

0 - Introduction

paul.millet@lameduse.fr



# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**

Version : 05/08/2024

WWW.LAMEDUSE.FR



LaMeDuSe

# Prérequis

- Connaissances de base de l'environnement Unix/Linux.



# Objectifs

- Comprendre le positionnement de Docker et des conteneurs
- Manipuler l'interface en ligne de commande de Docker pour créer des conteneurs
- Mettre en œuvre et déployer des applications dans des conteneurs
- Administrer des conteneurs
- Déployer rapidement des applications à l'aide de conteneurs
- Identifier les risques et challenges inhérents à Docker afin d'anticiper les bonnes solutions

# Tour de table

- Présentation des membres
- Vos attentes vis-à-vis de la formation

# Programme de la formation

1. De la virtualisation à Docker
2. Présentation de Docker
3. Mise en œuvre en ligne de commande
4. Création de conteneur personnalisé
5. Mettre en œuvre une application multiconteneur
6. Interfaces d'administration
7. Administrer des conteneurs en production
8. Orchestration et clustérisation
9. Ressources des TP
10. Annexe & Ressources connexes



# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**



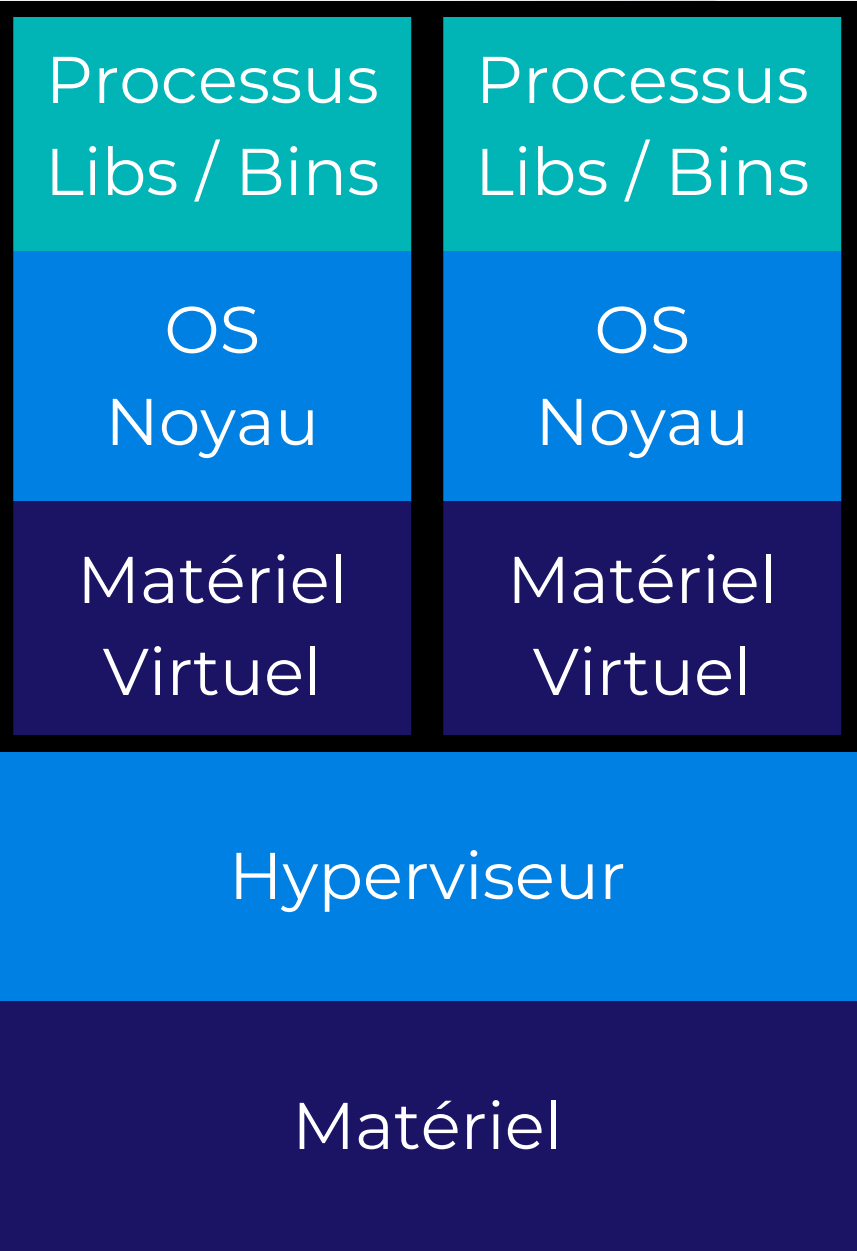


# De la virtualisation à Docker

1. Les différents types de virtualisation.
2. La conteneurisation : LXC, namespaces, control-groups.
3. Le positionnement de Docker.
4. Docker versus virtualisation.

