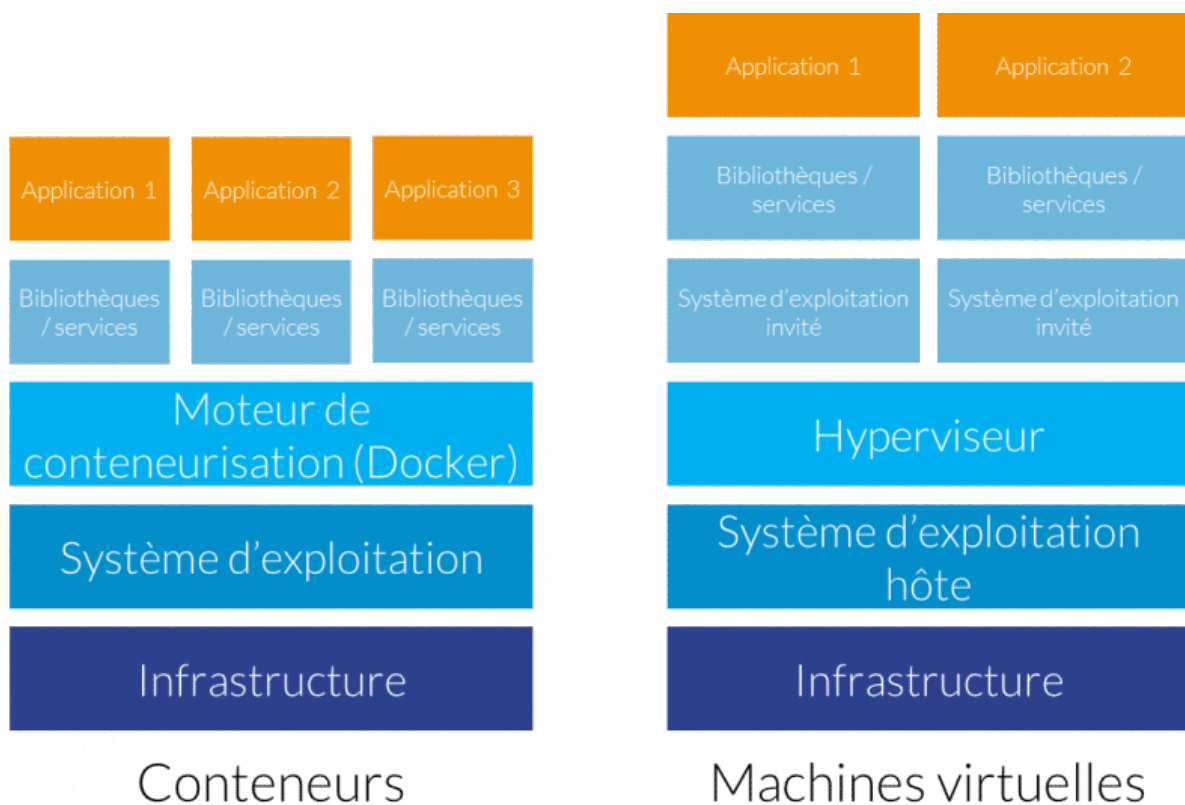


1.1 Faites une recherche sur les différents avantages qu'apporte la conteneurisation et essayez d'expliquer les grands principes avec vos mots

Portabilité : facile à exporter d'un système à un autre  
Vitesse : il utilise le noyau de la machine hôte. Temps de déploiement optimisé  
Évolutivité : Facilité pour rajouter de nouvelles fonctionnalités  
Agilité : facilite l'utilisation des processus DevOps  
Efficacité : conteneur avec une capacité plus petite que la machine virtuelle, temps de démarrage réduit  
Isolation : chaque conteneur est indépendant des autres.  
Sécurité : grâce à l'isolation, les malwares n'affectent qu'un conteneur. Possibilité de définir des autorisations de sécurité

1.2 Faites un schéma comparatif entre une VM et un conteneur (il y en a énormément sur Internet) et expliquez-le avec vos mots



On peut voir sur ce schéma que les conteneurs n'ont pas besoin d'un système d'exploitation invité car ils utilisent directement le noyau de l'hôte. Cette couche en moins réduit grandement le temps d'installation et de déploiement. Vu le volume réduit, il est donc possible de faire tourner plus d'applications sur un même serveur.

