UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Kubernetes

Sommaire

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction Docker et les containers

Introduction Docker et les containers

1. **Docker et les containers**:

- Docker est une plateforme qui permet de créer, déployer et exécuter des applications dans des conteneurs. Un conteneur est une unité standardisée d'exécution du logiciel qui regroupe le code de l'application et toutes ses dépendances, ce qui permet de garantir qu'elle s'exécute de la même manière, quels que soient l'environnement et le système d'exploitation.

2. **Révolution des containers**:
    1. **Isolation**: Avec Docker, chaque application est isolée dans son propre conteneur, ce qui signifie qu'elle fonctionne de manière constante dans différents environnements.
    2. **Portabilité**: Les conteneurs peuvent être déplacés entre différents environnements (développement, test, production) sans modification.
    3. **Efficacité**: Les conteneurs sont légers et démarrent rapidement, ce qui les rend idéaux pour la mise à l'échelle et le déploiement continu.
    4. **Microservices**: Docker facilite la transition vers une architecture de microservices, où chaque service est empaqueté dans son propre conteneur.

Introduction Docker et les containers

3. **Création et utilisation de containers**:
    1. **Dockerfile**: Un Dockerfile est un script qui contient les instructions pour construire une image Docker. Il spécifie la base OS, les logiciels à installer, les ports à ouvrir, etc.

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y nginx
CMD ["nginx", "-g", "daemon off;"]
```

    2. **Construire une image**:

```
docker build -t monimage:latest .
```

    3. **Exécuter un conteneur**:

