Formation Kubernetes

Ihab ABADI / UTOPIOS

SOMMAIRE

SOMMAIRE

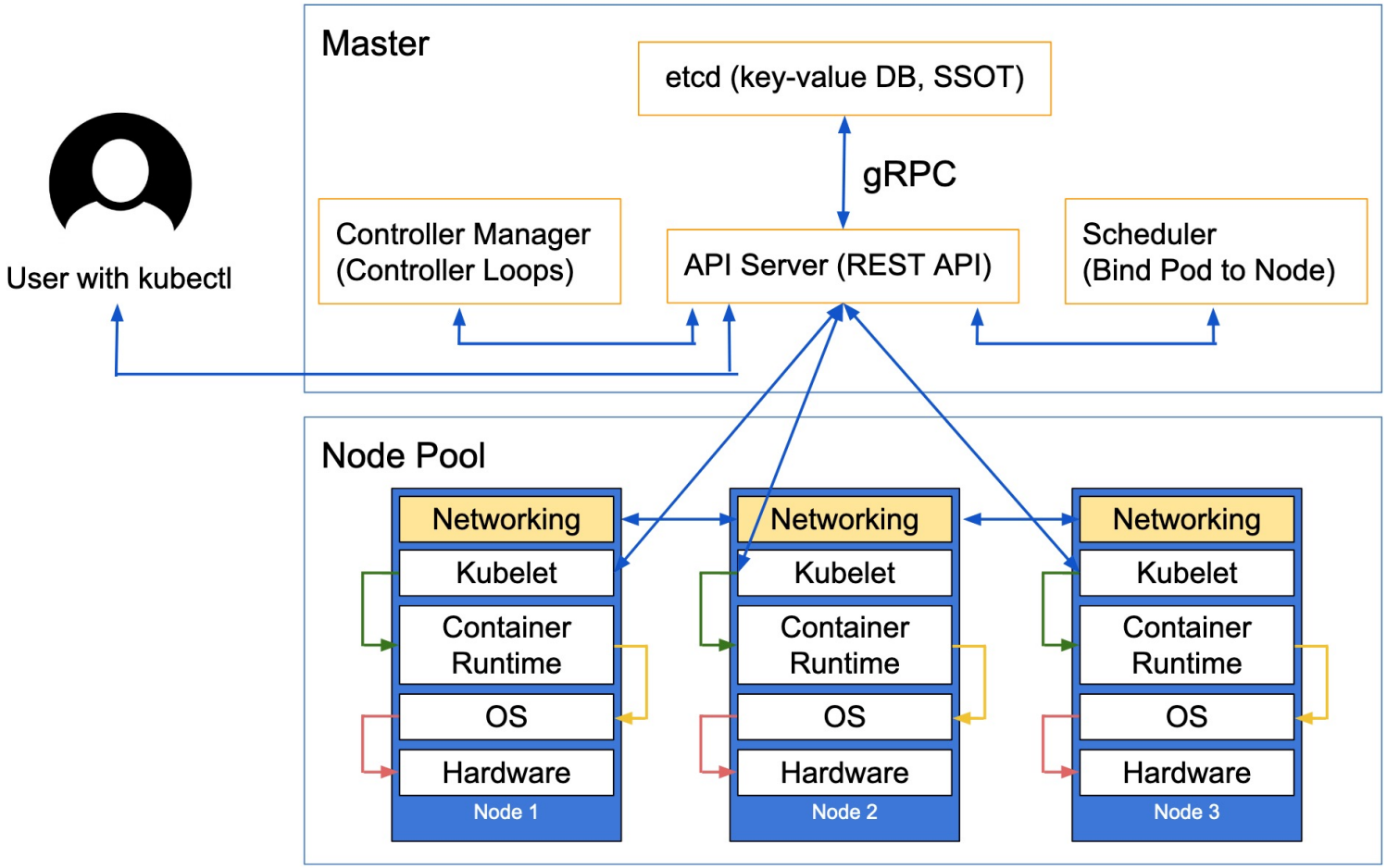
Introduction Kubernetes

- Qu’est-ce que Kubernetes?
- Un système mature et robuste d’orchestration de conteneur Créé par Google, basé sur Borg et Omega, les systèmes qui fonctionnent aujourd’hui chez Google et dont la robustesse est éprouvée depuis plus de 10 ans.
- Google exécute 2 milliards de conteneurs par semaine avec ces systèmes.
- Créé par trois employés de Google initialement pendant l'été 2014.

