

# DOCKER & KUBERNETES

---

Dr. Mohamed TALHA

DevOps, Java / JEE & Big Data Technical Leader

# Plan

- Docker, une vue d'ensemble
  - Docker, qu'est-ce que c'est ?
  - Conteneurisation versus Virtualisation
  - Image Docker & Conteneur Docker
  - Dockerfile
  - Ateliers : prise en main de Docker (installation Docker, création d'un Dockerfile, génération, exécution et publication d'une image Docker sur Docker Hub)
- Kubernetes (K8s)
  - K8s, qu'est-ce que c'est ?
  - Atelier : mettre en place un Cluster K8s avec minikube & kubectl
  - Objets K8s
  - Workloads K8s
  - Cluster K8s
  - Atelier : déployer une API REST Spring Boot sur un Cluster K8s

# DOCKER

---

- Docker, qu'est-ce que c'est ?
- Conteneurisation versus Virtualisation
- Image Docker & Conteneur Docker
- Dockerfile
- Ateliers Docker

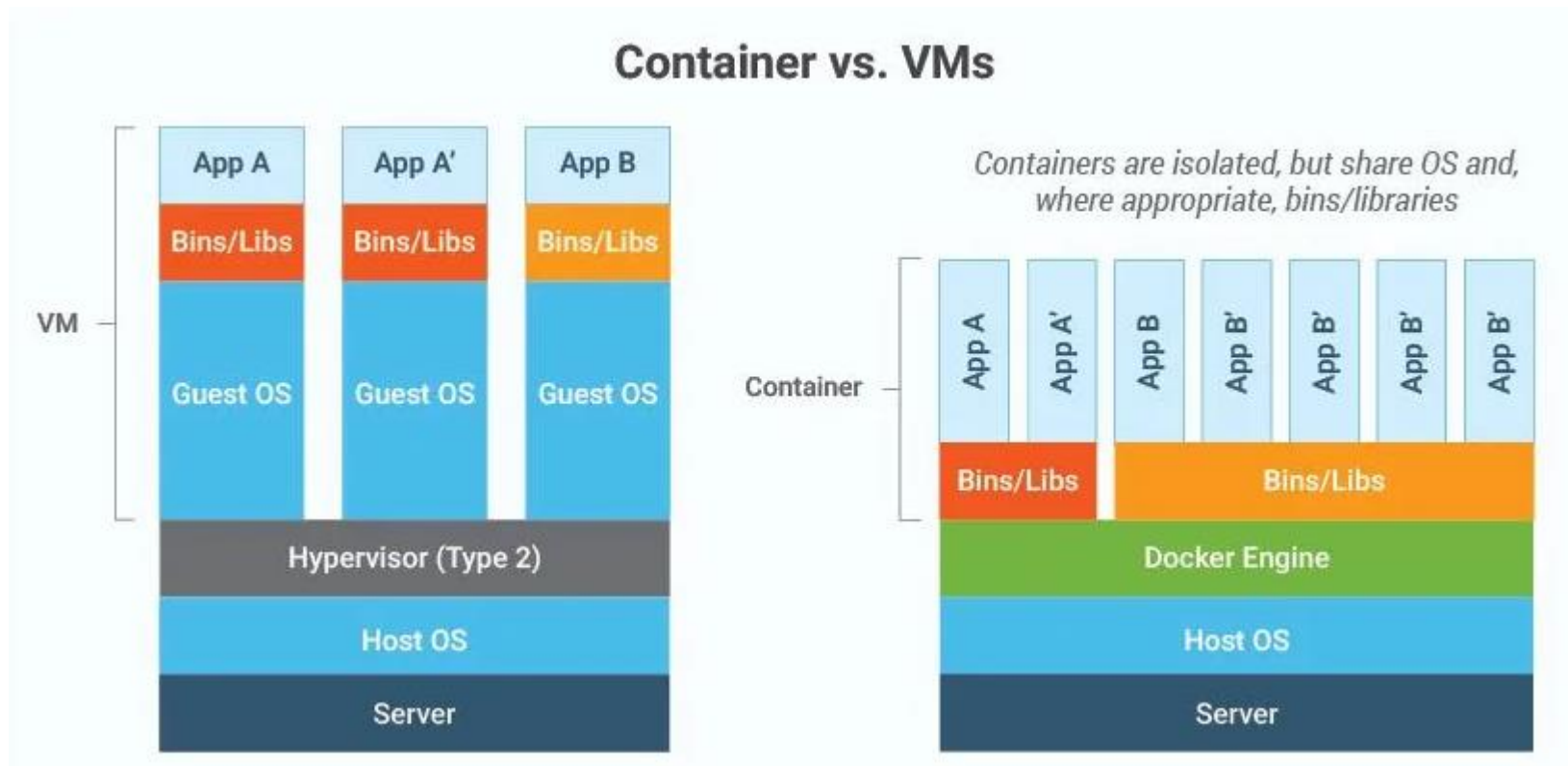
# Docker, qu'est-ce que c'est ?

- **Docker** est un logiciel open source permettant d'automatiser le déploiement des applications sous formes de packages dans des **conteneurs** virtuels.



- les **conteneurs Docker** ne contiennent que les applications et leurs dépendances (i.e. le binaire avec toutes ses librairies) et partagent tous le même système d'exploitation de l'infra.

