Cours de virtualisation avancée: *Kubernetes*, introduction

1 Introduction, les *clusters*

- Grappe de serveur
- Groupe de machines fonctionnant ensemble

2 *Kubernetes*

kubernetes

2.1 C'est quoi ?

- Ancien mot grec signifiant "timonier" (marin qui tient la barre)
- Souvent abrégé *"K8S"*
- Système *open-source* pour le **déploiement**, la **mise à l'échelle** et **gestion** de conteneurs
- Configuration via des fichiers *YAML*

2.2 Origine

- Lancé par *Google* en 2014
- *Google* s'associe à la *Linux Foundation* pour créer la *CNCF* en utilisant *K8S* comme projet de départ

CNCF: *Cloud Native Computer Foundation*

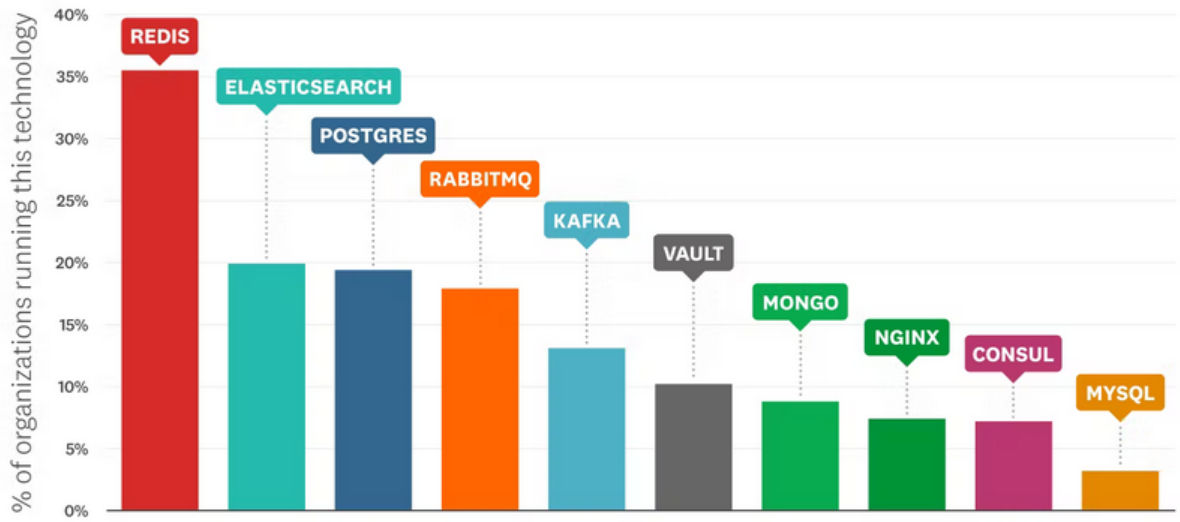
2.3 Quelques statistiques

- 66% : c'est le nombre d'entreprises qui utilisent *K8S* en prod en 2023 [7]
- Entre 2018 et 2020 : *K8S* top 10 projets les plus actifs *Github*
- Le nombre de conteneurs en prod dans le monde est inestimable mais *Google* estime en lancer **2 milliards par semaine**

3 Le monde des conteneurs

`./images/containers.webp`

3.1 Des applications souvent présentes



Redis	BDD clef-valeur
ElasticSearch	NOSQL / moteur de recherche
Postgresql	base de données SQL
RabbitMQ	agent de message
Kafka	agent de message / événements
Vault	BDD sécurisés
Mongo	BDD NOSQL
Nginx	serveur web
Consul	gestion de réseau
MySql	BDD SQL

3.2 Plusieurs orchestrateurs

3.2.1 *Kubernetes*

Le plus connu et utilisé, objet de ce cours

3.2.2 *Nomad*

Proposé par *Hashicorp*, solution d'orchestration hybride entre conteneurs, machines virtuelles et machines physiques.

3.2.3 *Docker Swarm*

Proposé par *Docker*, solution plus simple mais qui ne permet pas une utilisation aussi avancée.

3.2.4 *Rancher*

Une surcouche à *Kubernetes* qui permet de simplifier son utilisation à grande échelle.