Fonctionnement de Kubernetes

- Abstraction de l'infrastructure matérielle grâce au concept de "Node"
- Principe: Gérez vos applications comme un troupeau (généricité, opérations de masse) plutôt que comme des animaux de compagnie (chaque opération est personnalisée avec soin et amour pour l'individu).
- Kubernetes est le Linux pour les systèmes distribués:
  - Linux (OS) résout les différences matérielles (avec différents types de CPU, etc.)
  - Kubernetes abstrait les milliers de nœuds d'un cluster et fournit des méthodes industrielles pour gérer les applications.
- L'administrateur décrit et déclare l'"état souhaité", et Kubernetes convertit l'"état courant" en l'"état souhaité".

Architecture de Kubernetes



Les concepts de base de Kubernetes

Pod: Le bloc de base de Kubernetes, atomiquement planifiable, représente une instance unique d'uneapplication dans Kubernetes. Chaque Pod a sa propre IP interne et unique. Les Pods sont mortels.

Deployment: Inclut un modèle de Pod et un ensemble de réplicas. Kubernetes s'assurera que l'étatcourant (nombre de réplicas, modèle de Pod) correspond toujours à l'état souhaité. La mise à jour d'undéploiement déclenche une "rolling update".

Service: Sélectionne les Pods à l'aide d'étiquettes (labels) et fournit un moyen stable et immortel decommuniquer avec votre application en utilisant une IP interne ou un nom DNS.

Namespace: Méthode d'isolation logique, la plupart des objets kubernetes ont une portée relative à unseul Namespace. Vous pouvez y regrouper des applications et leur appliquer vos différentes stratégiesde gestion.

Quelque fonctionnalités de Kubernetes

Self-healing: redémarre les conteneurs qui échouent, remplace et re-planifie les conteneurs lorsque les nœuds meurent, tue les conteneurs qui ne répondent pas au contrôle d'intégrité défini par l'utilisateur et les publie auprès des clients seulement lorsqu'il sont prêts

Automatic binpacking: place automatiquement les conteneurs en fonction de leurs besoins en ressources et d'autres contraintes, sans sacrifier la disponibilité. Co-localisez les charges de travail critiques et "best-effort" afin de maximiser l'utilisation des ressources

Horizontal scaling and autoscaling: Faites passez votre application à l'échelle à l'aide d'une simple commande, d'une interface graphique, ou automatiquement en fonction de l'utilisation du processeur ou de métriques personnalisées.

Automated rollouts and rollbacks: Kubernetes déploie progressivement les modifications apportées à votre application ou à sa configuration, tout en surveillant l'intégrité de l'application afin de s'assurer qu'elle ne tue pas toutes vos instances en même temps. En cas de problème, Kubernetes annulera les changements pour vous.

Quelque fonctionnalités de Kubernetes

Service Discovery and Load Balancing: inutile de modifier votre application pour utiliser un mécanisme de découverte de service tiers. Kubernetes donne aux conteneurs leurs propres adresses IP et un seul nom DNS pour un ensemble deconteneurs, et peut équilibrer la charge entre eux.

Secret and configuration management: Déployez et mettez à jour les secrets et la configuration de l'application sans reconstruire votre image et sans exposer les secrets les fichiers de configuration de votre déploiement.

Storage Orchestration: monte automatiquement le système de stockage de votre choix, qu'il s'agisse du stockage local, d'un fournisseur de cloud public tel que GCP ou AWS, ou d'un système de stockage réseau tel que NFS, iSCSI, Gluster,Ceph, Cinder ou Flocker

Batch Execution: En plus des services, Kubernetes peut gérer vos batch et votre CI, en remplaçant les conteneurs qui échouent.