```
docker run -d -p 80:80 monimage:latest
```

    4. Gestion des conteneurs:
        - Lister les conteneurs: docker ps
        - Stopper un conteneur: docker stop [CONTAINER_ID]
        - Supprimer un conteneur: docker rm [CONTAINER_ID]

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Kubernetes et l’orchestration de containers

Kubernetes et l'orchestration de containers

1. **Pourquoi un orchestrateur?**

- Dans le monde moderne de la technologie de l'information, les applications sont souvent déployées à grande échelle, fonctionnant sur des centaines voire des milliers de conteneurs. Les défis à cette échelle comprennent:
    1. **Déploiement**: Comment déployer efficacement des milliers de conteneurs?
    2. **Réparation**: Comment réparer les conteneurs qui tombent en panne?
    3. **Mise à l'échelle**: Comment adapter les ressources pour des conteneurs en fonction de la demande?
    4. **Découverte et équilibrage de charge**: Comment les conteneurs peuvent-ils découvrir et communiquer entre eux?
- Un orchestrateur, comme Kubernetes, aide à répondre à ces questions en automatisant le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.

Kubernetes et l'Orchestration de Containers

2. **Avantages de Kubernetes**
    1. **Automatisation**: Kubernetes peut automatiquement déployer, échelonner et équilibrer les charges entre les conteneurs.
    2. **Santé et Auto-réparation**: Il surveille la santé des conteneurs et remplace ceux qui échouent, et peut également automatiser les mises à jour.
    3. **Gestion des ressources**: Il assure que chaque conteneur reçoit les ressources (CPU, mémoire) dont il a besoin.
    4. **Découvrabilité**: Avec son système de service intégré, Kubernetes facilite la découverte et la communication entre les conteneurs.
    5. **Stockage**: Il peut monter et ajouter des systèmes de stockage pour conserver les données persistentes.
    6. **Extensibilité**: Grâce à sa modularité et sa flexibilité, Kubernetes peut s'étendre pour répondre aux besoins les plus complexes.
    7. **Communauté active**: Étant open source, il bénéficie d'une grande et active communauté qui continue à contribuer et à améliorer le système.

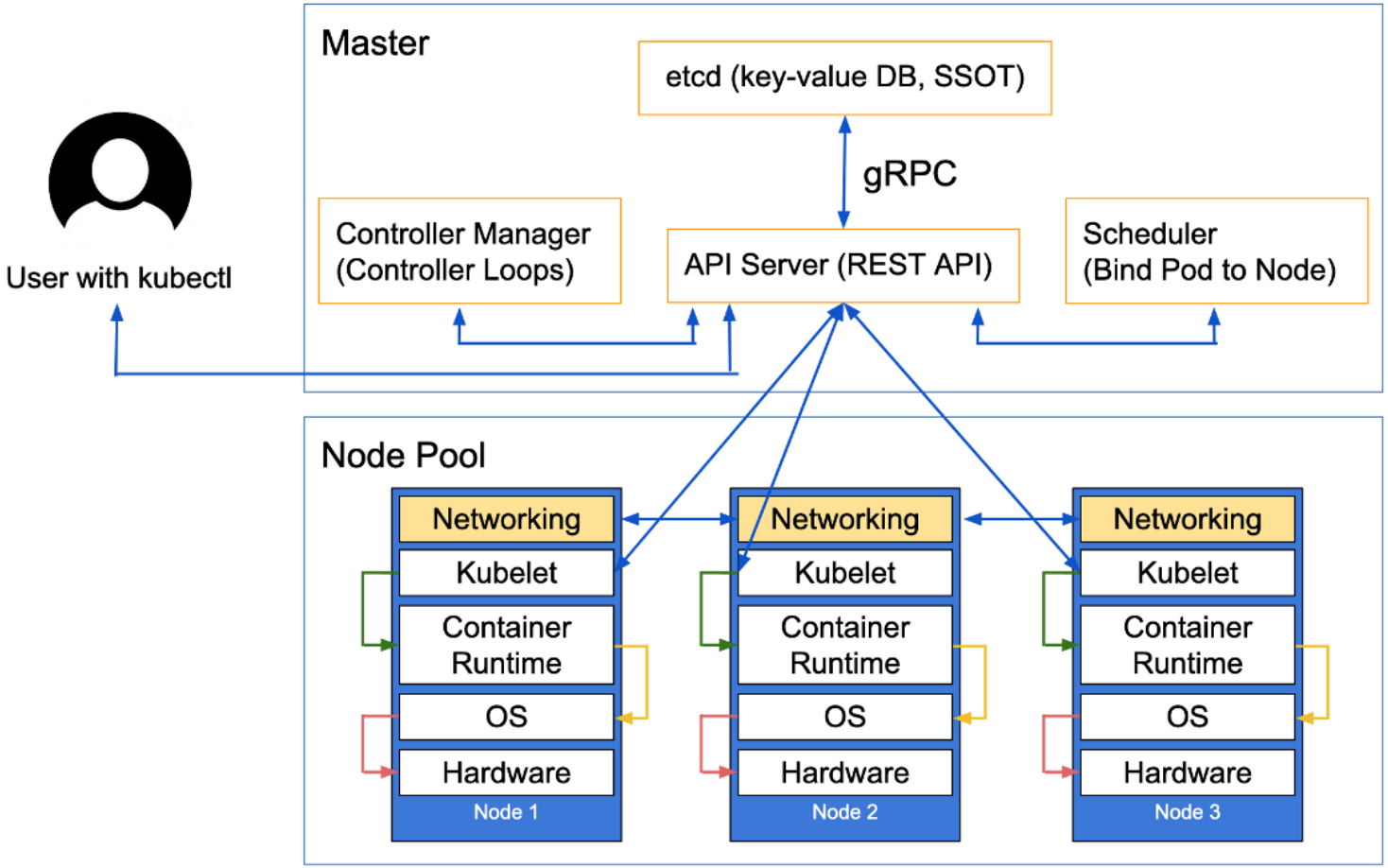
Kubernetes et l'Orchestration de Containers

3. **Mise en place de Kubernetes**
    1. **Prérequis**:
        - Avoir une connaissance de base de Docker.
        - Avoir un cluster de machines (physiques ou virtuelles).
    2. **Installation**:
        - Vous pouvez installer Kubernetes en utilisant des outils comme kubeadm, kind, k3s.
        - Pour une solution clé en main, envisagez des services comme Google Kubernetes Engine (GKE) ou Azure Kubernetes Service (AKS).
    3. **Configuration initiale**:
        - Configurer le réseau pour les pods.
        - Configurer le stockage pour la persistance des données.
4. **Déployer des applications**:
    - Utiliser des fichiers de configuration YAML pour décrire les ressources nécessaires (comme les déploiements, les services et les volumes).

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Architecture de Kubernetes

Architecture de Kubernetes



Architecture de Kubernetes

1. **Principes de fonctionnement**

- Kubernetes est basé sur une architecture de type maître-esclave (ou master-worker). Le maître (ou master) prend des décisions concernant le cluster, telles que la planification, et répond aux demandes d'API, tandis que les esclaves (ou workers) exécutent les conteneurs.

    1. **État désiré vs état actuel**: L'une des idées fondamentales de Kubernetes est la notion d'état désiré. Vous définissez ce que vous souhaitez voir s'exécuter (par exemple, je veux 3 instances de mon application), et Kubernetes s'efforce de s'assurer que la réalité correspond à cet état.