3.3 Solutions à louer

4 Pourquoi un orchestrateur

`./images/why.gif`

- Déploiement et mise à l'échelle automatique
- Haute disponibilité / réplication
- Optimisation des ressources
- Supervision et journalisation

4.1 Déploiement et mise à l'échelle automatique

- Déploiement via de la *CI*
- Mises à jours automatiques des conteneurs
- Allocation automatique des ressources

4.2 Haute disponibilité

- Réplication des applications
  - Pour les performances
  - Pour la sécurité
- Tolérance aux pannes de machines

4.3 Optimisation des ressources

- Utilisation de conteneurs
- Choix des cibles de déploiement en fonction de leur usage et occupation
- Dé-allocation des ressources en cas de baisse de charge

4.4 Supervision et journalisation

- Gestion unifiée des journaux
- Outils automatisés

5 *Kubernetes*

5.1 Terminologie et concepts

Namespace · Un "espace de noms"

· Permet de regrouper les ressources pour s'y retrouver et simplifier la gestion

Pod doc

· La plus petite unité d'exécution

· C'est dans ces éléments que sont présents les conteneurs

· Réseau local entre les conteneurs d'un même *pod*

Service doc

· Une manière d'exposer une application s'exécutant sur un ensemble de *pods* en tant que service réseau

· Possède une adresse *IP* et un nom *DNS*

· Permet l'équilibration de la charge

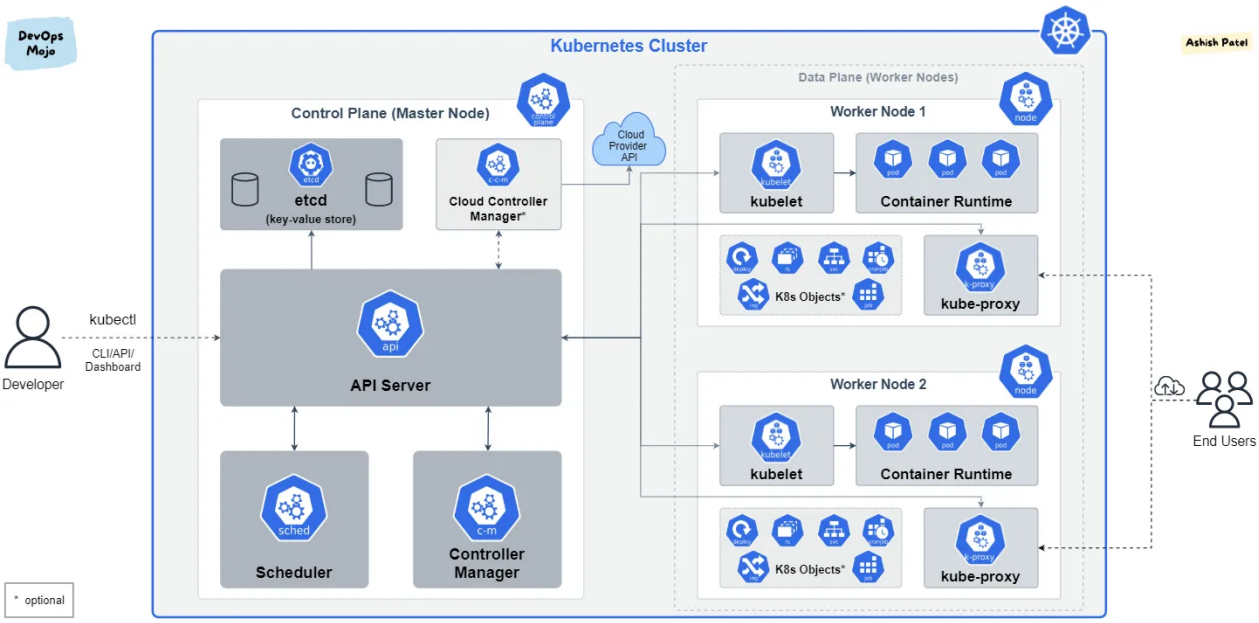
Volume doc

· Permet le stockage des fichiers d'un *pod* (sinon fichiers perdu lors de la destruction du *pod*)

· Permet de partager des fichiers entre des conteneurs au sein d'un *pod*

Il reste encore une multitudes de concepts et de termes propres à *Kubernetes*. Je vous invite donc à consulter cette page: https://kubernetes.io/fr/docs/concepts/.

5.2 Architecture



- Composé de plusieurs nœuds
- Au moins 1 nœud de gestion et 1 nœud de travail

5.2.1 Nœud de gestion

- Aussi appelé *"control plane"* dans la terminologie *K8S*
- Composant principal
- Gestion de la charge de travail
- Services qui contrôlent le *cluster*

etcd stocke les éléments de configuration

API server reçoit et traite les requêtes de gestion. Vérifie en permanence que l'état du *cluster* correspond aux configurations stockées dans `etcd`

Scheduler se charge de la répartition de la charge

Controller manager applique les actions requises à la gestion du *cluster*

5.2.2 Nœud de travail

- C'est sur ces nœuds que les conteneurs sont déployés
- Cela peut être des machines physiques ou virtuelles

kubelet* ou *node manager
- Se charge de la gestion des *pods* sur le nœud
- Reçoit ses ordres du *control plane*
- Informe le *control plane* de l'état du nœud

kube-proxy
- Se charge de la gestion du réseau

container runtime L'environnement d'exécution des conteneurs, souvent *Docker*

6 Prise en main

- Les interactions avec *K8S* se font via la commande `kubectl`.
- Quelques interactions sont aussi possible via le *dashboard* mais ce dernier est limité et ne sera pas abordé pendant ce cours.