1.3 Faites une liste des différentes commandes Docker et expliquez ce qu'elles font avec vos mots

`docker ps (-a)` : affiche toutes les instances qui tournent sur l'environnement. Le "-a" montre aussi les stoppés.

docker images (-a) : montre les images construite. le “-a” montre les intermédiaires

docker network ls : liste les differents resseau

docker-compose ps : affiche les conteneurs lancer avec docker-compose

docker-compose up (-d) (--build) : le up demare le conteneur. le “-d” le joue en tâche de fond. le “--build” contruit image.

docker-compose stop : arrête le conteneur

docker build (-t NAME ) PATH /URL : permet de spécifier le liens/nom de l’image

docker exec -it NAME /ID “sh” /”/bin/bash” : permet de lancer un bash

docker rm ID /NAME : supprime un conteneur

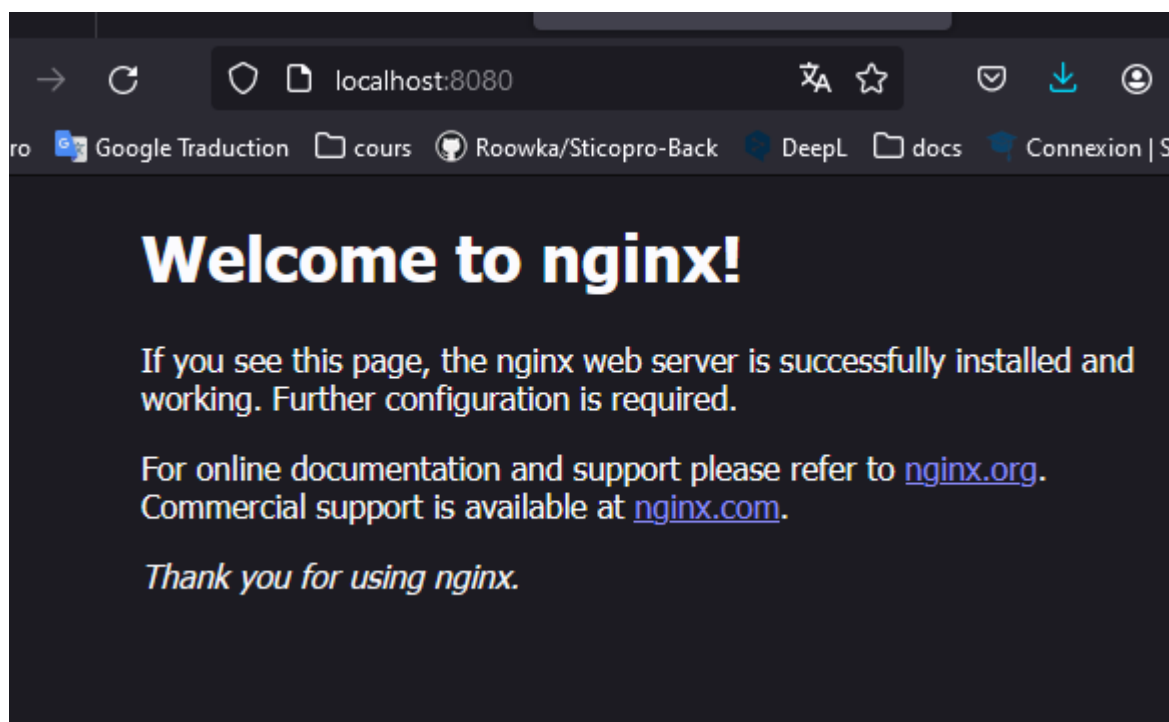
docker-compose rm : supprime tous les conteneurs démarrés avec compose

docker rmi ID /NAME : supprime l’image et toutes les image intermediaires

docker logs ID /NAME (-f --tail NBLINE ) : affiche les logs l’option “ -f -tail NBLINE” permet de voir le flux de logs.