Démarrage Avec Kubernetes

Play with Kubernetes: fonctionne instantanément dans votre navigateur
Créez un cluster sur votre portable ou votre poste de travail avec minikube
Créez un cluster en 2 lignes de commande avec kubeadm
Créer un cluster de production sur AWS avec kops
Créer un cluster de production sur GCE avec GKE (Google Container Engine)
kubicorn, Juju: d'autres méthodes visant à simplifier l'installation

Gestion des pods dans Kubernetes

Un Pod est un groupe d'un ou plusieurs conteneurs, avec un stockage et un réseau partagé.
Par conséquence, ces conteneurs communiquent plus efficacement entre eux et assurent une localisation de la donnée.
Il existe deux types de Pods :
Pod à conteneur simple.
Pod à conteneur multiple.

Gestion des pods dans Kubernetes

Création d’un pod:
Kubernetes utilise un mécanisme de templates en yaml pour décrire les différents types objets
Chaque objet kubernetes doit définir les paramètre suivants :
apiVersion : la version de l'API Kubernetes que vous utilisez pour créer cet objet .
kind : le type d'objet k8s que vous comptez créer.
metadata : données de type clé valeur permettant d’identifier de manière unique l’objet.
spec : contient la spécification avec des champs imbriqués propres à chaque objet k8S.
Le format est donc différent pour chaque objet Kubernetes

Gestion des pods dans Kubernetes

Création d'un pod à conteneur unique:

Pour créer notre premier pod, on peut utiliser la description suivante

On peut utiliser la commande

kubectl create -f nginx-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    type: web
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Gestion des pods dans Kubernetes

Pour tester notre pod il faut se connecter au nœud sur le quel est hébergé notre pod.
l'ip du Pod n'est accessible qu'à partir du notre nœud sur lequel il est hébergé

Gestion des pods dans Kubernetes

Création d'un pod à conteneur multiple:

Pour créer notre premier pod à conteneur multiple, on peut utiliser la description suivante

On peut utiliser la commande

kubectl create -f nginx-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: multic
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: alpine
    image: alpine
    volumeMounts:
    - name: html
      mountPath: /html
    command: ["/bin/sh", "-c"]
    args:
      - date >> /html/index.html;
        while true; do
          sleep 1;
        done

  volumes:
  - name: html
    hostPath:
      path: /data
      type: DirectoryOrCreate
```

Exercice Pod

Créer un pod avec deux conteneurs qui peuvent communiquer entre eux.
1er conteneur : Application web
2ème conteneur : Base de données relationnelles.

Gestion des ReplicaSets

Les ReplicaSets permettent de vérifier que le nombre de Pods souhaités est bien disponible.
La création et manipulation des ReplicaSets se fait à l'aide de description en yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
  labels:
    app: web-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-pod
  template:
    metadata:
      labels:
        app: web-pod
    spec:
      containers:
      - name: nginx
        image: nginx
```

Gestion des ReplicaSets

On peut scaler les pods à l'aide des ReplicaSets soit:

- A l'aide de la commande scale
- A l'aide de la description du ReplicaSet et une mise à à l'aide de la commande apply

Exercice ReplicaSets

- Créer un replicaSet de 3 pods du Pod de l'exercice 1
- Scaler le nombre de pods en l'augmentant à 5
- Diminuer le nombre de pods à 2

Gestion des deployments

- Un deployment est une abstraction qui permet à la fois de définir les replicaSets et les Pods
- Un objet de type deployment peut être créer, à la fois, à l'aide d'un document de description ou en cli

Gestion des deployments

- Pour créer un deployment à l'aide du cli, on utilise la commande:
- Kubectl run
- Pour créer un deployment à l'aide d'un document de description on utilise :
- La version de kubernetes apps/v1
- L'attribut kind Deployment
- Exemple

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

Gestion des deployments

- Le mécanisme des deployments nous offre la possibilité de modifier la configuration de notre spécification à la volée à l'aide de la commande :
  - Kubectl set
- Le mécanisme nous permet également de naviguer dans l'historique des versions et de revenir en arrière à l'aide des commandes
  - Kubectl rollout
- Le mécanisme nous permet également de procéder à une mise à l'échelle soit manuelle ou automatique à l'aide de la commande :
  - Kubectl scale

Gestion des services

- Kubernetes nous permets de gérer l'accès au différents pods à leurs propres adresses IP
- Il permet également d'équilibrer les charges entre eux.
- Il existe 4 types de services :
- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Gestion des services