    2. **Autoguérison**: Si un conteneur tombe en panne, Kubernetes le redémarre pour maintenir l'état désiré. De même, si une machine entière tombe en panne, les conteneurs qui s'y exécutaient sont redistribués.

Architecture de Kubernetes

2. **Composants de Kubernetes**
    1. **API Server (serveur API)**: Point d'entrée pour les commandes. Tout dans Kubernetes est traité comme une API.
    2. **etcd**: Base de données clé-valeur utilisée pour tout le stockage de configuration et d'état.
    3. **kubelet**: Agent qui s'exécute sur chaque noeud et s'assure que les conteneurs sont en cours d'exécution dans un pod.
    4. **kube-proxy**: Maintient les règles réseau sur les noeuds pour permettre la communication vers les conteneurs.
    5. **Scheduler (ordonnanceur)**: Décide quel noeud doit exécuter un conteneur.

Architecture de Kubernetes

3. **Masters vs Workers**
    1. **Master (Maître)**:
        - Gère le cluster.
        - Prend des décisions globales (par exemple, la planification).
        - Détecte les événements du cluster (par exemple, un conteneur qui a échoué).
    - *Composants typiques d'un noeud master: API Server, etcd, Scheduler, et autres composants de contrôle.*
    2. **Worker (Esclave)**:
        - Exécute les conteneurs.
        - Rapporte à master.
        - Chaque worker est équipé de Docker (ou une autre solution conteneur), kubelet, et kube-proxy.

Architecture de Kubernetes

4. **Couche réseau**

- La couche réseau dans Kubernetes est cruciale car elle permet la communication entre les conteneurs et aussi entre le cluster et l'extérieur.

    1. **Pod Networking**: Chaque pod reçoit sa propre adresse IP. Les conteneurs au sein d'un pod partagent cette adresse IP et le port, ce qui signifie qu'ils peuvent se communiquer via `localhost`.
    2. **Service Networking**: Expose un ensemble de pods en tant que service. Les services permettent la communication entre les pods et l'extérieur du cluster.
    3. **Network Policies**: Permet de contrôler la communication entre les pods.
    4. **CNI (Container Network Interface)**: Il s'agit d'un ensemble de normes et de plugins qui permettent l'intégration de différentes solutions réseau avec Kubernetes.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Concepts de base

Concepts de base

1. **Concepts de base de Kubernetes**

- Kubernetes introduit un certain nombre de concepts et d'abstractions pour aider les utilisateurs à déployer, gérer et échelonner leurs applications.

2. **Kubernetes API**

- L'API Kubernetes est la colonne vertébrale du système. Elle est utilisée pour créer, mettre à jour et surveiller les diverses ressources disponibles dans Kubernetes.
  - **Versioning**: Kubernetes prend en charge plusieurs versions d'API en même temps pour assurer une compatibilité ascendante.
  - **Ressources**: Les objets dans Kubernetes, tels que les pods, les services, etc., sont tous représentés comme des ressources API.
  - **Opérations CRUD**: L'API Kubernetes permet d'effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur ces ressources.

Concepts de base

3. **Outil `kubectl`**

- `kubectl` est l'outil en ligne de commande pour interagir avec le cluster Kubernetes. Il utilise l'API Kubernetes pour communiquer avec le cluster.
  - **Commandes de base**:
    - `kubectl get`: Affiche une ou plusieurs ressources.
    - `kubectl describe`: Montre les détails d'une ressource spécifique.
    - `kubectl create`: Crée une ressource.
    - `kubectl delete`: Supprime des ressources.
    - `kubectl apply`: Applique une configuration à une ressource.

Concepts de base - Ressources de base

1. **Pod**

Un pod est la plus petite unité déployable dans Kubernetes. Il peut contenir un ou plusieurs conteneurs.

- **Multi-container Pods**: Plusieurs conteneurs fonctionnant ensemble dans un seul pod partagent le même réseau et le même espace de stockage.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
```