# Conteneurisation vs Virtualisation



Le **Docker Engine** est l'application à installer sur la machine hôte pour créer et gérer des conteneurs Docker. Il s'agit du moteur du système Docker qui regroupe et relie les différents composants entre eux.

# Image docker

- Une **Image Docker** permet d'installer et lancer une application dans un conteneur Docker.
- Une image Docker est un modèle (en lecture seule) composé de plusieurs couches ; ces couches contiennent l'application que l'on souhaite déployer ainsi que les bibliothèques et les fichiers binaires requis.



- Un **Conteneur Docker** n'est donc qu'une instance d'une **Image Docker**.

# Dockerfile

- Un **Dockerfile** est un fichier texte qui définit une suite de commandes UNIX qui s'exécutent les unes après les autres pour créer une Image Docker.
- Un Dockerfile précise le système d'exploitation sur lequel sera basé le conteneur, les variables d'environnement, les emplacements de fichiers, ports réseau, etc.
- Docker dispose de deux utilitaires :
  - **build** qui permet de créer une Image Docker à partir d'un Dockerfile
  - **run** qui permet de lancer le conteneur qui est une instance de l'image Docker

# Les commandes Dockerfile (1/3)

- **FROM** : elle définit l'image de base qui sera utilisée par les instructions suivantes → chaque image est forcément basée sur une autre image. Une des images les plus utilisées est « **Alpine Linux** » qui est une distribution légère qui existe depuis 2006 et réputée pour sa sécurité. Le choix de la bonne image de base est primordial pour une bonne performance.
- **LABEL** : elle ajoute des métadonnées à l'image avec un système de clés-valeurs ; elle permet par exemple d'indiquer l'auteur du Dockerfile ou la version du fichier.
- **ENV** : elle permet de définir des variables d'environnements utilisables dans le Dockerfile et le conteneur.
- **ARG** : elle permet de définir des variables temporaires qu'on peut utiliser dans un Dockerfile.



# Les commandes Dockerfile (2/3)

- **RUN** : elle exécute des commandes Linux ou Windows lors de la création de l'image.
- **COPY** : elle permet de copier des fichiers depuis la machine locale vers le conteneur Docker.
- **ADD** : elle permet de copier des fichiers depuis la machine locale vers le conteneur Docker (tout comme COPY) mais prend en charge des liens ou des archives.
- **ENTRYPOINT** : comme son nom l'indique, c'est le point d'entrée du conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur.
- **CMD** : elle spécifie les arguments qui seront envoyés à l'ENTRYPOINT.

# Les commandes Dockerfile (3/3)

- **WORKDIR** : elle définit le répertoire de travail à utiliser pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.
- **EXPOSE** : elle expose un port.
- **VOLUMES** : elle crée un point de montage qui permettra de persister les données.
- **USER** : elle désigne quel est l'utilisateur qui lancera les prochaines instructions RUN, CMD ou ENTRYPOINT (par défaut c'est l'utilisateur root).
- et bien d'autres... ➔ <https://docs.docker.com/reference/dockerfile/>

# Exemple de Dockerfile

```
# ----- DÉBUT COUCHE OS -----
FROM debian:stable-slim
# ----- FIN COUCHE OS -----

# MÉTADONNÉES DE L'IMAGE
LABEL version="1.0" maintainer="mohamed.talha"

# VARIABLES TEMPORAIRES
ARG DOCUMENTROOT="/var/www/html"

# ----- DÉBUT COUCHE APACHE -----
RUN apt-get update -y && apt-get install apache2
# ----- FIN COUCHE APACHE -----

# ----- DÉBUT COUCHE MYSQL -----
RUN apt-get install mariadb-server
COPY db/articles.sql /
# ----- FIN COUCHE MYSQL -----

# ----- DÉBUT COUCHE PHP -----
RUN apt-get \
  php-mysql \
  php && \
  rm -f ${DOCUMENTROOT}/index.html && \
  apt-get autoclean -y
COPY app ${DOCUMENTROOT}
# ----- FIN COUCHE PHP -----

# OUVERTURE DU PORT HTTP
EXPOSE 80

# RÉPERTOIRE DE TRAVAIL
WORKDIR ${DOCUMENTROOT}

# DÉMARRAGE DES SERVICES LORS DE L'EXÉCUTION DE L'IMAGE
ENTRYPOINT service mariadb start && mariadb < /articles.sql && apache2ctl -D FOREGROUND
```

# Ateliers Docker

- **Atelier 1** : télécharger et installer Docker sur votre machine locale (ayant un OS récent) ou une machine virtuelle.
- **Atelier 2** : manipuler les images et les conteneurs Docker.
- **Atelier 3** : créer un Dockerfile, générer l'Image Docker et exécuter un conteneur Docker.
- **Atelier 4** : publier une image sur le repository Docker Hub.
- ➔ Suivre les 4 ateliers sur [GitHub Ateliers Docker](#)

# KUBERNETES

---

- K8s, qu'est-ce que c'est ?
- Atelier : minikube & kubectl
- Objets K8s
- Contrôleurs K8s
- Cluster K8s
- Atelier : déployer une API REST Spring Boot sur K8s