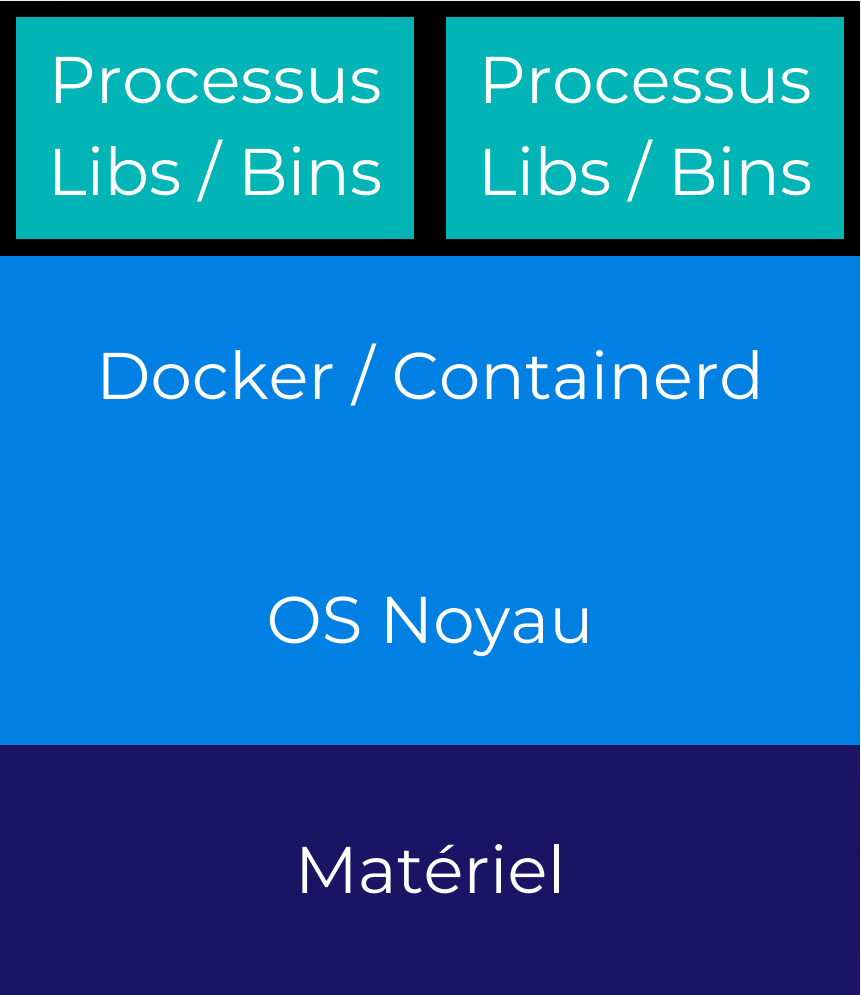


# Les différents types de virtualisation



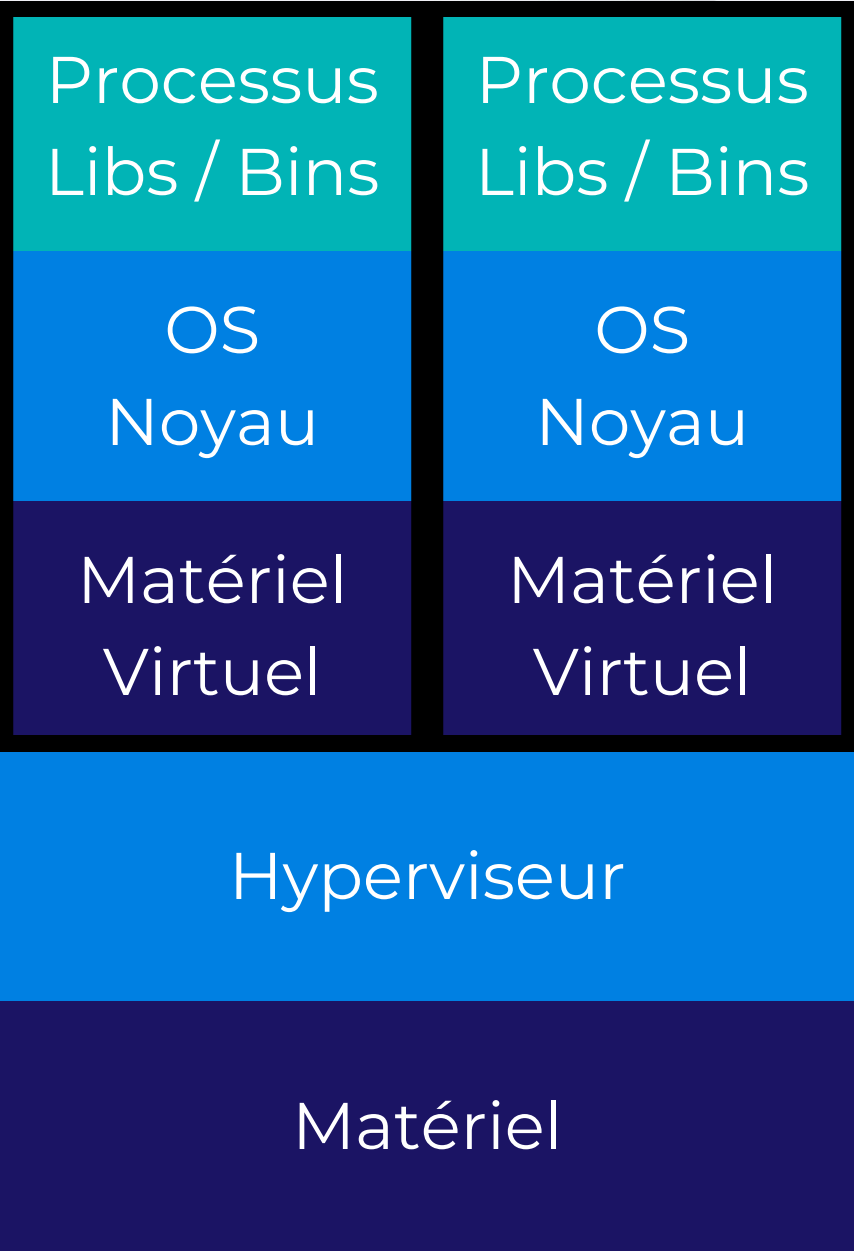
Machine virtuelle

|                                                           |                                                                  |
|-----------------------------------------------------------|------------------------------------------------------------------|
| Stockage persistant (Statefull)                           | Stockage non persistant (Stateless)<br>OU<br>Stockage persistant |
| Séparation des fichiers et librairie                      | Séparation des fichiers et librairie                             |
| Système d'exploitation indépendant (as son propre kernel) | Utilise les ressources du système d'exploitation (kernel)        |
| Matériel virtualisé                                       | Pas de virtualisation matérielle                                 |
| Consommation importante<br>Isolation totale               | Consommation réduite<br>Isolation partielle                      |



