubernetes, représentant une instance unique d'une ou plusieurs applications conteneurisées. Il peut être utile de penser à un Pod comme à une enveloppe logique qui contient un ou plusieurs conteneurs, partageant le même espace réseau et de stockage, ainsi que les configurations et les ressources.

Contenu d'un Pod :

- **Conteneurs** : Un Pod peut contenir un ou plusieurs conteneurs qui partagent le même cycle de vie et les mêmes ressources. Ces conteneurs sont généralement conçus pour fonctionner ensemble et interagir les uns avec les autres pour offrir une fonctionnalité cohérente. Par exemple, un Pod pourrait contenir un conteneur pour l'application web et un autre pour la base de données.
- **Espace réseau** : Tous les conteneurs à l'intérieur d'un Pod partagent le même espace réseau. Cela signifie qu'ils peuvent communiquer entre eux via localhost et partager des ports. Kubernetes attribue une adresse IP unique à chaque Pod, permettant ainsi aux autres Pods de communiquer avec lui.
- **Stockage** : Les Pods peuvent inclure des volumes de stockage qui sont montés dans les conteneurs lors de leur exécution. Ces volumes peuvent être utilisés pour stocker des données persistantes ou partager des données entre les conteneurs du même Pod. Les volumes Kubernetes peuvent être de différents types, tels que des volumes vides, des volumes basés sur des hôtes ou des volumes cloud.

Une application est généralement découpée en un ou plusieurs Pod, on peut voir un pod comme un service métier d’une application.

Par exemple :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
spec:
  containers:
  - name: conteneur-web
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: conteneur-bdd
    image: mysql:latest
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "motdepasse"
```

- Le Pod nommé "mon-pod" contient deux conteneurs : "conteneur-web" avec l'image nginx et "conteneur-bdd" avec l'image mysql.
- Le conteneur "conteneur-web" expose le port 80.
- Le conteneur "conteneur-bdd" définit une variable d'environnement "MYSQL_ROOT_PASSWORD" pour configurer le mot de passe root MySQL.

Ce Pod combine ainsi un conteneur pour un serveur web (nginx) et un autre conteneur pour une base de données MySQL dans une seule unité logique. Les deux conteneurs partagent le même espace réseau et peuvent communiquer entre eux localement. De plus, le conteneur MySQL peut utiliser un volume Kubernetes pour stocker ses données de manière persistante.

Cycle de vie

1. Lancement du Pod :
    a. Kubectl create -f nginx-pod.yml
2. Liste des Pods presents
    a. Kubectl get pods
3. Lancement d'une commande dans un Pod
    a. Kubectl exec mon-pod -c conteneur-web -- nginx -v
4. Shell interactif dans un pod
    a. Kubectl exec -it mon-pod -- /bin/bash

5. Détail du Pod
    a. Kubectl describe pod mon-pod
6. Suppression du Pod
    a. Kubectl delete pod mon-pod
7. Redirection de port pour y accéder
    a. Kubectl port-forward mon-pod 8080:80

Étape de Scheduling dans Kubernetes

L'étape de scheduling dans Kubernetes est cruciale pour allouer efficacement les ressources et exécuter les conteneurs sur les nœuds appropriés. Voici quelques éléments clés à considérer :

1. Labels et Sélecteurs

Les labels sont des paires clé-valeur attachées aux objets Kubernetes. Ils permettent de marquer les ressources avec des métadonnées, ce qui facilite le filtrage et la sélection des ressources pour le scheduling. Les sélecteurs sont utilisés pour spécifier les critères de sélection basés sur ces labels.

Exemple :

```
metadata:
  labels:
    app: frontend
    tier: production
```

2. Ressources Minimales et Max

Les ressources minimales et maximales définissent les limites de l'utilisation des ressources (CPU, mémoire, etc.) pour les conteneurs. Cela permet au scheduler de prendre des décisions intelligentes sur le placement des charges de travail en fonction des ressources disponibles sur les nœuds.

Exemple :

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Dans cet exemple, le conteneur nginx demande au moins 64 MiB de mémoire et 250 milli-CPU (ou 0,25 CPU). Il ne peut pas utiliser plus de 128 MiB de mémoire et 500 milli-CPU.

3. Stratégies de Scheduling

Kubernetes offre plusieurs stratégies de scheduling pour placer les conteneurs sur les nœuds disponibles, telles que la répartition équilibrée de la charge, la densité maximale ou les contraintes spécifiques.

Exemple :

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: machine-type
            operator: In
            values:
            - high-memory
            - ssd
```

Dans cet exemple, la charge de travail est planifiée sur des nœuds avec les étiquettes "machine-type: high-memory" ou "machine-type: ssd".