# K8s, qu'est-ce que c'est ?

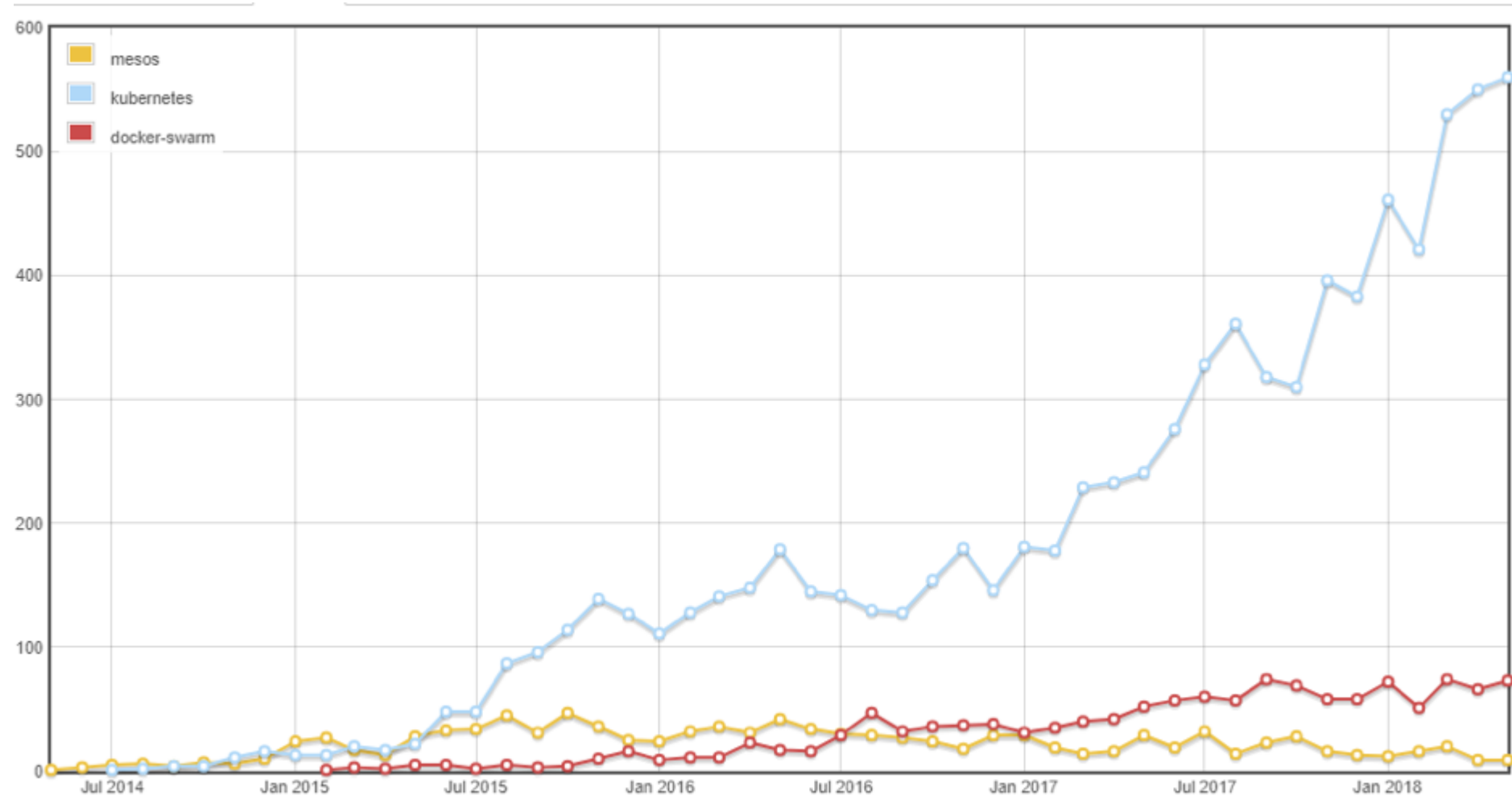
- **Kubernetes** (ou **K8s**) est un système open-source permettant d'automatiser le déploiement, la mise à l'échelle (scalabilité) et la gestion des applications conteneurisées.
- En d'autres termes, **K8s** est un orchestrateur de conteneurs → il permet de donc gérer le cycle de vie (déployer, exécuter, surveiller, mettre à l'échelle et coordonner) des conteneurs (Docker-Containers par exemple).



**kubernetes**

- Du fait de sa flexibilité, **K8s** est présent dans la plupart des fournisseurs Cloud (Google, AWS, Azure, etc.)