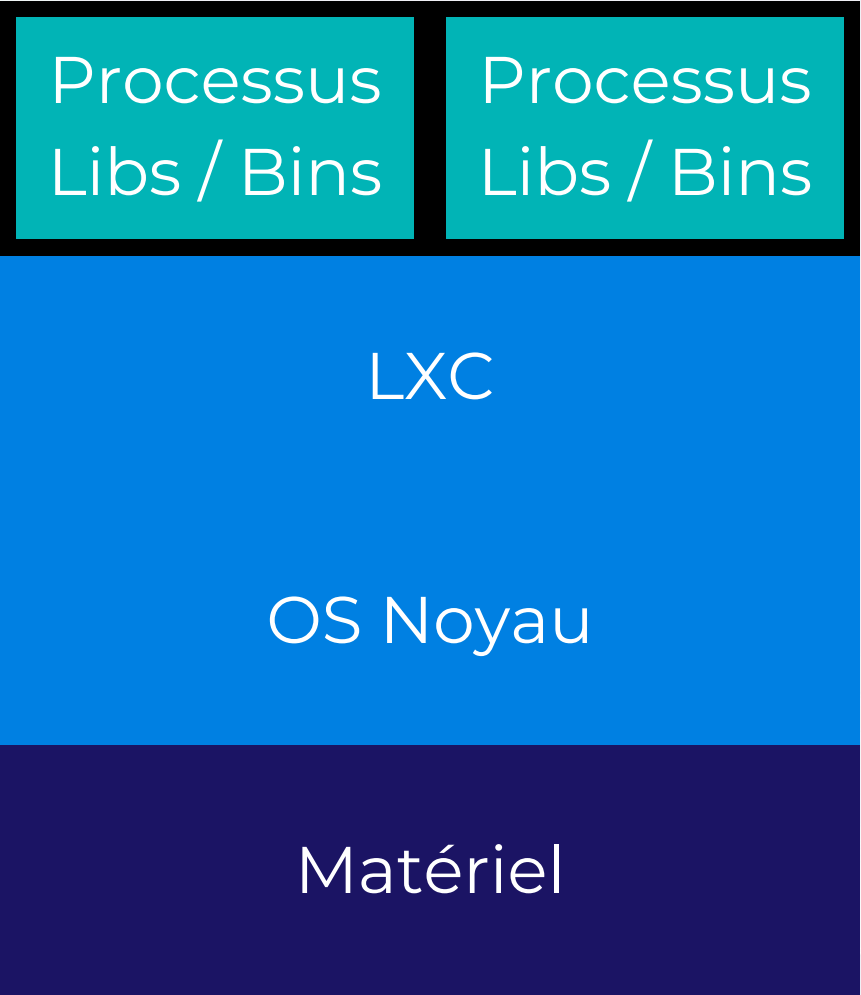
Conteneurisation

# La contenerisation : Machine virtuelle VS LXC



Machine virtuelle

|                                                                 |                                                                 |
|-----------------------------------------------------------------|-----------------------------------------------------------------|
| Stockage persistant<br>(Statefull)                              | Stockage persistant<br>(Statefull)                              |
| Séparation des fichiers et<br>bibliothèque                      | Séparation des fichiers et<br>bibliothèque                      |
| Système d'exploitation<br>indépendant<br>(as son propre kernel) | Utilise les ressources du<br>système d'exploitation<br>(kernel) |
| Matériel virtualisé                                             | Pas de virtualisation<br>matérielle                             |
| Consommation importante<br>Isolation totale                     | Consommation réduite<br>Isolation partielle                     |



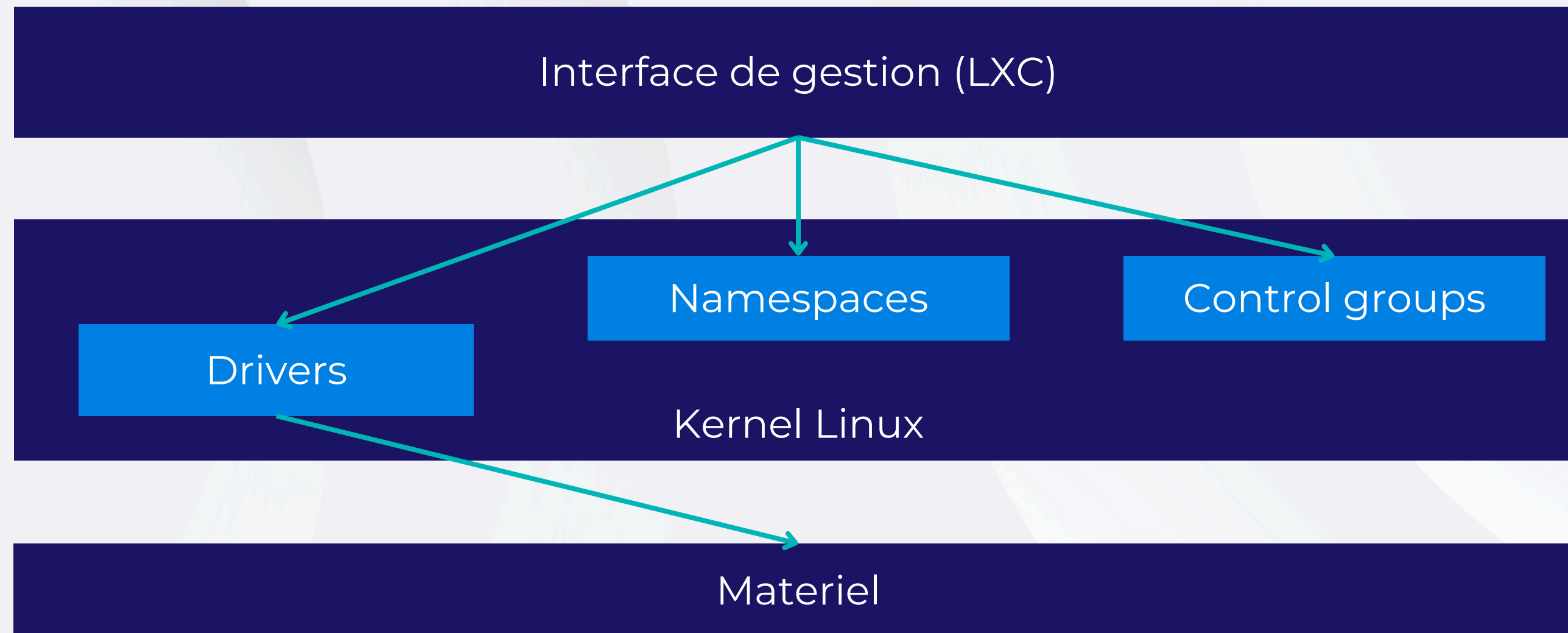
Conteneurisation

LXC



LaMeDuSe

# La contenerisation : namespaces, control-groups et drivers



# Qu'est ce que les namespaces 1/3

Définition : permettent l'isolation de ressources pour un ou plusieurs processus.

Types :

- PID : Isolation de l'identifiant de processus. Deux processus peuvent avoir un même identifiant dans deux espaces séparés.
- UTS : Isolation du nom d'hôte et de domaine.
- Mount: Isolation du système de fichier, point de montage, permettant à chaque espace d'avoir sa propre vue du système de fichier.