docker-compose logs (ID /NAME ) : idem avec les conteneur compose

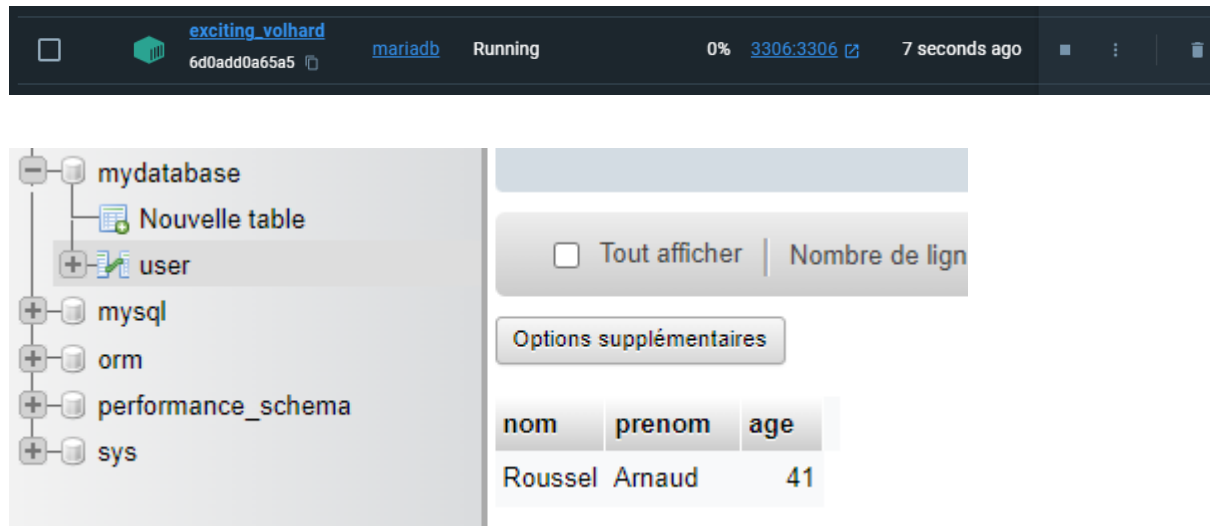
#### 1.4 Un screen prouvant que votre application fonctionne



### 1.5 Un recapitulatif des lignes de commandes utilisées

```
docker run --name some-nginx2 -d -p 8080:80 nginx  
docker ps
```

### 1.6 Avoir une base de données qui tourne sur le port 3306 avec des données persistantes



### 1.7 Expliquer le principe de volume Docker et son utilité

Le volume de conteneur permet de conserver les données même en cas de suppression du conteneur.

### 1.8 Quelle est la différence entre une application stateful et stateless ?

Une application stateless ne stocke pas de donnée ni de transaction et ne fait que du traitement à court terme.

Une application stateful peut être réutilisée indéfiniment. Les transactions peuvent s'affecter les unes des autres. Elles gardent une trace des éléments type URL et paramètres.

### 1.9 Faites une recherche sur les différents mode de réseau (Liste + Explication)

Le bridge : réseau par défaut docker connecte automatiquement les conteneurs mais ils ne sont pas accessibles depuis l'extérieur.

Le none: aucune interface réseau aucune communication interne ou externe.

Le driver Host: permet au conteneur d'utiliser la même interface que hôte. Les conteneurs sont accessibles depuis l'extérieur.

Le driver overlay: permet de créer plusieurs hosts

Le driver macvlan: permet d'attribuer des adresses MAC pour simuler un vrai réseau

### 1.10 Quels sont les modes de réseaux les plus couramment utilisé ?

Les plus courants sont le bridge ( par défaut ) et le driver host ( utilisable depuis l'exterieur)

- Lancer un conteneur qui utilise l'image nginx:latest. En utilisant les commandes adéquates, vous devez trouver :

- La liste des réseaux docker sur votre machine

```
C:\Windows\System32>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c41f06b911bc        bridge             bridge             local
85d94dc65b18        host               host               local
aea564e1846a        none               null               local
```

- A quel réseau appartient ce conteneur

docker network inspect bridge

```
{
  "Name": "bridge",
  "Id": "c41f06b911bc95214b2a83484bd53feb3b2229ee5a",
  "Created": "2023-12-18T07:42:29.837718595Z",
  "Scope": "local",
  "Driver": "bridge",
  "EnableIPv6": false,
  "IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
      {
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1"
      }
    ]
  },
  "Internal": false,
  "Attachable": false,
  "Ingress": false,
  "ConfigFrom": {
    "Network": ""
  },
  "ConfigOnly": false,
  "Containers": {
    "34e11480cb271c76dffc7cf0603ee63b0ad96f679099": {
      "Name": "some-nginx2",
      "EndpointID": "e47a7f6125b24f038de13a468c",
      "MacAddress": "02:42:ac:11:00:03",
      "IPv4Address": "172.17.0.3/16",
      "IPv6Address": ""
    }
  }
}
```