# K8s vs. docker-swarm vs. mesos

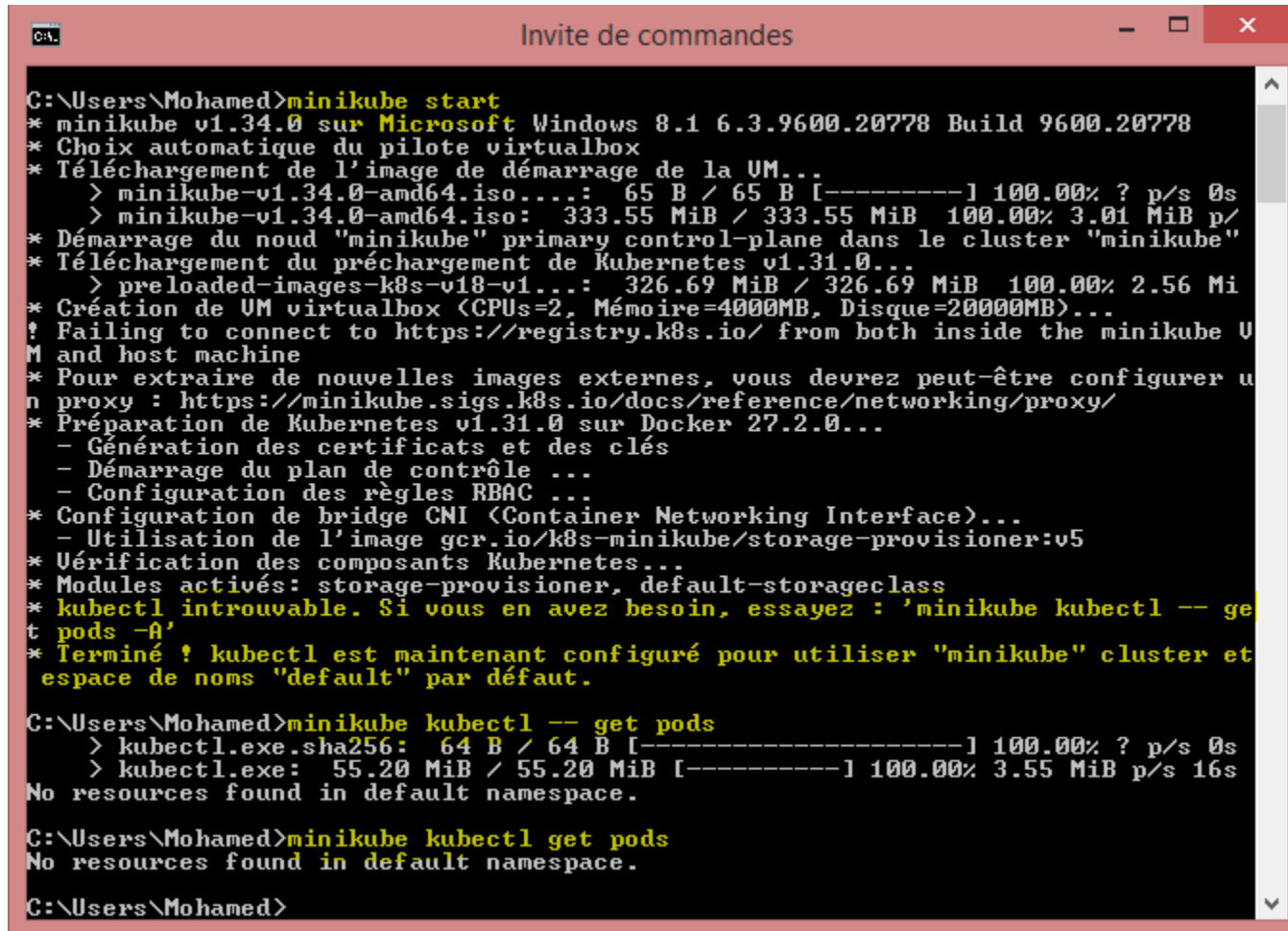


# Atelier K8s : minikube et kubectl

- **minikube** est une version allégée de K8s déployable sur un poste de travail permettant d'éviter d'avoir un réseau de plusieurs machines pour déployer Kubernetes.
  - ➔ installer minikube en suivant les étapes de la doc officielle : <https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download>
- **kubectl** est une interface en ligne de commande (**CLI**) très populaire utilisée pour interagir avec les clusters Kubernetes. Par exemple, il est possible d'afficher l'ensemble des pods d'un cluster K8s avec la commande suivante : `kubectl get pods`
  - ➔ Pour plus d'info sur les commandes kubectl : <https://kubernetes.io/docs/reference/kubectl/>



# Atelier K8s : minikube et kubectl



```
C:\Users\Mohamed>minikube start
* minikube v1.34.0 sur Microsoft Windows 8.1 6.3.9600.20778 Build 9600.20778
* Choix automatique du pilote virtualbox
* Téléchargement de l'image de démarrage de la VM...
  > minikube-v1.34.0-amd64.iso....: 65 B / 65 B [-----] 100.00% ? p/s 0s
  > minikube-v1.34.0-amd64.iso: 333.55 MiB / 333.55 MiB 100.00% 3.01 MiB p/
* Démarrage du nœud "minikube" primary control-plane dans le cluster "minikube"
* Téléchargement du préchargement de Kubernetes v1.31.0...
  > preloaded-images-k8s-v18-v1....: 326.69 MiB / 326.69 MiB 100.00% 2.56 Mi
* Création de VM virtualbox (CPUs=2, Mémoire=4000MB, Disque=20000MB)...
! Failing to connect to https://registry.k8s.io/ from both inside the minikube U
M and host machine
* Pour extraire de nouvelles images externes, vous devrez peut-être configurer u
n proxy : https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Préparation de Kubernetes v1.31.0 sur Docker 27.2.0...
  - Génération des certificats et des clés
  - Démarrage du plan de contrôle ...
  - Configuration des règles RBAC ...
* Configuration de bridge CNI (Container Networking Interface)...
  - Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
* Vérification des composants Kubernetes...
* Modules activés: storage-provisioner, default-storageclass
* kubectl introuvable. Si vous en avez besoin, essayez : 'minikube kubectl -- ge
t pods -A'
* Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et
espace de noms "default" par défaut.

C:\Users\Mohamed>minikube kubectl -- get pods
  > kubectl.exe.sha256: 64 B / 64 B [-----] 100.00% ? p/s 0s
  > kubectl.exe: 55.20 MiB / 55.20 MiB [-----] 100.00% 3.55 MiB p/s 16s
No resources found in default namespace.

C:\Users\Mohamed>minikube kubectl get pods
No resources found in default namespace.

C:\Users\Mohamed>
```