Concepts de base - Ressources de base

2. **Deployment**

Gère le déploiement de pods. Il peut créer ou supprimer des pods pour maintenir l'état désiré.

- **Mise à jour et déploiement continu**: Avec les Deployments, vous pouvez mettre à jour vos pods sans interruption de service.

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

Concepts de base - Ressources de base

3. **Label**
    - Les étiquettes (Labels) sont des paires clé-valeur associées aux ressources pour les organiser.
        - **Sélecteurs**: Utilisés pour filtrer les ressources basées sur leurs étiquettes.
4. **Namespace**

- Permet de diviser les ressources d'un cluster entre plusieurs utilisateurs ou projets.
    - **Isolation**: Chaque namespace fournit une portée pour les noms de ressources.

```
kubectl create namespace development
```

Concepts de base - Ressources de base

5. **ConfigMap et Secret**
   - **ConfigMap**: Utilisé pour séparer la configuration du contenu de l'image conteneur.
   - **Secret**: Stocke des informations sensibles, comme les mots de passe.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-config
data:
  game.properties: |
    enemies=aliens
    lives=3
```

```
echo -n 'mypassword' | base64
# Sortie : bXlwYXNzd29yZA==
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  password: bXlwYXNzd29yZA==
```

Concepts de base - Ressources de base

6. **Service**

- Expose un ensemble de pods comme un service réseau. Il fournit un IP stable et un DNS pour les pods.
  - **Types**: ClusterIP (interne), NodePort, LoadBalancer (externe), et ExternalName.

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

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Kubernetes au quotidien

Kubernetes au quotidien

- L'exploitation d'un cluster Kubernetes au quotidien nécessite une bonne maîtrise des outils disponibles et une compréhension des meilleures pratiques.

1. **Utilisation avancée de `kubectl`**
    - **Connexion à l'intérieur d'un pod**: Il peut s'avérer nécessaire d'inspecter l'état interne d'un pod, en particulier lors du débogage.

```
kubectl exec -it <nom-du-pod> -- /bin/bash
```

        - *Cette commande vous donne un shell interactif à l'intérieur du pod, ce qui permet d'examiner directement les fichiers, d'interroger les services locaux, etc.*

2. **Port Forward**
    - Accéder à des applications ou des services en cours d'exécution dans un pod pour le débogage ou la vérification peut être crucial.

```
kubectl port-forward <nom-du-pod> <port-local>:<port-du-pod>
```

Ainsi, vous pouvez accéder localement à un service comme s'il était en cours d'exécution sur votre machine.

Kubernetes au quotidien

3. **Historisation des déploiements**
   - L'une des caractéristiques puissantes de Kubernetes est sa capacité à gérer l'historique des versions d'un déploiement. Chaque fois que vous modifiez un déploiement, Kubernetes conserve un enregistrement de la modification dans ce qu'on appelle une "révision".
   - Pour consulter l'historique des déploiements :

```
kubectl rollout history deployment/<nom-du-déploiement>
```

   - Revenir à une révision précédente
     - Si un déploiement pose problème, il est possible de revenir à une révision précédente :