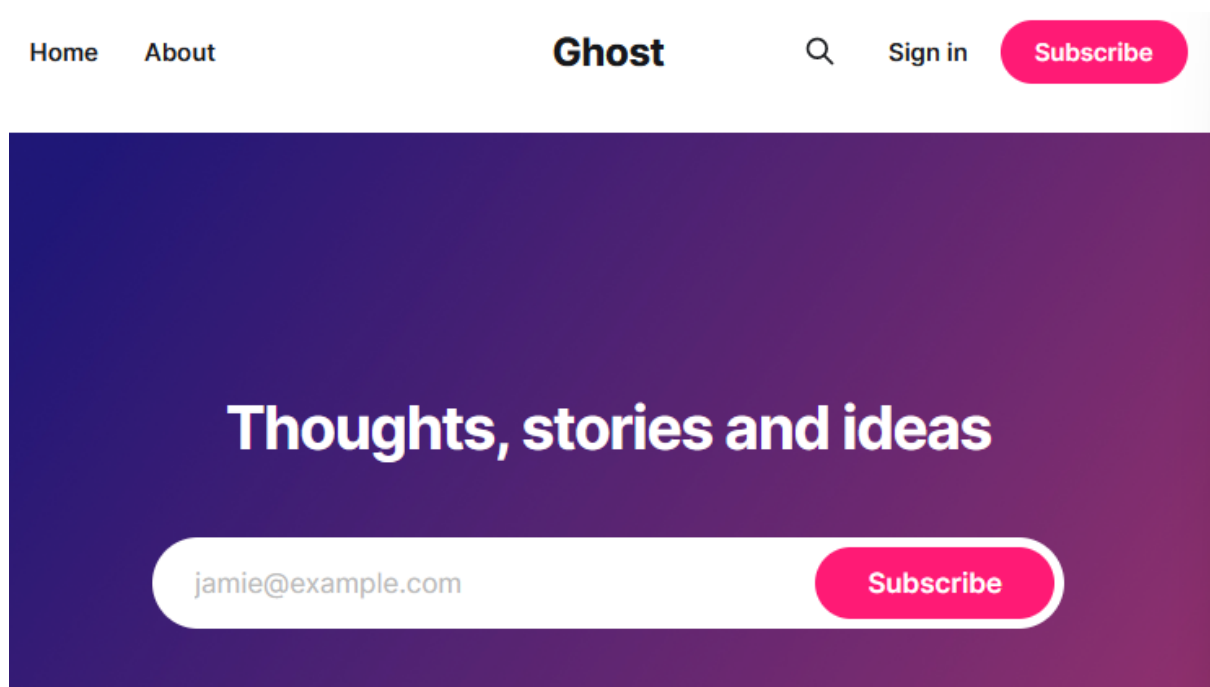
### 1.11 Les commandes Docker utilisées pour configurer l'environnement.

```
docker network create --driver bridge ghost2sql
docker network ls
docker run -d --name some-ghost -p 3001:2368 -e NODE_ENV=development --network
ghost2sql ghost
docker run --name some-mysql -p 3306:3306 --network ghost2sql -e
MYSQL_ROOT_PASSWORD=Pa$$w0rd -d mysql
docker network inspect ghost2sql
```

### 1.12 Une documentation décrivant votre processus, les configurations effectuées et les défis rencontrés.

J'ai commencer créé un reseau de type bridge. Verifier qu'il etait bien créer dans la liste de reseau. J'ai créé un conteneur ghost sur le reseau que je venai de créer en attribuant le port 80 . J'ai créé un conteneur MySQL sur le reseau que je venai de créer en attribuant le port 3306. J'ai verifié que les 2 contenueurs etait bien sur le reseau.