# Atelier K8s : minikube et kubectl

- Exécutez et interprétez les commandes suivantes :
  - minikube start
  - minikube status
  - minikube dashboard
  - minikube kubectl -- get pods
  - minikube kubectl -- get pods -A
  - minikube kubectl -- get nodes
  - minikube stop

# OBJETS K8S

---

- POD
- VOLUME
- SERVICE
- NAMESPACE

# Objets K8s : PODS

- L'objet principal dans un cluster K8s est le **pod**. Un nœud du cluster héberge un ou plusieurs pods ; un pod exécute un ou plusieurs conteneurs, dispose d'un volume de stockage et d'une adresse IP.
- Exemple : une entreprise propose une application et une base de données pour chacun de ses clients →
  - Nous pouvons alors déployer un pod par client et chaque pod contient un premier conteneur pour l'application et deuxième conteneur pour la base de données.
  - Nous pouvons aussi déployer le conteneur de la base de données dans un pod dédié et mettre plusieurs pods parallèles pour exécuter le conteneur de l'application.



*Lorsqu'il n'y a plus de processus qui tourne sur un pod, celui-ci est systématiquement supprimé. La suppression d'un pod entraîne la perte de toutes les données qui lui sont attachées.*

# Objets K8s : PODS

- La configuration d'un POD se fait par le biais d'un Manifest qui est un YAML (ou parfois JSON) comprenant 4 sections :
  - **apiVersion** : la version
  - **kind** : Pod (d'autres type existent comme Deployment, ReplicaSet, etc.)
  - **Metadata** : par ajouter des informations (name, labels, etc.).
  - **spec** : pour spécifier les conteneurs à exécuter dans le pod
- Voici un exemple d'un Manifest basique (*pod-exemple.yaml*) :

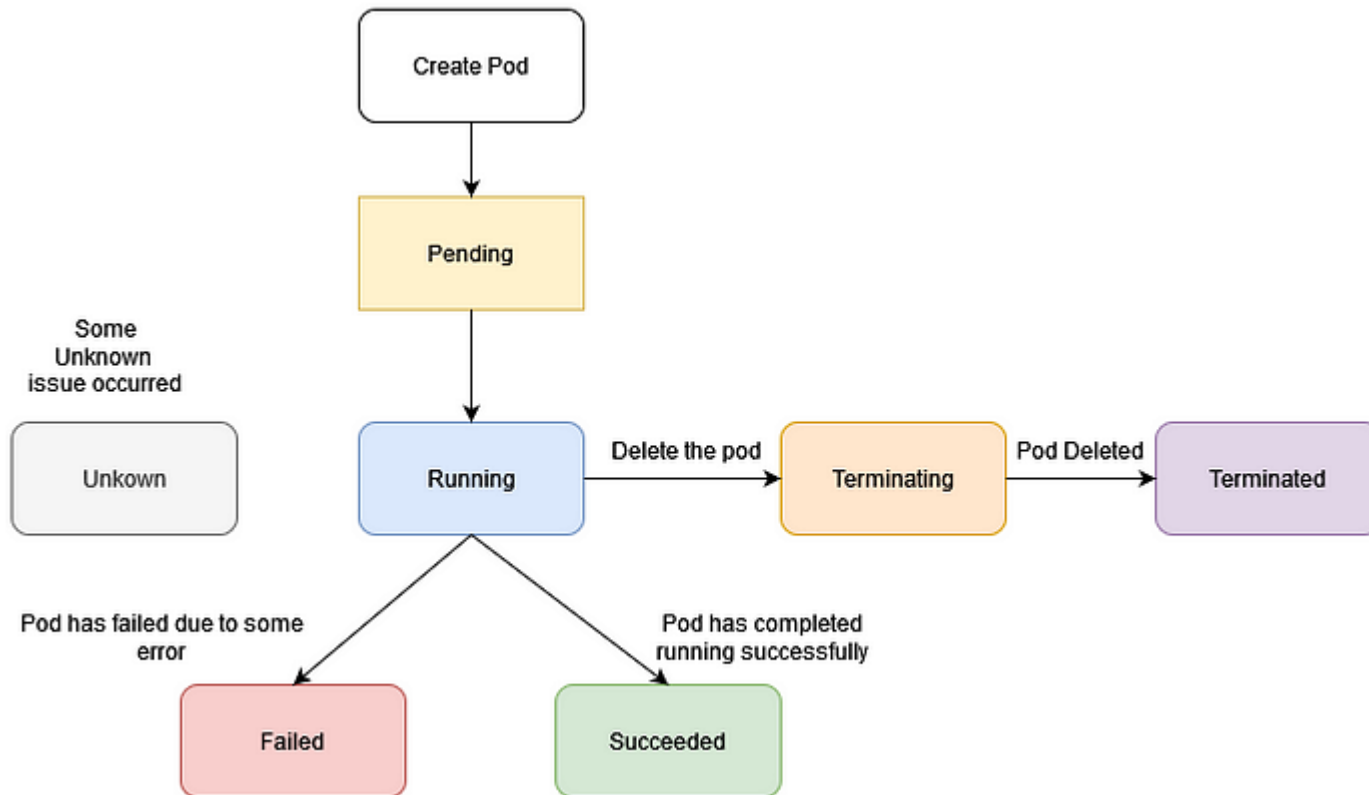
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