```
   kubectl rollout undo deployment/<nom-du-déploiement> --to-revision=<numéro-de-révision>
```

Il est essentiel de tester les modifications dans un environnement séparé avant de les déployer en production. Cependant, si un problème survient, l'historisation des déploiements est un filet de sécurité précieux pour restaurer rapidement un état fonctionnel.

Kubernetes au quotidien

4. **Montée en charge**
    - La mise à l'échelle automatique des pods permet de gérer dynamiquement la charge. Vous pouvez définir des métriques, comme l'utilisation du CPU, pour déclencher la mise à l'échelle automatique de vos pods.
5. **Mises à jour d'application**
    - Grâce aux déploiements progressifs, vous pouvez mettre à jour votre application tout en garantissant une disponibilité maximale. En utilisant des stratégies comme "RollingUpdate", vous minimisez les interruptions lors des mises à jour.
6. **Monitoring**
    - Utilisez des solutions comme Prometheus pour collecter des métriques et Grafana pour visualiser ces métriques en temps réel. L'intégration d'alertes peut vous informer des anomalies avant qu'elles ne deviennent critiques.

Kubernetes au quotidien

7. **Troubleshooting**
   - Lorsque des problèmes surviennent, il est essentiel de pouvoir les diagnostiquer rapidement.
     - `kubectl describe`: Donne un aperçu détaillé de la ressource.
     - `kubectl get events`: Montre les événements du cluster, utiles pour comprendre les changements récents ou les erreurs.
8. **Best practices**
   1. **Utilisez des namespaces**: Isoler les applications ou les environnements peut aider à la gestion et à la sécurité.
   2. **Sécurisez votre cluster**: Utilisez RBAC pour contrôler l'accès, et envisagez d'utiliser un réseau de pod chiffré.
   3. **Optimisez les images de conteneur**: Des images plus légères réduisent le temps de démarrage et les vulnérabilités potentielles.
   4. **Mettez en place des sauvegardes**: Assurez-vous de sauvegarder régulièrement les configurations et les données essentielles.
   5. **Mise en place d'une surveillance proactive**: Ne réagissez pas seulement aux problèmes, anticipez-les en surveillant activement les métriques et les logs.

Introduction aux Volumes

Un volume dans Kubernetes est une abstraction qui permet aux conteneurs de stocker et partager des données de manière persistante. Contrairement aux volumes Docker qui sont liés à la durée de vie d'un conteneur, les volumes Kubernetes existent tant que le pod existe.

Types de Volumes

1. **Volumes éphémères**

- **emptyDir** : Créé lorsqu'un pod est assigné à un nœud et dure toute la durée de vie du pod. Idéal pour stocker des données temporaires ou échanger des données entre conteneurs d'un même pod.
- **configMap** et **secret** : Utilisés pour injecter des configurations et des secrets (comme des mots de passe ou des clés API) dans les pods sous forme de fichiers.

2. **Volumes persistants
- **PersistentVolume (PV) et **PersistentVolumeClaim (PVC)** : Les PV sont des ressources de stockage allouées par l'administrateur, et les PVC sont des demandes de stockage par les utilisateurs. Cette séparation permet une gestion flexible et un provisionnement dynamique du stockage.

- **hostPath** : Monte un fichier ou un répertoire du système de fichiers du nœud dans un pod. Il doit être utilisé avec précaution car il lie directement le cycle de vie des données à celui du nœud.

Types de Volumes

3. **Volumes réseau**

- **NFS** (Network File System) : Monte un répertoire distant via NFS. Il permet de partager des données entre différents pods et nœuds.
- **CephFS** et **GlusterFS** : Fournissent des systèmes de fichiers distribués qui peuvent être montés sur plusieurs nœuds et pods.

Création et Utilisation des Volumes

1. **Définition d'un Volume dans un Pod**
   Vous pouvez définir des volumes dans le spec d'un pod. Voici un exemple basique :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: html-volume
  volumes:
    - name: html-volume
      emptyDir: {}
```

Dans cet exemple, un volume `emptyDir` est monté dans le conteneur à `/usr/share/nginx/html`.