### 1.13 Des captures d'écran de votre blog Ghost fonctionnel.




### 2.1 Le Dockerfile devra être versionné sur Github. Renseigner l'url sur votre rapport

[git@github.com:Andemion/CourDockerMDS.git](https://github.com/Andemion/CourDockerMDS.git)

### 2.2.1 Avoir une image la plus légère possible. A titre indicatif, celle de votre formateur fait 44.1MB.

	<a href="#">horribledocker</a> 2cd81414a682	1.0.0	<a href="#">In use</a>	2 minutes ago	43.24 MB
---	--	-------	------------------------	---------------	----------

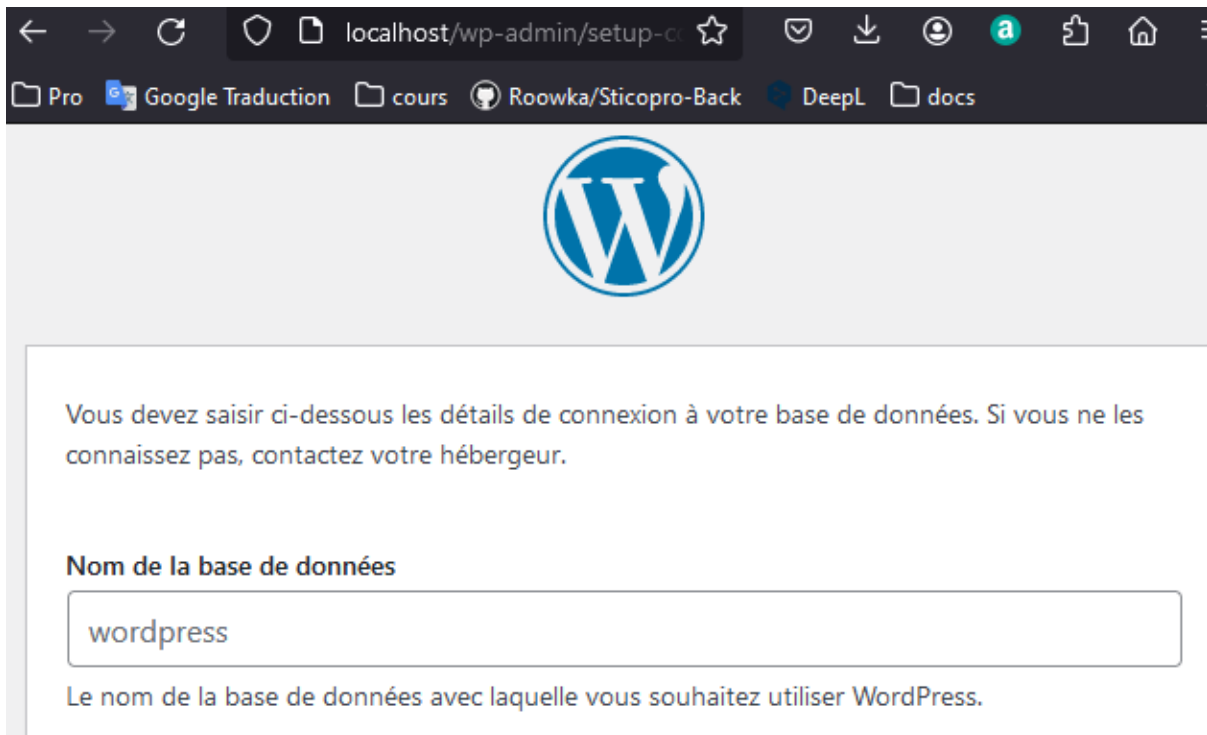
2.2.2 L'image doit être fonctionnelle et le conteneur doit écouter sur le port 8000 de votre machine

	<a href="#">agitated_swirles</a> 540dd72afc7a	<a href="#">horribledoc</a> Running	0%	<a href="#">8000:80</a>
---	--	-------------------------------------	----	-------------------------

2.2.3 Vous devrez push le repo git update sur votre propre github, indiquer le lien et la taille dans votre rapport

[git@github.com](mailto:git@github.com):Andemion/CourDockerMDS.git

2.3.1 Avoir une application fonctionnelle sur l'url <http://localhost:80>



2.3.2 Comment configurez-vous les variables d'environnement?

Les variables sont enregistrées sous la clef "environment" dans chaque service sous la clef "services"

2.3.3 Comment l'application wordpress communique-t-elle avec le conteneur MariaDB ?

J'ai configuré un network avec les 2 services à l'interieur et rajouté la fonction ("depend\_on":-db) sur l'app wordpress

#### 2.4.1 En faisant des recherches, expliquez avec vos mots l'interet d'avoir une registry interne dans une entreprise

Les registres de conteneurs permettent aux développeurs de gagner un temps précieux lors de la création et de la distribution d'applications cloud-native, en servant d'intermédiaire pour le partage d'images de conteneurs entre les systèmes.