# Qu'est ce que les namespaces 2/3

Types :

- IPC: Isolation de la communication inter-processus (mémoire partagée, semaphore, message queues).
- Network: Isolation des ressources réseaux (Interface, adresse IP, table de routage, règle de pare feu).
- User: Isolation de l'utilisateur et des groupes. Les processus de différent espace ont une vue différente des utilisateurs et groupes. Permet d'exécuter un processus avec des permissions administrateur, alors que le container a des privilèges limités sur le système hôte.

## Qu'est ce que les namespaces 3/3

- Cgroup: Isolation de la vue des controlgroups permettant que les processus ne voient que le controlgroup aux quel ils appartiennent et donc sans avoir connaissances des autres présents sur le système.
- Time: Isole le temps du système permettant au processus de différent namespace d'avoir une notion du temps différente (e.g. différentes horloges)

## Qu'est ce que les controlgroups

- Permet une limitation de l'usage des ressources par controlgroup.
- CPU: Limitation de l'usage du temps processeurs.
- Memory: Limitation de l'usage de la mémoire. En cas d'excès, le kernel peut exécuter des actions telles que le swapping ou tuer les processus dans le groupe.
- Block I/O: Limitation de l'usage de l'I/O du disque. Empêchant ainsi la saturation du disque.
- Network: Limitation de la bande passante réseau.
- Devices: Autorise ou restreint l'accès à des ressources matérielles (e.g. disques, interfaces réseau).
- PIDs: Limitation du nombre de processus par groupe.



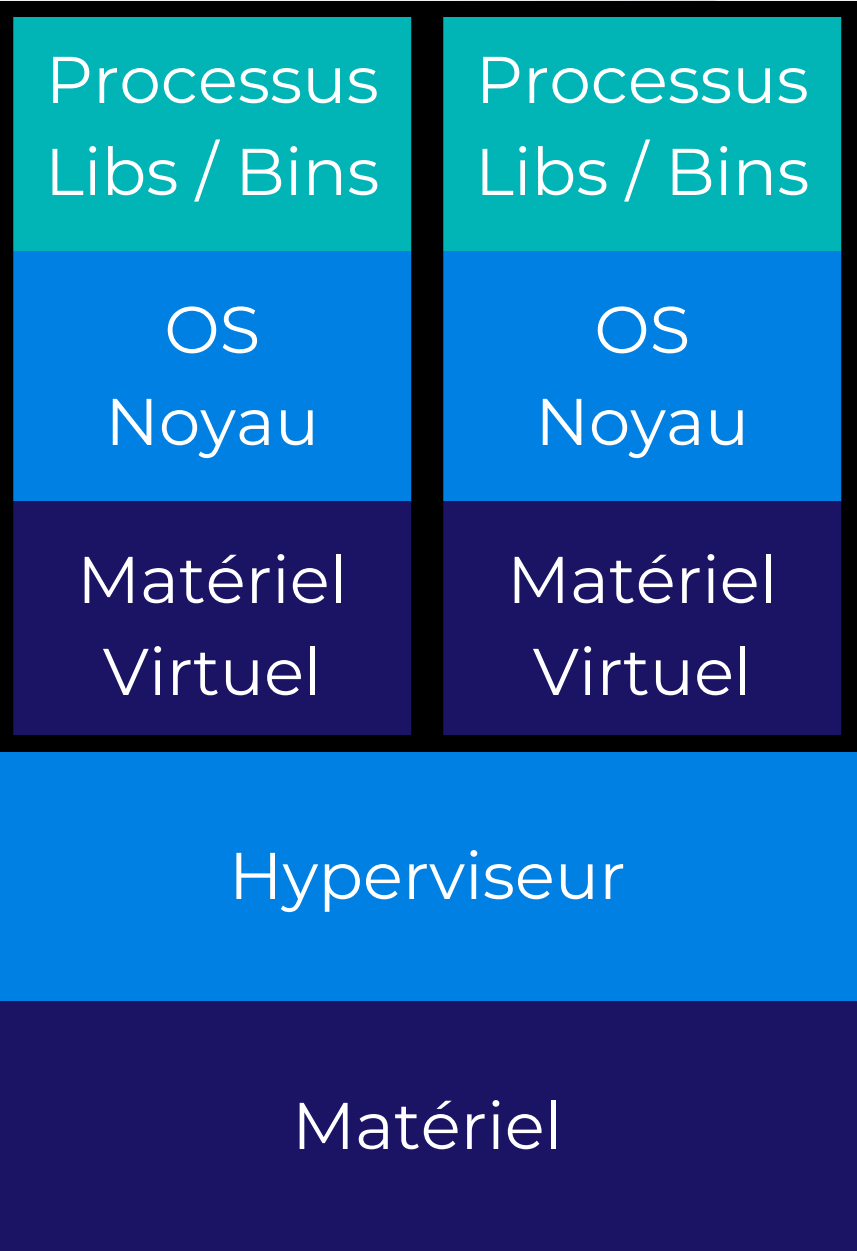
# Le positionnement de Docker

- Simplicité de déploiement et de distribution (images docker, dockerfile)
- Abstraction des drivers (stockage, réseau, IPAM, journalisation)
- Simplicité d'utilisation (utilitaire docker)
- Simplicité de déploiement multi-containers (docker compose)
- Optimisation de la gestion du stockage (système de fichier par couche)
- Éco-système d'image faite par la communauté (docker hub)

Docker : Standardiser un environnement applicatif pour faciliter son déploiement et sa distribution.