- Le Pod précédent peut être créé par la commande kubectl suivante :

```
kubectl apply -f pod-exemple.yaml
```

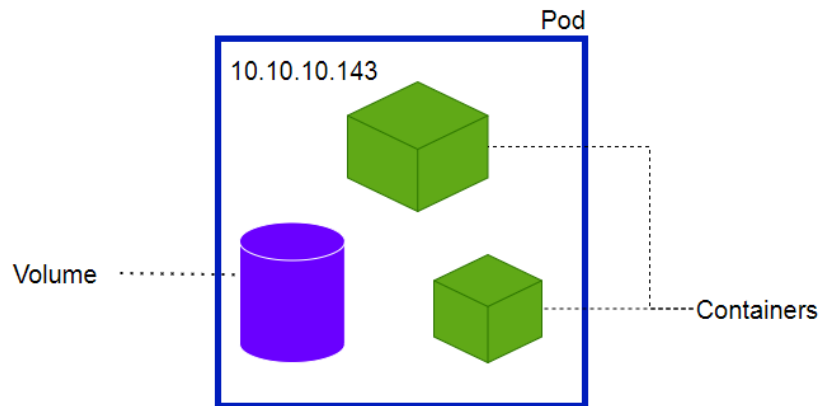
# Objets K8s : PODS


- Le cycle de vie d'un Pod est déterminé au travers d'un ensemble d'états (**Pod STATUS**) :



# Objets K8s : VOLUMES (2/4)

- Le système de fichiers dans un conteneur Kubernetes fournit un stockage éphémère → le redémarrage du conteneur effacera toutes ses données.
- Un **Volume** K8s fournit un stockage qui persiste pendant toute la durée de vie du pod.
- Un Volume peut être utilisé comme espace disque partagé pour les conteneurs à l'intérieur du pod.



 Le redémarrage du pod effacera toutes ses données. Selon le besoin, on peut utiliser d'autres type de stockage K8s comme **Persistent Volume**, **config Map**, **Secret**, etc.

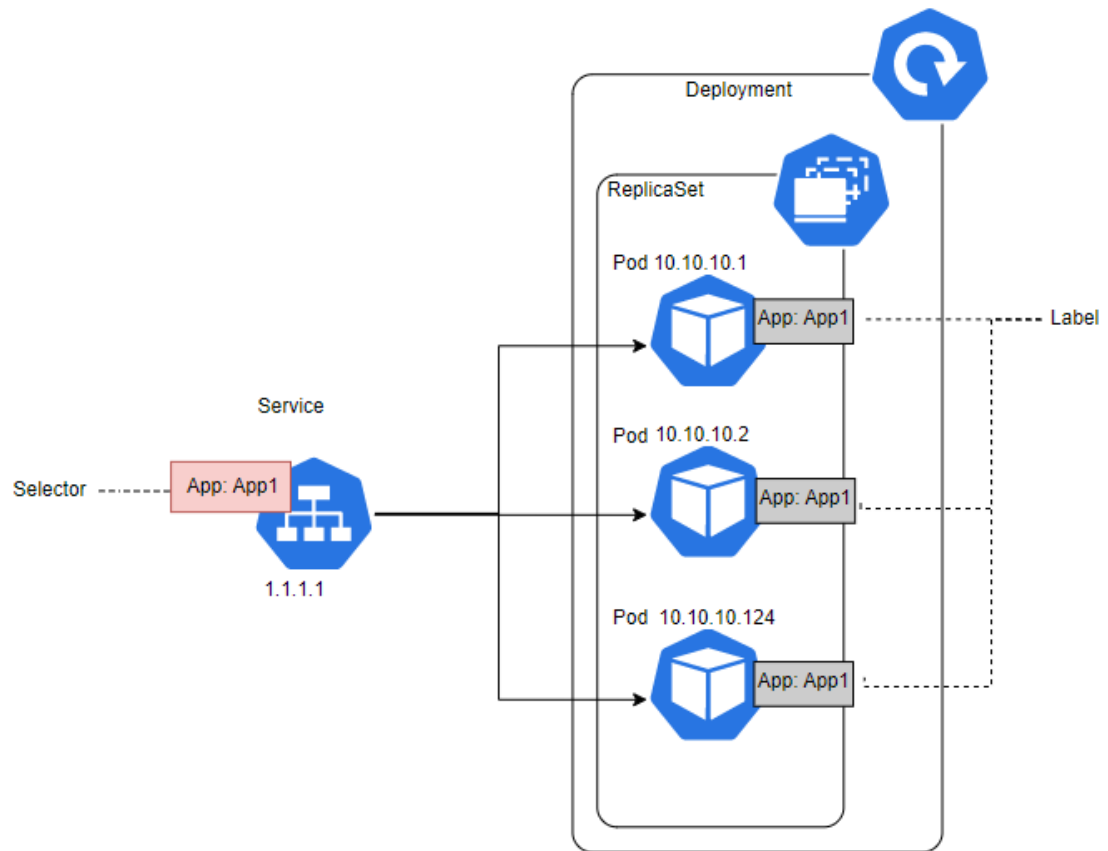
# Objets K8s : SERVICES (3/4)

- L'accès à un conteneur se fait via l'adresse IP de son pod. si ce dernier redémarre, un nouveau pod sera créé et aura une nouvelle adresse IP. Comment peut-on accéder au conteneur sans se soucier de l'adresse IP du pod ? ➔ **Service K8s**.
- Un Service K8s permet aux pods de recevoir du trafic même après le redémarrage (changement d'adresse IP).
- Concrètement, pour cibler des pods, un service va utiliser ce qu'on appelle un **Selector**, qui va chercher dans le cluster des objets possédant des paires **clé / valeurs** qui correspondent à celles qu'il attend.