#### 2.4.2 Faites une liste des commandes utilisées pour réaliser ces tâches

```
docker login
docker build -t andemyon/mon-app:1.0.0 .
docker push andemyon/mon-app:1.0.0
```

#### 2.4.3 Mettez le lien de votre image pour que votre formateur puisse la pull depuis ça machine

```
docker pull andemyon/mon-app
```

#### 2.5.1 Lien vers l'image Docker sur Docker Hub.

```
docker pull andemyon/back
docker pull andemyon/backoffice
```

#### 2.5.2 Lien vers le dépôt GitHub contenant le code source et les fichiers Docker

[git@github.com:Andemion/CourDockerMDS.git](https://github.com/Andemion/CourDockerMDS.git)

#### 3.1.1 Faites une recherche avancée sur la clusterisation des conteneurs

La clusterisation des conteneurs vise à gérer plusieurs d'entre eux de manière très organisée au sein d'un groupe. Cela aide à avoir des applications en ligne tout le temps, à gérer beaucoup de travail et à simplifier leur gestion. Ça permet aussi de toujours avoir l'applications en lign

#### 3.1.2 Faites une recherche sur le concept de microservice

Les microservices représentent une méthode d'architecture logicielle où les applications sont conçues comme une série de services distincts et spécialisés. Les microservices offrent une méthode adaptable pour créer des applications modulaires, évolutives et faciles à maintenir.

#### 3.1.3 Faites une recherche sur les concepts de scalability, availability et Le load Balancing

Scalability : la scalabilité fait référence à la capacité d'un système à s'adapter et à gérer l'augmentation de la charge de travail ou du nombre d'utilisateurs sans perdre les performances.

Availability :il s'agit de la capacité d'un système à rester opérationnel et accessible pour les utilisateurs, même en cas de défaillance d'une partie du système.

Load Balancing :l'équilibrage de charge est une technique utilisée pour distribuer la charge de travail sur plusieurs ressources afin d'optimiser les performances, d'assurer la scalabilité et de maintenir la disponibilité.

### 3.1.4 En quoi la clusterisation permet de répondre à ces problématiques ?

Les clusters permet l'ajout simple de nouveaux nœuds ou serveurs. En répartissant la charge entre ces nœuds, le système peut s'agrandir horizontalement pour répondre à une demande croissante.Les clusters répartissent équitablement la charge entre les nœuds, évitant la surcharge et garantissant des performances optimales. Les load balancers redirigent le trafic entre les nœuds pour équilibrer la charge et maintient la disponibilité.

### 3.1.5 Faites une recherches sur des outils permettant d'orchestrer un cluster de conteneurs

- Kubernetes (K8s) : outil principal pour orchestrer les conteneurs. Il déploie, ajuste la taille et gère les applications conteneurisées sur plusieurs machines.
- Docker Swarm : Intégré à Docker, simple à utiliser pour les déploiements Docker.
- Amazon Elastic Kubernetes Service (EKS) : Gère facilement les applications Kubernetes sur AWS.
- Google Kubernetes Engine (GKE) : Équivalent à EKS, mais pour Google Cloud Platform.
- Azure Kubernetes Service (AKS) : Offre un service entièrement géré sur Microsoft Azure.

### 3.2.1 Faites une liste des commandes que vous avez utilisé et expliquez leurs buts

- Initialisation d'un cluster Swarm : docker swarm init
- docker swarm join: ajout de nœuds au cluster
- docker service create : Création d'un service
- docker service scale : mise à l'échelle des services
- docker node ls / docker service ls : Visualisation de l'état du cluster
- docker service update : Mise à jour d'un service

### 3.2.2 Faites un résumé des différents éléments que vous avez rencontré pendant les labs

- Comprendre comment initialiser un cluster Swarm, les différences entre les nœuds managers et workers.
- La création de services pour déployer des applications et la compréhension des options de configuration.
- Mise à l'échelle, mise à jour et suppression des services, ainsi que la visualisation de leur état.
- Comment utiliser des fichiers de configuration pour déployer des services dans un cluster Swarm.
- Utiliser des stacks pour déployer des applications plus complexes, combinant plusieurs services.