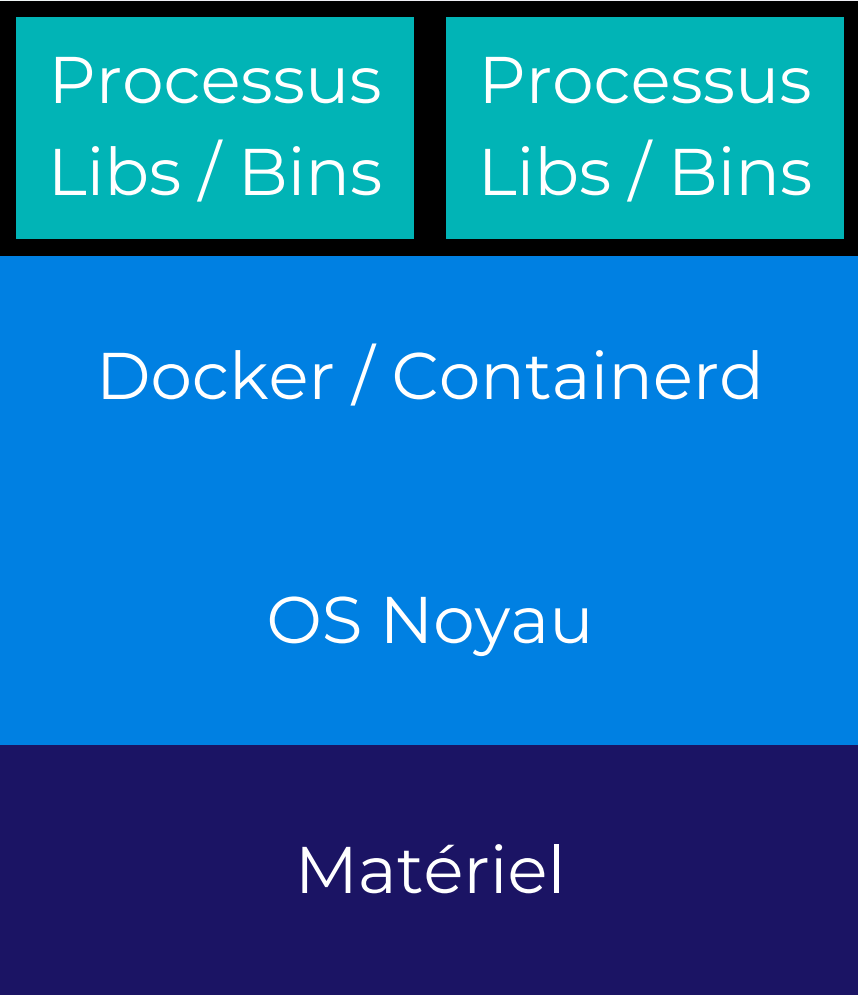


# La contenerisation : Virtualisation VS Docker



Machine virtuelle

|                                                           |                                                           |
|-----------------------------------------------------------|-----------------------------------------------------------|
| Stockage persistant (Statefull)                           | Stockage non persistant par défaut (Stateless)            |
| Séparation des fichiers et librairie                      | Séparation des fichiers et librairie                      |
| Système d'exploitation indépendant (as son propre kernel) | Utilise les ressources du système d'exploitation (kernel) |
| Matériel virtualisé                                       | Pas de virtualisation matérielle                          |
| Consommation importante<br>Isolation totale               | Consommation réduite<br>Isolation partielle               |



Conteneurisation  
Docker / Containerd



# LaMeDuSe

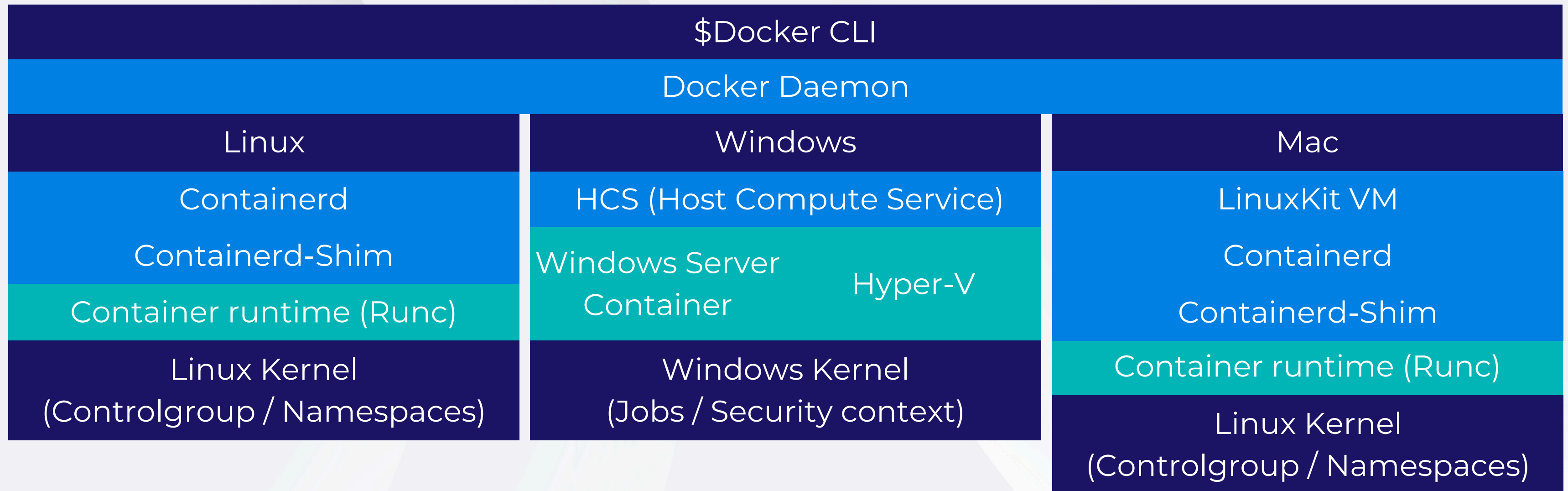
**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**