- Pour créer un objet de type service, on peut utiliser :
- Un document de description
- La commande kubectl expose
- Pour créer un un service à l'aide d'un document de description on utilise:
- La version de L'api Kubernetes v1
- Un kind service
- Exemple de service NodePort
- Exemple de service ClusterIP

Gestion des services - Exercice

- Exposer via un service un pod répliqué 5, chaque pod contient une application web
- Notre pod application web communiquera avec un pod qui contient un seul replicas et un container de base de données

Gestion des clusters virtuels

- Kubernetes offre la possibilité de gérer des clusters virtuels à l'aide des namespaces
- Un namespace est un mécanisme qui permet de limiter et de diviser les ressources physiques d'un cluster entre plusieurs utilisateurs.
- Kubernetes limite les ressources en utilisant un système de quotas
- Chaque ressource créée se trouve dans un namespace
- Kubernetes possède un namespace par défaut

Gestion des clusters virtuels

- Pour créer un namespace, on utilise L'api Kubernetes à l'aide :
- Du CLI avec la commande kubectl create namespace <nom_namespace>
- Avec un fichier de description avec comme type d'objet namespace
- Exemple

Gestion des clusters virtuels

- Pour associer une ressource à un namespace, on utilise:
- L’argument namespace à l’excution de nos ressources.
- L’argument namespace dans l’objet metadata dans la description des ressources.
- Exemple

Gestion des clusters virtuels

- Kubernetes offre la possibilité de restreindre les ressources physiques au ressources d'un namespaces
- Kubernetes utilise des objets de types ResourceQuota Object
- Un objet ResourceQuota est une ressource Kubernetes qu'on peut créer à l'aide d'un document de description
- Les limitations peuvent être sur les capacités d'utilisation de la mémoire et CPU.
- Un ResourceQuota est définit pour la totalité du namespace.

Exercice

- Créer un namespace dédié aux pods de notre application web.
- Limiter les quotas de ressources à 1GO de mémoire 500MO

Cycle de vie des Pods

- Un pod passe par plusieurs états au cours de son existence.
- Pending
- Running
- Succeded
- Failed
- Unknown

Cycle de vie des Pods

- Un Kubelet exécute régulièrement des sondes pour déterminer l'état d'un pod.
- Pour interagir avec les sondes, Kublet exécute des Handlers.
- Il existe 3 types de Handlers:
  - ExecAction: Un handler qui permet d'exécuter une commande à l'intérieur d'un conteneur, le diagnostique est considéré Ok si la commande renvoie 0.
  - HTTPGetAction: Un handler qui permet d'exécuter une requête HTTP sur l'adresse IP du conteneur, le diagnostique est considéré Ok si la requête envoie un code retour entre 200 et 400
  - TCPSockerAction: Un handler qui permet d'exécuter un contrôle TCP sur l'adresse IP d'un conteneur, le diagnostique est considéré Ok si le port est ouvert.
- En fonction du code retour du handlers l'exécution de la sonde peut avoir 3 résultats:
- Success
- Failure
- Unknown

Cycle de vie des Pods

- Il existe 3 types de Sondes :
- livenessProbe : Une sonde qui indique si le conteneur est en cours d'exécution, si la sonde échoue, le conteneur est tué par Kubelet et soumis à la stratégie de redémarrage, état par défaut Success
- readinessProbe: Une sonde qui indique si le conteneur est prêt à recevoir des requêtes, si la sonde échoue l'adresse IP du conteneur est enlevé de la totalité des services.
- startupProbe: Une sonde qui indique si l'application à l'intérieur du conteneur a bien démarrée, si une startupProbe est fournie, les autres sondes sont désactivées en attendant le succès de la startupProbe, si la sonde échoue, le conteneur est tué par Kubelet et soumis à la stratégie de redémarrage, état par défaut Success.

Cycle de vie des Pods

- Exemple de livenessProbe avec un ExecAction handler et l'image k8s.gcr.io/busybox
- Exemple de livenessProbe avec un HttpGetAction handler et l'image k8s.gcr.io/liveness
- Exemple de livenessProbe avec un TCPSocketAction handler et l'image k8s.gcr.io/goproxy:0.1

Cycle de vie des Pods (Exercice)

- Ajouter Une stratégie LivenessProbe, ReadinessProbe, StartupProbe à notre application Web

Gestion de la persistance dans Kubernetes

- La persistance d'un conteneur est éphémère.
- Ce principe reste valable dans Kubernetes.
- Pour sauvegarder et partager des fichiers, on utilise le mécanisme des volumes, à l'image du mécanisme de volume des conteneurs Docker.
- A la différence des volumes Docker, Kubernetes possède une multitude de types de volumes qui peuvent être utiliser en même temps.
- Par défaut les volumes ont une durée de vie (celle du Pod).