- les commandes pour voir l'état du cluster, surveiller les services et les nœuds, ainsi que pour mettre à jour les services en cours d'exécution.

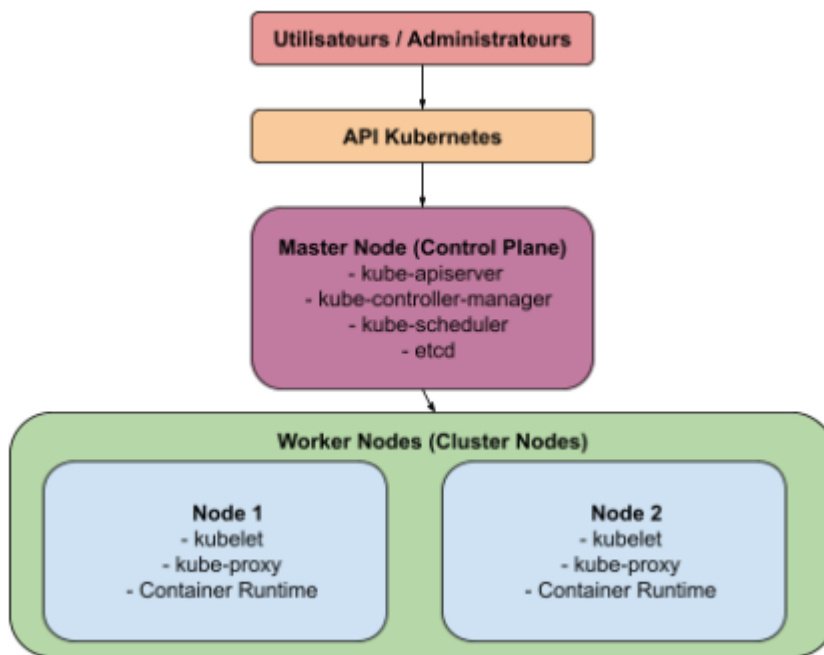
### 3.2.3 Un document qui identifie concrètement le problème et une remédiation au problème

Le problème c'est que chaque réplique persiste les données dans un volume Docker distinct, ce qui crée des incohérences et des problèmes de versionning sur chaque réplique. Cela conduit à des pertes de fichiers ou à une inaccessibilité intermittente pour les utilisateurs. Le tout principalement dû au volume et network en drive.

### 3.3.1 Faire une liste de l'ensemble des composants se trouvant sur les nœuds Kubernetes et expliquez leurs fonctionnements

- kube-apiserver : Composant central de l'API Kubernetes. Il expose l'API Kubernetes, permet aux utilisateurs et aux autres composants de communiquer avec le cluster. Toutes les opérations sur le cluster passent par cet élément.
- kube-controller-manager : Regroupe plusieurs contrôleurs qui surveillent en permanence l'état réel du cluster Kubernetes grâce à l'API et répondent en conséquence pour maintenir l'état voulu.
- kube-scheduler : Planifie les pods sur les nœuds du cluster, en fonction de diverses exigences telles que les ressources disponibles, les contraintes, ...
- kubelet : Chaque nœud possède un kubelet, qui agit comme un agent. Il gère les pods, s'assure qu'ils fonctionnent correctement, et communique avec le kube-apiserver pour gérer les actions requises.
- kube-proxy : Maintient les règles réseau sur les nœuds. Il gère la communication réseau pour les services exposés à l'intérieur ou à l'extérieur du cluster.
- etcd : Base de données de stockage clé-valeur cohérente, légère et distribuée, utilisée pour stocker la configuration et les données d'état du cluster Kubernetes.
- Container runtime (Docker, containerd,...) : Logiciel qui exécute les conteneurs. Kubernetes prend en charge plusieurs runtimes, dont Docker, containerd et d'autres.