# Présentation de Docker

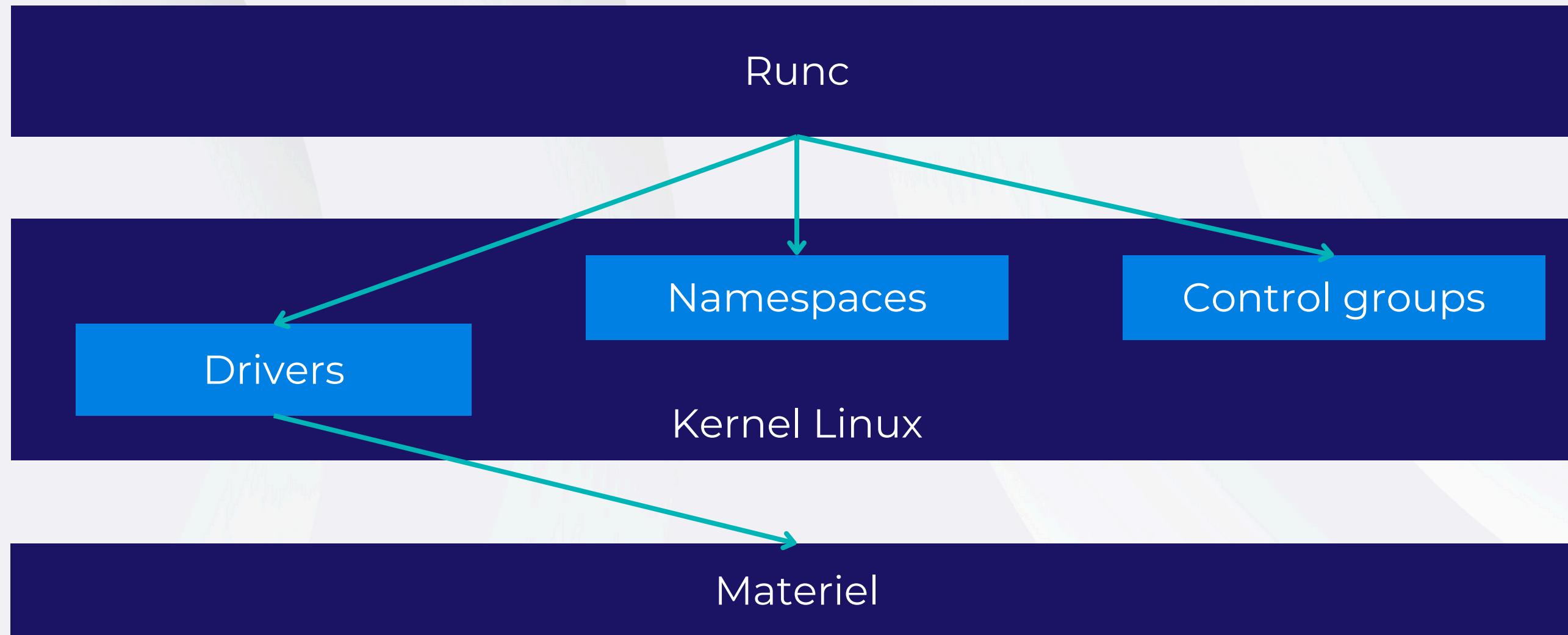
- L'architecture de Docker.
- Disponibilité et installation de Docker sur différentes plateformes (Windows, Mac et Linux).
- Création d'une machine virtuelle pour maquettage.
- La ligne de commande et l'environnement.
- TP: Créer une machine virtuelle pour réaliser un maquettage.

# L'architecture de Docker





# Architecture



# Open Container Initiative

- Produire une interface standard pour l'exploitation de container
  - Docker / Containerd
  - Podman
  - Gvisor
  - Cri-o
- Composant de liaison moteur - kernel
  - Runc
  - Crun

# Installation de Docker (Linux)

1. Activer la virtualisation dans le bios
2. Télécharger et exécuter le script Docker <https://get.docker.com>
3. Activer et lancer le service docker
4. Ajouter l'utilisateur au groupe docker (permet à celui-ci d'interagir avec le service docker en utilisant l'utilitaire docker)

# Installation de Docker (Windows)

1. Activer la virtualisation dans le bios
2. Activer Hyper-V et la plateforme de container dans les fonctionnalités de windows
3. (optionnel) Activer la plateforme Windows Subsystem for Linux (WSL)
4. Télécharger et installer Docker Desktop  
<https://docs.docker.com/desktop/release-notes/>

## Notes :

- Sur les environnements Windows avec WSL vous pouvez exécuter des containers Docker pour Linux.



# Installation de Docker (Mac)

1. Télécharger et installer Docker Desktop

<https://docs.docker.com/desktop/release-notes/>

Notes :

- Docker supporte officiellement les deux dernières versions de MacOS
- Pour les Mac avec processeur Apple il est recommandé d'installer Rossetta 2 (softwareupdate --install-rosetta)

# Création d'une machine virtuelle pour maquettage (Etapes)

1. Activer la virtualisation matérielle (Intel VT-x / AMD-V)
2. Installer l'outil de virtualisation (VMware, VirtualBox, ...)
3. Configurer la machine virtuelle
  - a. Activer la virtualisation (Intel VT-x / AMD-V)
4. Installer le système d'exploitation pour accueillir docker

# La ligne de commande et l'environnement

## Gestion du container

- `run [option] image`  $\Rightarrow$  exécuter une image
- `start [option] nom_du_container`  $\Rightarrow$  démarre un container
- `stop [option] nom_du_container`  $\Rightarrow$  stoppe un container
- `restart [option] nom_du_container`  $\Rightarrow$  redémarre un container
- `kill [option] nom_du_container`  $\Rightarrow$  tue un container
- `exec [option] nom_du_container commande`  $\Rightarrow$  execute une commande dans un container
- `attach [option] nom_du_container`  $\Rightarrow$  attache la console à l'entrée, sortie et sortie d'erreur d'un container
- `container list [option]`  $\Rightarrow$  liste les containers en cours d'exécution
  - (`--all` : listes tout les containers, y compris ceux stoppé)

- 2.7 • `rm [-f] nom_du _container`  $\Rightarrow$  suppression du container





# La ligne de commande et l'environnement

## gestion des images

- `build [option] chemin_dockerfile`  $\Rightarrow$  créer une image docker a partir d'un dockerfile
- `push [option] nom_image:[tag]`  $\Rightarrow$  envoie une image docker a un registre
- `image ls`  $\Rightarrow$  liste les images docker
- `image prune`  $\Rightarrow$  retire toute les images non utilisée
- `image rm [option] nom_image`  $\Rightarrow$  supprime l'image



# La ligne de commande et l'environnement

## gestion des réseaux

- `network create [option] nom_reseau`  $\Rightarrow$  créer un réseau
- `network rm [option] nom_reseau`  $\Rightarrow$  supprime un réseau
- `network prune`  $\Rightarrow$  supprime les réseaux non utilisés
- `network ls`  $\Rightarrow$  list les réseaux
- `network connect [option] nom_reseau nom_container`  $\Rightarrow$  connecte le container a un réseau
- `network disconnect [option] nom_reseau nom_container`  $\Rightarrow$  déconnecte le container d'un réseau

# La ligne de commande et l'environnement

## gestion des volumes

- volume create [option] [nom\_du\_volume]
- volume inspect [option] nom|id
- volume ls
- volume prune
- volume rm [option] nom\_de\_volume

# TP : Créer une machine virtuelle pour réaliser un maquettage

3 - Mise en œuvre en ligne de commande

paul.millet@lameduse.fr



# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**





# Mise en œuvre en ligne de commande

- Mise en place d'un premier conteneur.
- Le Docker hub : ressources centralisées.
- Mise en commun de stockage interconteneur.
- Mise en commun de port TCP interconteneur.
- Publication de ports réseau.
- Le mode interactif.
- TP : Configurer un conteneur en ligne de commande.