Gestion de la persistance dans Kubernetes (HostPath)

- HostPath est un volume qui permet de monter un fichier ou un répertoire du système du fichiers du nœud parent du pod.
- HostPath propose une multitude de types de path:
- DirectoryOrCreate
- Directory
- FileOrCreate

Gestion de la persistance dans Kubernetes (Volume persistant)

- Le couplage entre cycle de vie de volume et pod, rend la persistance éphémère.
- Pour répondre à cette problèmatique, on utilise des ressources de type PersistentVolume.
- Un PersistentVolume a un cycle de vie indépendant de tout pod.
- Chaque Pod a la possibilité de réclamer des ressources à partir du PersistentVolume à l'aide d'un objet de type PersistentVolumeClaim

Gestion de la persistance dans Kubernetes (Exercice)

- Ajouter un volume persistent à notre cluster Kubernetes de notre application web.
- Ajouter un claim pour le volume persistent.
- Monter un volume pour notre conteneur de base de données.

Gestion des variables d'environnement

- La déclaration des variables d'environnement dans un conteneur peut se faire de plusieurs façons.
- Variable d'environnement basique avec le champ env (clés name et value)
- À l'aide du volume ConfigMap
- À l'aide du volume secret

Gestion des variables d'environnement (ConfigMap)

- ConfigMap est un volume qui permet de partager des variables d'environnement dans plusieurs pods
- ConfigMap peut être créer :
- Depuis un fichier
- Comme une description de ressource Kubernetes.

Gestion des variables d’environnement (Secret)

- Secret est un volume qui permet de stocker des données sensibles
- Secret peut être créer :
- Depuis un fichier
- Comme une description de ressource Kubernetes.

Gestion des variables d’environnement (Exercice)

- Créer des variables d’environnement qui permettent de propager sur plusieurs pods, les accès à une base de données, ainsi une clé de cryptage.

Gestion des Scheduler

- Le scheduler est le composant qui permet à Kubernetes de planifier le déploiement des nœuds.
- WorkFlow du Scheduler
- Création du pod.
- Le scheduler constate qu'il n'y a pas de nœud attribué au nouveau.
- Le scheduler assigne un nœud au pod.

Gestion des Scheduler

- Nous avons la possibilité de spécifier au scheduler le nœud de déploiement
- Par nom
- Par type
- Par affinité
- Par rejet et tolerations

Gestion des Scheduler (DaemonSet)

- Un objet de type daemonSet et un objet qui permet de garantir l'existence d'un pod sur la totalité des nœuds

Gestion des Jobs

- La fonction principale d’un Job est de créer un ou plusieurs pods et de suivre l’état des pods.
- Ils garantissent que le nombre spécifié de pods est terminé avec succès.
- Lorsqu'un nombre spécifié d'exécutions réussies de pods est terminé, la tâche est considérée comme terminée.
- Exemple de création

Gestion des Jobs (Scheduler)

- À l’aide d’un planificateur, nous pouvons programmé l’exécution d’un job
- Exemple

Gestion des Jobs (Scheduler) Exercice

- Créer un job qui s'exécute 2 fois par jours, (le conteneur écrira la date et le heure dans un fichier, sauvegardé dans un volume persistant)

Gestion des NetworkPolicy

- NetWorkPolicy est un mécanisme de contrôle d'accès réseau inter-Pod
- NetWorkPolicy permet de mettre en place une des restrictions sur les communications entrante et sortante entre Pod.
- Le trafic entrant par le Pod est géré par la propriété Ingress
- Le trafic sortant d'un Pod est géré par la propriété Egress
- Démo

Gestion des NetworkPolicy (Exercice)

- Créer une NetWorkPolicy, pour restreindre l'accès entrant à un pod contenant une base de données mysql uniquement au pod contenants le rôle client-mysql