Les objets : Service

Dans Kubernetes, un Service est une abstraction qui définit un ensemble de pods et une politique pour accéder à ces pods. Il fournit une adresse IP stable et un nom DNS pour les communications à l'intérieur ou à l'extérieur du cluster, en fonction de sa configuration. Voici une explication de chaque type de service avec des exemples et des cas d'utilisation appropriés :

1. ClusterIP

Le type de service ClusterIP expose le service sur une adresse IP interne, accessible uniquement à l'intérieur du cluster Kubernetes. C'est idéal pour les communications entre les différents composants internes du cluster.

Exemple de YAML :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mon-service
spec:
  selector:
    app: mon-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Commande de création :

```
kubectl apply -f mon-service.yaml
```

2. NodePort

Le type de service NodePort expose le service sur un port fixe sur chaque nœud du cluster. Il permet d'accéder au service depuis l'extérieur du cluster en utilisant l'adresse IP du nœud et le port spécifié.

Exemple de YAML :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mon-service
spec:
  type: NodePort
  selector:
    app: mon-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Commande de création :

```
kubectl apply -f mon-service.yaml
```

3. LoadBalancer

Le type de service LoadBalancer expose le service sur une adresse IP externe (généralement fournie par un fournisseur cloud) et distribue automatiquement le trafic entre les pods associés au service. C'est idéal pour les charges de travail nécessitant un équilibrage de charge externe.

Exemple de YAML :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mon-service
spec:
  type: LoadBalancer
  selector:
    app: mon-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Commande de création :

```
kubectl apply -f mon-service.yaml
```

Cas d'utilisation appropriés :

- **ClusterIP** : Utilisez-le pour la communication entre les services internes au cluster, comme les microservices s'adressant les uns aux autres.
- **NodePort** : Utile lorsqu'un accès externe est nécessaire, mais sans équilibrage de charge, par exemple pour les tests ou le développement.
- **LoadBalancer** : Idéal pour les applications qui nécessitent un équilibrage de charge externe pour gérer le trafic provenant de l'extérieur du cluster, par exemple pour une application web publique.

Les objets : Deployment

Dans Kubernetes, un déploiement est un objet qui gère le déploiement d'applications sur un cluster. Il fournit des fonctionnalités de gestion des mises à jour, du scaling, et de la mise en œuvre de stratégies de roll-back en cas d'échec. Voici une explication détaillée des déploiements et des concepts associés :

1. Déploiement (Deployment)

Un déploiement définit l'état désiré de l'application et les règles pour maintenir cet état en créant et en mettant à jour des répliques des pods. Il assure le déploiement initial et gère les mises à jour de version de manière contrôlée.

Exemple de YAML :

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

Commande de création :

```
kubectl apply -f mon-deploiement.yaml
```

2. Mise à Jour d'un Déploiement

Pour mettre à jour un déploiement, vous modifiez simplement le champ **spec.template** avec la nouvelle version de l'image. Kubernetes se charge de déployer les nouvelles répliques et d'effectuer une transition en douceur.

Exemple de commande de mise à jour :

```
kubectl set image deployment/mon-deploiement mon-conteneur=nouvelle-image:version
```

On peut aussi :

1. **Modifier le tag de l'image dans le fichier YAML** : Modifiez le champ **spec.template.spec.containers[0].image** pour inclure la nouvelle version de l'image.

2. **Appliquer le changement** : Utilisez **kubectl apply** pour appliquer les modifications au déploiement.

Exemple de YAML modifié :

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

Après avoir enregistré ces modifications dans votre fichier YAML, vous pouvez les appliquer avec la commande suivante :

```
kubectl apply -f mon-deploiement.yaml
```

3. Mise à l'Échelle (Scaling)

La mise à l'échelle d'un déploiement se fait en ajustant le nombre de répliques de pods en cours d'exécution, ce qui peut être effectué à la fois manuellement et automatiquement.

Exemple de mise à l'échelle :

```
kubectl scale deployment mon-deploiement --replicas=5
```

4. Horizontal Pod Autoscaler (HPA)

L'HPA ajuste automatiquement le nombre de répliques de pods en fonction de l'utilisation des ressources, telle que la CPU ou la mémoire. Cela garantit que les applications ont suffisamment de ressources pour répondre à la demande tout en évitant les gaspillages.