## Mise en place d'un premier conteneur.

On execute la commande  
\$docker run hello-world

Containerd vérifie si l'image  
existe localement

Containerd récupère l'image  
du container demandée  
nommée "hello-world"

Lancement du container

```
warstrolo@dev-xubuntu-01:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53cc4d415d839c98be39331c948609b659ed725170ad2ca8eb36951288f81b75
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

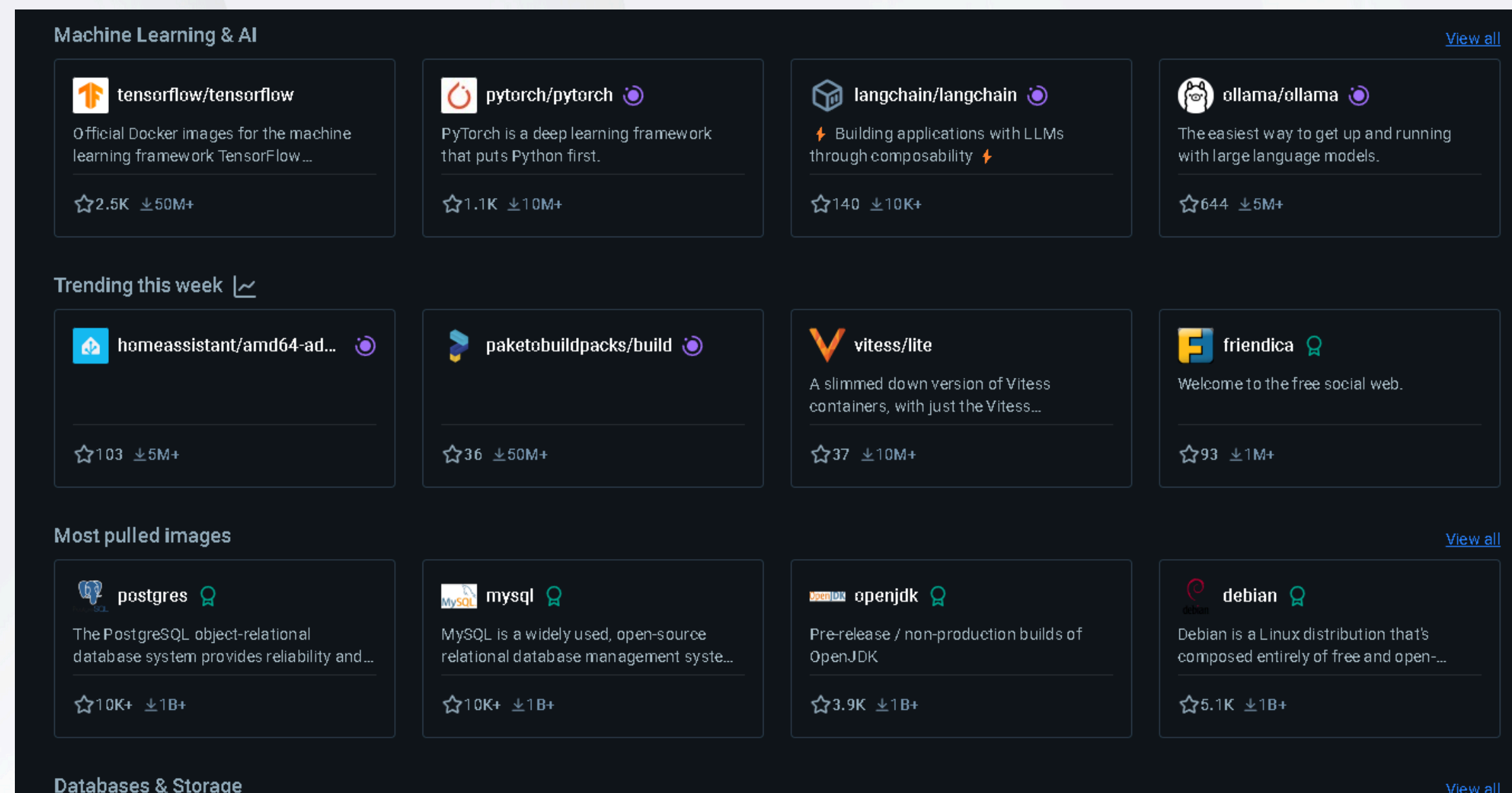
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

# Le Docker hub : ressources centralisées.



- url : <https://hub.docker.com>
- Permet l'accès à une collection d'image faite par la communauté
- Peut héberger vos images sans avoir à déployer un registre soi-même





## Mise en commun de stockage interconteneur.

- Création du volume docker : “random\_volume”
  - `docker volume create random_volume`
- Lancement des premiers container dans deux terminaux séparé
  - `docker run --name writer --volume random_volume:/tmp registry.lameduse.cloud/public/lameduse-ressources-dok_filewrite`
  - `docker run --name reader --volume random_volume:/tmp registry.lameduse.cloud/public/lameduse-ressources-dok_fileread`
- options
  - `--name` : assigne un nom au container, permet de le spécifier après dans les commandes (`rm`, `stop`, `start`)
  - `--volume` : assigne le volume au container et le monte sur “/tmp”
  - `registry.lameduse....filewrite` : nom de l’image que nous utilisons



# Différence entre un volume et un “bind-mount”

- Volume Docker :
  - Stockage géré par docker
  - Chemin (sous linux): /var/lib/docker/volumes/
- Bindmount:
  - Stockage géré par le système hôte
  - Le chemin est celui que vous avez choisi

# Mise en commun de stockage interconteneur.