# Objets K8s : SERVICES (3/4)

Le **Service** (exposé via son adresse IP 1.1.1.1) va utiliser un **Selector** qui va chercher tous les pods ayant le label "App: App1" ayant le status "Running" et récupérer leurs adresses IP pour pouvoir les interroger.



# Objets K8s : NAMESPACES (4/4)

- un **NameSpace** fournit un mécanisme pour isoler des groupes de ressources au sein d'un même cluster.
- Les namespaces sont utilisés dans des environnements avec de nombreux utilisateurs répartis sur plusieurs équipes ou projets, ou même séparant des environnements tels que le développement, l'intégration, la qualification et la production.
- La portée basée sur un NameSpace s'applique uniquement aux objets avec NameSpace (ex. pod, services, etc.) et non aux objets à l'échelle du cluster (ex. StorageClass, Nodes, Persistent Volume, etc.).

# WORKLOADS K8S

---

- Deployment
- ReplicaSet
- StatefulSet
- DaemonSet
- Job

# Workloads

- Un Pod K8s a un cycle de vie défini. Lorsqu'un nœud du cluster subit une panne, tous les pods de ce nœud seront en échec. K8s traite cet échec en recréant automatiquement des nouveaux pods grâce à ses **Workloads** qui s'assurent du bon nombre de pods souhaités et que le bon type de pods soient en fonction.
- Kubernetes fournit plusieurs ressources workload :
  - Deployment
  - ReplicaSet
  - StatefulSet
  - DaemonSet
  - Job & CronJob

# Workloads K8s : Deployment

- Le **Deployment** est une bonne approche pour manager une application stateless sur un cluster K8s où tous les Pods d'un Deployment sont interchangeables et peuvent être remplacés si besoin.
- Un Deployment représente un ensemble de ReplicaSets d'une application (une abstraction de haut-niveau) pour contrôler les pods.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
    
```

# Workloads K8s : ReplicaSet

- Un **ReplicaSet** c'est le workload qui s'assure que le nombre de pods souhaités est égal à ceux en fonction.
- Une fois un ReplicaSet est créé, toute modification apportée à un conteneur n'est pas prise en charge → il faudra redémarrer manuellement le ReplicaSet.
- En revanche, toutes les modifications apportées à un Deployment sont appliquées automatiquement.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
```

# Workloads K8s : StatefulSet, DaemonSet, Job

- Un **StatefulSet** est un workload qui permet de lancer un ou plusieurs Pods qui ***persistent*** des données. Le StatefulSet fait le lien entre le Pod et le Persistent Volume. [Voici un exemple](#).
- Un **DaemonSet** gèrent les Pods qui effectuent des actions sur le nœud sur lequel ils tournent. On utilise DaemonSet le plus souvent pour gérer du stockage, du monitoring ou de la journalisation de log. [Voici un exemple](#).
- Un **Job** crée des Pods et s'assure qu'ils se terminent avec succès. Un Job est une tâche ponctuelle (exécutée une seule fois). Un **CronJob**, en revanche, est une tâche récurrente planifiée. Voici deux exemples : [Job](#) et [CronJob](#).