Création et Utilisation des Volumes

PersistentVolume :

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
```

PersistentVolumeClaim :

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Création et Utilisation des Volumes

Utilisation du PVC dans un Pod :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: html-volume
  volumes:
    - name: html-volume
      persistentVolumeClaim:
        claimName: example-pvc
```

Volumes Dynamiques

Kubernetes supporte le provisionnement dynamique de volumes, ce qui signifie que les volumes peuvent être créés à la demande lorsque des PVC sont créés, en utilisant des `StorageClass`.

StorageClass :

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

PersistentVolumeClaim utilisant une StorageClass :

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-fast-pvc
spec:
  storageClassName: fast
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Volumes Bonnes Pratiques

- **Séparation des préoccupations** : Utilisez des PVC pour permettre aux développeurs de demander du stockage sans se soucier des détails de l'implémentation.
- **Sécurité** : Utilisez des volumes secrets pour les informations sensibles et gérez les accès avec RBAC.
- **Réplication et Sauvegarde** : Pour les données critiques, utilisez des solutions de stockage qui supportent la réplication et les sauvegardes automatiques.

RBAC

Kubernetes Role-Based Access Control (RBAC) est un mécanisme pour gérer les permissions dans un cluster Kubernetes. Il utilise des rôles et des liaisons de rôles (bindings) pour déterminer si un utilisateur ou un groupe d'utilisateurs peut effectuer une action donnée sur une ressource spécifique dans le cluster.

Concepts clés de RBAC

1. **Role** et **ClusterRole** :
   - Un **Role** définit un ensemble de permissions au niveau du namespace. Il contient des règles qui spécifient quelles actions (verbes) sont autorisées sur quelles ressources dans un namespace spécifique.
   - Un **ClusterRole** est similaire à un Role, mais il est global au niveau du cluster. Il peut être utilisé pour définir des permissions pour des ressources non liées à un namespace (comme des nœuds) ou pour être utilisé dans tous les namespaces.
2. **RoleBinding** et **ClusterRoleBinding** :
   - Un **RoleBinding** associe un Role à des utilisateurs, groupes ou comptes de service dans un namespace spécifique. Cela signifie que les entités mentionnées dans le RoleBinding peuvent effectuer les actions définies dans le Role dans ce namespace.
   - Un **ClusterRoleBinding** associe un ClusterRole à des utilisateurs, groupes ou comptes de service à l'échelle du cluster. Cela signifie que les entités mentionnées dans le ClusterRoleBinding peuvent effectuer les actions définies dans le ClusterRole dans tous les namespaces ou sur des ressources globales.

Fonctionnement de RBAC

1. **Création d'un Role ou ClusterRole** :
   Un administrateur définit un Role ou un ClusterRole en spécifiant les ressources et les actions autorisées.