### 3.3.2 Faites un schéma de l'architecture globale Kubernetes



### 3.3.3 Faites une recherches sur l'ensemble des ressources Kubernetes citées plus haut et expliquez leurs rôles

- Namespaces : Les espaces de noms Kubernetes permettent de diviser un cluster en unités logiques, fournissant un moyen de regrouper les objets Kubernetes en fonction de leur finalité, projet, ou environnement.
- Pods : Les pods sont l'unité de base dans Kubernetes, contenant un ou plusieurs conteneurs. Ils partagent un contexte réseau et un stockage.
- Services : Ils fournissent un moyen stable pour accéder aux pods. Ils peuvent exposer des applications déployées à l'intérieur du cluster ou à l'extérieur, gérant la découverte dynamique des pods.
- Volumes : Stocke des données persistantes pour les conteneurs. Ils offrent un stockage partagé entre les conteneurs dans un même pod ou pour le stockage persistant des données.
- Deployment : Décrit l'état souhaité pour les pods et les contrôlent via des mises à jour de versions, des rollbacks, et des montées en charge.
- StatefulSet : Cette ressource est utilisée pour les applications qui nécessitent des identités uniques, des stockages persistants et un déploiement ordonné.
- DaemonSet : Assure le déploiement d'un pod sur chaque nœud dans le cluster, souvent utilisé pour les services tels que des agents de surveillance ou des outils de collecte de données.
- Jobs et CronJobs : Exécute des tâches à compléter avant de s'arrêter, tandis que les CronJobs planifient des tâches à des moments spécifiques, comme des tâches planifiées.

### 3.3.4 Pour les ressources listées dans la partie , trouvez un exemple de manifeste YAML et decrivez la

Namespaces (crée un nouvel espace "namespace"):

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace
```

Pods (crée un pod "nginx-pod" dans un conteneur ):

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
  ports:
    - containerPort: 80
```

Services (crée un service "nginx-service" qui sélectionne les l'étiquette de pod "app: nginx" et expose le port 80):

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

Volumes (crée une demande de revendication de volume persistant (PVC) "my-volume" pour une capacité de stockage de 1Gi.):

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-volume
spec:
  accessModes:
    - ReadWriteOnce
resources:
  requests:
    storage: 1Gi
```

Deployment (crée un déploiement "nginx-deployment" avec 3 conteneurs):

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-container

image: nginx:latest

ports:

- containerPort: 80

StatefulSet (crée un StatefulSet avec 2 conteneurs Nginx. Il définit un stockage persistant de 1Gi pour chaque réplique):

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: statefulSet

spec:

replicas: 2

serviceName: "nginx"

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

volumeMounts:

- name: nginx-persistent-storage

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:  
  name: nginx-persistent-storage  
spec:  
  accessModes: [ "ReadWriteOnce" ]  
  resources:  
    requests:  
    storage: 1Gi

### 3.3.5 Faites une recherche sur l'ensemble des commandes Kubernetes les plus utilisées

kubectl create : Crée une ressource à partir d'un fichier de configuration  
kubectl apply : met à jour un fichier de configuration sur une ressource  
kubectl get : affiche les informations sur les ressources  
kubectl describe : Affiche des détails sur une ressource spécifique  
kubectl delete : Supprime une ressource  
kubectl rollout : Gère les déploiements et les mises à jour  
kubectl scale : Change le nombre de répliques d'un déploiement  
kubectl rollout history : Affiche l'historique des mises à jour d'un déploiement  
kubectl logs : Affiche les journaux d'un pod  
kubectl exec : Exécute une commande à l'intérieur d'un conteneur d'un pod  
kubectl port-forward : Fait suivre un port local vers un port d'un pod  
kubectl top : Affiche les informations de consommation des ressources par les pods  
kubectl expose : Crée une exposition pour un service existant  
kubectl get services : Affiche les services disponibles  
kubectl get statefulsets : Affiche les ensembles d'états  
kubectl get persistentvolumeclaims : Affiche les demandes de revendication de volumes persistants  
kubectl get storageclasses : Affiche les classes de stockage  
kubectl get configmaps : Affiche les maps de configuration  
kubectl get secrets : Affiche les secrets  
kubectl apply -f : Applique un fichier de configuration sur une ressource