Exemple de YAML pour un HPA :

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: mon-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mon-deploiement
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

Ce YAML définit un HPA pour surveiller l'utilisation de la CPU et ajuster automatiquement le nombre de répliques de pods pour maintenir en moyenne 50% d'utilisation de la CPU.

Les objets : Namespace

Un Namespace dans Kubernetes est un moyen de diviser un cluster en plusieurs espaces virtuels logiques. Chaque Namespace fournit une portée isolée pour les ressources Kubernetes telles que les pods, les services, les déploiements, etc. Voici une explication des avantages des Namespaces et des exemples pour différentes opérations :

Avantages des Namespaces :

1. **Isolation Logique** : Les Namespaces permettent d'isoler les ressources et de limiter leur visibilité à l'intérieur d'un périmètre défini.

2. **Gestion des Autorisations** : Ils facilitent la gestion des autorisations en permettant d'attribuer des rôles et des permissions spécifiques à chaque Namespace.

3. **Éviter les Conflits de Noms** : Les Namespaces permettent d'éviter les conflits de noms entre les ressources en les isolant les unes des autres.

Création d'un Namespace :

Exemple YAML :

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: mon-namespace
```

Commande de création :

```
kubectl create namespace mon-namespace
```

Utilisation d'un Namespace :

Exemple de Contexte dans le fichier de configuration kubeconfig :

```yaml
apiVersion: v1
kind: Config
clusters:
- name: mon-cluster
  cluster:
    server: https://kubernetes.example.com
    certificate-authority: /path/to/certificate-authority
contexts:
- name: mon-contexte
  context:
    cluster: mon-cluster
    namespace: mon-namespace
    user: mon-utilisateur
current-context: mon-contexte
```

Ajout de Ressources dans un Namespace :

Exemple YAML pour un déploiement dans un Namespace spécifique :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-deploiement
  namespace: mon-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mon-app
  template:
    metadata:
      labels:
        app: mon-app
    spec:
      containers:
      - name: mon-conteneur
        image: mon-image:tag
```

Manipulation des Ressources dans un Namespace :

Pour afficher les ressources dans un Namespace spécifique :

```
kubectl get pods -n mon-namespace
```

Pour supprimer un Namespace et toutes ses ressources :

```
kubectl delete namespace mon-namespace
```

Exemple concret : Déploiement d'une application multi-environnements

Supposons que vous ayez trois environnements pour votre application : développement, test et production. Vous pouvez créer un Namespace distinct pour chaque environnement afin de séparer les ressources et de mieux contrôler les déploiements.

1. **Création des Namespaces :**

Créez trois Namespaces pour chaque environnement :

```
kubectl create namespace development
kubectl create namespace staging
kubectl create namespace production
```

2. **Déploiement de l'application :**

Déployez votre application dans chaque Namespace en spécifiant le Namespace cible dans la configuration YAML du déploiement. Par exemple, pour le déploiement dans l'environnement de développement :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-app
  namespace: development
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mon-app
  template:
    metadata:
      labels:
        app: mon-app
    spec:
      containers:
      - name: mon-conteneur
        image: mon-image:tag
```

3. **Gestion des Accès et des Autorisations :**

Définissez des rôles et des rôles contraignants pour chaque Namespace afin de limiter l'accès aux ressources. Par exemple, vous pouvez autoriser les développeurs à déployer dans le Namespace de développement, mais pas dans les environnements de test ou de production.

4. **Surveillance et Maintenance :**

Utilisez les commandes Kubernetes pour surveiller et gérer les ressources dans chaque Namespace de manière isolée. Par exemple, pour voir les pods dans l'environnement de test :

```
kubectl get pods -n staging
```

5. **Déploiement Graduel :**

Utilisez les Namespaces pour effectuer des déploiements progressifs, en testant d'abord dans l'environnement de développement, puis en passant à l'environnement de test avant de déployer en production.

En utilisant des Namespaces de cette manière, vous pouvez organiser efficacement vos ressources Kubernetes, isoler les environnements et contrôler l'accès aux ressources en fonction des besoins de votre organisation. Cela facilite également la gestion et la maintenance de vos applications à différentes étapes de leur cycle de vie.

Annexes

J’ai créer un repo sur github pour pouvoir pratiquer les différentes commandes avec un exemple d’un cluster avec trois environnements Dev, Staging et Production sur ce lien : https://github.com/JowaDev/k8s