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

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list", "create", "delete"]
```

Fonctionnement de RBAC

2. **Création d'un RoleBinding ou ClusterRoleBinding** :
   Un RoleBinding ou ClusterRoleBinding associe un Role ou ClusterRole aux utilisateurs, groupes ou comptes de service.

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

Fonctionnement de RBAC

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-binding
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

Fonctionnement de RBAC - Étapes de vérification des permissions

1. **Demande d'accès** :
   Lorsque quelqu'un essaie d'accéder à une ressource ou d'effectuer une action dans Kubernetes, l'API Server reçoit la demande.
2. **Validation des permissions** :
   L'API Server vérifie les bindings existants (RoleBinding et ClusterRoleBinding) pour voir si le demandeur a les permissions nécessaires. Il consulte les Roles et ClusterRoles pour déterminer si l'action demandée sur la ressource est autorisée.
3. **Accès accordé ou refusé** :
   Si un binding approprié existe et que les permissions sont accordées, l'API Server permet l'accès ou l'action. Sinon, l'accès ou l'action est refusé.

Utilisation courante de RBAC

1. **Sécuriser le Kubernetes Dashboard** :
   Vous pouvez créer des comptes de service et des roles spécifiques pour restreindre l'accès au dashboard et s'assurer que seuls les utilisateurs autorisés peuvent y accéder.
2. **Définir des rôles spécifiques pour les développeurs** :
   Par exemple, un rôle qui permet aux développeurs de déployer des applications sans leur donner accès à des ressources sensibles du cluster.
3. **Gestion des permissions des comptes de service** :
   Les comptes de service utilisés par les applications peuvent avoir des rôles spécifiques leur permettant uniquement de lire ou écrire les ressources dont ils ont besoin.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Aller plus loin avec Kubernetes

Aller plus loin avec Kubernetes

- Lorsqu'on commence à approfondir Kubernetes, de nombreuses fonctionnalités avancées peuvent être exploitées pour gérer des cas d'utilisation plus complexes et optimiser l'infrastructure.

1. **Ressources avancées**
    - **StatefulSet**: Les `StatefulSets` gèrent le déploiement et la mise à l'échelle des ensembles d'applications avec état.
    - **Identification stable**: Chaque pod possède une identité unique, stable, sous forme de nom ordinal, comme `web-0`, `web-1`, même après redémarrage.
    - **Stockage stable**: Chaque pod est associé à un volume persistant, permettant la persistance de données malgré les redémarrages.
    - **Déploiement ordonné**: Ils garantissent un ordre de déploiement et de mise à l'échelle. Par exemple, avec une base de données répliquée, vous voudriez garantir qu'une instance est opérationnelle et prête avant d'en démarrer une autre.

Aller plus loin avec Kubernetes

- **DaemonSet**
  - Les `DaemonSets` sont conçus pour garantir qu'une copie de votre pod s'exécute sur chaque nœud du cluster.
    - **Utilisation courante**: Les cas d'utilisation typiques incluent des agents de log, de surveillance ou de sécurité qui doivent exister sur chaque machine.
    - **Sélection de nœuds**: Vous pouvez spécifier sur quels nœuds le DaemonSet doit s'exécuter en utilisant un sélecteur de nœud.
- **Probe**
  - Les `Probes` offrent des mécanismes pour que Kubernetes vérifie la santé des conteneurs dans un pod.
    - **livenessProbe**: Kubernetes utilise cette sonde pour savoir quand redémarrer un conteneur. Si la sonde échoue, le conteneur est tué.
    - **readinessProbe**: Avant de permettre au trafic d'atteindre un nouveau pod, Kubernetes vérifie que les conteneurs du pod sont prêts à traiter les requêtes.
    - **startupProbe**: Utilisée pour savoir quand une application a démarré. Utile pour les conteneurs qui ont besoin de temps pour se mettre en état de démarrage.

Aller plus loin avec Kubernetes

2. **Architecture microservices**

- Avec la montée des microservices, Kubernetes est devenu la plate-forme de choix grâce à sa flexibilité et à sa capacité de mise à l'échelle.
    - **Découplage**: Les microservices séparent votre application en services plus petits et indépendants, ce qui facilite la mise à jour, le dépannage et l'évolutivité de chaque service indépendamment.
    - **Réseau**: Grâce aux fonctionnalités de découvertes de services et de mise en réseau de Kubernetes, les microservices peuvent communiquer entre eux de manière fluide.

Aller plus loin avec Kubernetes

3. **Kubernetes et son écosystème**
    - **Helm**: Helm est souvent appelé le gestionnaire de paquets de Kubernetes.
    - **Charts**: Ce sont des packages de ressources Kubernetes. Avec Helm, les développeurs peuvent créer des charts reproductibles pour leurs applications.
    - **Déploiement simplifié**: Helm facilite le déploiement, la mise à jour et la suppression d'applications sur un cluster Kubernetes.
4. **Prometheus**
    - C'est un outil de surveillance et d'alerte open-source.
        - **Intégration étroite avec Kubernetes**: Prometheus peut automatiquement découvrir des services et des métriques.
        - **Requêtes puissantes**: Son langage de requête permet aux utilisateurs d'inspecter leurs métriques et de générer des alertes.