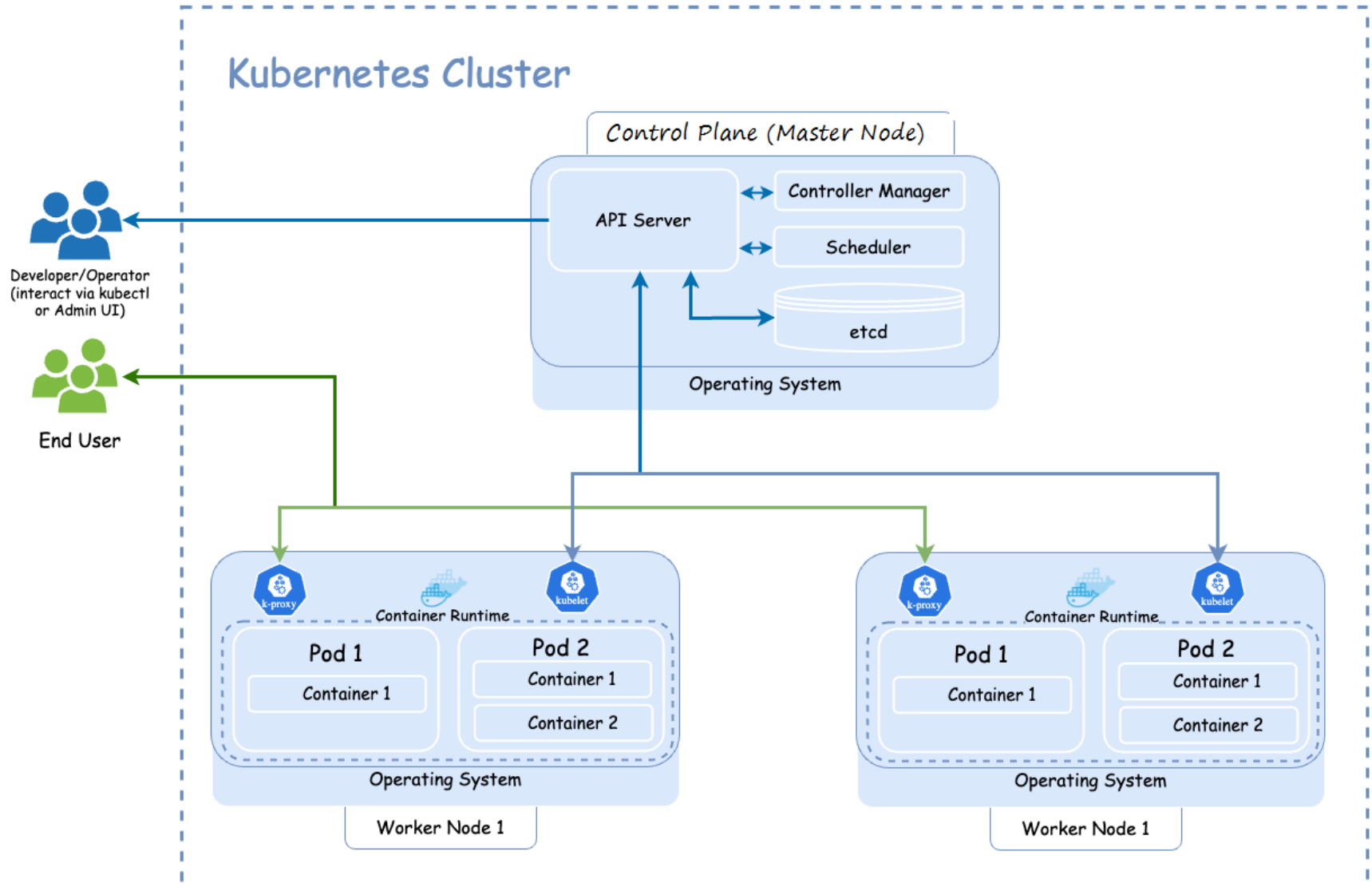
# CLUSTER K8S

---

- Control Plane
- Compute Machine
- Persistent Storage
- Container Registry



# Cluster K8s : Architecture



# Cluster K8s : Control Plane

- **Control Plane** est un ensemble de processus qui assignent les tâches, contrôlent les nœuds, s'assurent que les configurations décrites dans les différents manifests sont respectées et assurent les ressources pour que les conteneurs peuvent fonctionner correctement.
- Le plan de contrôle est composé de quatre éléments principaux :
  - **kube-apiserver** : c'est une API REST qui prend en charge les demandes internes et externes (via *kubectl* par exemple, ou *kubeadm*).
  - **kube-scheduler** : ce processus planifie l'attribution d'un pod à un nœud en fonction des besoins en ressources (CPU, Mémoire, etc.) du pod.
  - **kube-controller-manager** : c'est un processus qui combine un ensemble de contrôleurs (contrôleur de nœuds, de tâches, de réplication, etc.) dans le but de surveiller l'état du cluster en faisant appliquer la configuration des pods, des déploiements, des services et de toutes les ressources K8s.
  - **etcd** : c'est une base de données clé-valeur où sont stockées les données de configuration et les informations sur l'état du cluster.

# Cluster K8s : Worker nodes

- Les pods sont exécutés dans des **nœuds** de calcul en fonction des ressources requises pour chaque pod et celles disponibles dans chaque nœud. La mise à l'échelle d'un cluster K8s (scalabilité) consiste tout simplement à ajouter des nœuds.
- Dans un nœud on trouve :
  - des **pods** : un pod est une instance d'une application exécutant un ou plusieurs conteneurs étroitement couplés. Chaque pod est instancié à partir d'un Manifest, dispose d'une adresse IP, d'un stockage éphémère et peut se connecter à un stockage persistant.
  - **kubelet** : c'est un agent qui assure la communication entre le plan de contrôle et le nœud (exécuter les actions) et s'assure que les conteneurs des pods sont exécutés.
  - **Container Runtime** : c'est le logiciel utilisé pour exécuter les conteneurs. Le plus connu c'est **Docker** mais d'autres existent comme RKT et CRI-O.
  - **kube-proxy** : c'est un proxy réseau qui s'exécute sur chaque nœud du cluster, et qui aide à faire le routage du trafic réseau aux pods.

# Ateliers K8s

- **Atelier 1** : Installer minikube & kubectl
- **Atelier 2** : Créer une simple API REST avec Spring Boot
- **Atelier 3** : Créer un Dockerfile et générer l'image Docker
- **Atelier 4** : Créer le Deployment et le Service
- **Atelier 5** : Déployer et lancer l'API REST sur minikube
- ➔ Suivre les 5 ateliers sur : [GitHub\\_Ateliers\\_K8s](#)

# Références

- <https://kubernetes.io/fr/docs/home/>
- <https://containers.goffinet.org/>
- <https://docs.docker.com/reference/dockerfile/>