6.1 Appliquer une configuration

Cela se fait avec la commande

```
kubectl apply -f <fichier.yaml>
```

6.2 Lister des ressources

Syntaxe:

```
kubectl get <ressource>
```

```
 kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
profiles-app   3/3     Running   0          22s

 kubectl get services
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)
profiles-app-svc   LoadBalancer   10.97.188.25   <pending>     8000:30487/TCP,8001:32451/TC
```

6.3 Lire les journaux d'un *pod*

Syntaxe:

```
kubectl logs <nom ressource>
```

```
 kubectl logs profiles-app
...
```

6.4 Supprimer une ressource

Syntaxe:

```
kubectl delete <type ressource> <nom ressource>
ou
kubectl delete -f <fichier.yaml>
```

```
 kubectl delete pods profiles-app
pod "profiles-app" deleted
```

6.5 Décrire une ressource

Syntaxe:

```
kubectl describe <type ressource> <nom ressource>
```

```
 kubectl -n profiles-app describe pod profiles-app
Name:             profiles-app
Namespace:        profiles-app
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
```

```
Start Time:       Thu, 09 May 2024 17:35:54 +0200
Labels:           run=profiles-app
Annotations:      <none>
Status:           Running
IP:               10.244.0.7
...
```

6.6 Créer des ressources

6.6.1 Le *YAML*

Dans *K8S* les ressources se créent depuis des fichiers *YAML* décrivant la ressource en question et ses paramètres.
Le language *YAML* est un language de sérialisation semblale à *JSON*.
L'extension de fichier est `.yaml` ou `.yml`.
Exemple de fichier *YAML*:

```yaml
---
clef: valeur
liste_de_nombres:
  - 1
  - 2
  - 3
liste_de_clefs_valeurs:
- titre: blade_runner
  note: 10
```

6.7 Types de ressources et exemples

6.7.1 *Namespace*

doc

```yaml
---
apiVersion: v1          # nécessaire pour que K8S comprenne
kind: Namespace         # le type de ressource
metadata:               # les informations de la ressource
  name: profiles-app    # ici, son nom
```

6.7.2 *Pod*

doc

```yaml
---
apiVersion: v1
kind: Pod                       # ici le type est "pod"
metadata:
  namespace: profiles-app-ns    # on choisi le namespace
  name: profiles-app            # le nom du pod
spec:                           # puis ses spécifications
  containers:                   # il contient des conteneurs
  - name: database              # nom du conteneur
    image: image:latest         # l'image à utiliser
    env:                        # variables d'environnement
    - name: COULEUR             # avec un nom
      value: "rouge"            # et une valeur
```

6.7.3 *Deployment*

doc

```yaml
apiVersion: apps/v1
kind: Deployment          # un nouveau type, "Deployment"
metadata:
  name: nginx-deployment # on lui met un nom
  labels:                # et des labels
    app: nginx
spec:
  replicas: 3            # le nombre de réplications voulues
  selector:
    matchLabels:
      app: nginx
  template:              # champs qui seront appliqués aux pods
    metadata:
      labels:
        app: nginx
    spec:
      containers:        # on spécifie les conteneurs
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

6.7.4 *Service*

doc

```yaml
---
apiVersion: v1
kind: Service                # ici le type est "service"
metadata:
  namespace: profiles-app-ns # on choisi le namespace
  name: my-service           # on choisi un nom pour le service
spec:
  selector:                  # règles pour sélectionner le pod
    app.kubernetes.io/name: profiles-app # selection par nom
  ports:                     # on choisi les ports à exposer
    - protocol: TCP          # le protocole
      port: 80               # le port *CIBLE*
      targetPort: 9376       # le port *DU POD*
```

7 Aller plus loin

- [1] https://www.cncf.io/reports/cncf-annual-survey-2023/
- *Services*: `https://kubernetes.io/fr/docs/concepts/services-networking/service/`
- *Pods*: `https://kubernetes.io/docs/concepts/workloads/pods/`
- *Deployments*: `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`