```
warstrolo@dev-xubuntu-01:~$ docker volume create random_volume
random_volume
```

```
warstrolo@dev-xubuntu-01:~$ docker run --name writer --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_filewrite:latest
Unable to find image 'registry.lameduse.fr/public/lameduse-ressources-dok_filewrite:latest' locally
latest: Pulling from public/lameduse-ressources-dok_filewrite
c6a83fedfae6: Already exists
9ff7791c907d: Pull complete
c9f58c058851: Pull complete
ac275217c340: Pull complete
Digest: sha256:3f449e1c927d2dbf80e37e7d36bb4ed4bfb95c6dc9ded224a51b005aae812f43
Status: Downloaded newer image for registry.lameduse.fr/public/lameduse-ressources-dok_filewrite:latest
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:39:58 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:00 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:02 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:04 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:06 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:08 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:10 UTC 2024
Writing date and time to random_file:
Contents of random file:
Fri Aug 16 13:40:12 UTC 2024
Writing date and time to random_file:
Contents of random file:
```

```
warstrolo@dev-xubuntu-01:~$ docker run --name reader --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_fileread:latest
Unable to find image 'registry.lameduse.fr/public/lameduse-ressources-dok_fileread:latest' locally
latest: Pulling from public/lameduse-ressources-dok_fileread
c6a83fedfae6: Already exists
e68bca6f1f98: Pull complete
9473657d6aaa: Pull complete
46a9cdf9721f: Pull complete
Digest: sha256:63d4985104dac08edd07e2dc64e5aa84cbbf9010f55d1dc3156e6964cbd71e24
Status: Downloaded newer image for registry.lameduse.fr/public/lameduse-ressources-dok_fileread:latest
Reading from random_file:
Fri Aug 16 13:40:28 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:33 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:35 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:37 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:39 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:41 UTC 2024
```

## Mise en commun de port TCP interconteneur

- Création du réseau docker : “random\_network”
  - `docker network create random_network`
- Lancement des premiers containers dans deux terminaux séparé
  - `docker run --name reader --network random_network registry.lameduse.cloud/public/lameduse-ressources-dok_netread`
  - `docker run --name writer -it --network random_network -e TARGET_HOST=reader registry.lameduse.cloud/public/lameduse-ressources-dok_netwrite`
- options
  - `--network`: assigne le réseau au container
  - `-e` : lance le container avec la variable d'environnement
  - `-it` : mode interactif



# Mise en commun de port TCP interconteneur

```
warstrolo@dev-xubuntu-01:~$ docker network create random_network  
4fcfc5b25e96b8f309047e58746d1141f4ce1c2e1453b12f23a54f87f27dcbbc8
```

```
warstrolo@dev-xubuntu-01:~$ docker run --name reader --network random_network registry.lameduse.fr/public/lameduse-ressources-dok_netread  
Server is listening on port 8080  
Received message: test  
  
Received message: hello  
  
Received message: world
```

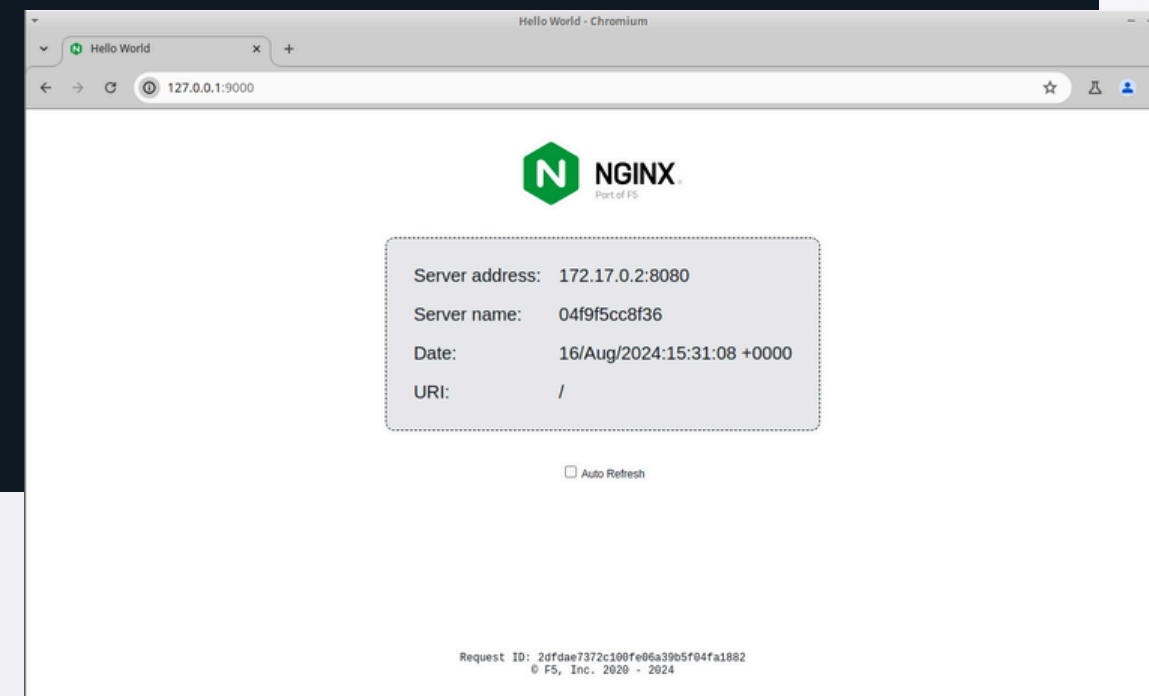
```
cwarstrolo@dev-xubuntu-01:~$ docker run --name writer -it --network random_network -e TARGET_HOST=reader registry.lameduse.fr/public/lameduse-ressources-dok_netwrite  
test  
Response: Message received  
hello  
Response: Message received  
world  
Response: Message received
```



# Publication de port réseau

- `docker run -p 8080:8080 --name nginx nginxdemos/nginx-hello`
- `-p port_hote:port_container` : spécifie la redirection de port

```
warstrolo@dev-xubuntu-01:~$ docker run -p 8080:8080 nginxdemos/nginx-hello
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/08/16 15:28:11 [warn] 1#1: the "user" directive makes sense only if the master process runs with super-user privileges, ignored in /etc/nginx/nginx.conf:2
nginx: [warn] the "user" directive makes sense only if the master process runs with super-user privileges, ignored in /etc/nginx/nginx.conf:2
2024/08/16 15:28:11 [notice] 1#1: using the "epoll" event method
2024/08/16 15:28:11 [notice] 1#1: nginx/1.27.1
2024/08/16 15:28:11 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
2024/08/16 15:28:11 [notice] 1#1: OS: Linux 6.8.0-40-generic
2024/08/16 15:28:11 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/08/16 15:28:11 [notice] 1#1: start worker processes
2024/08/16 15:28:11 [notice] 1#1: start worker process 20
2024/08/16 15:28:11 [notice] 1#1: start worker process 21
2024/08/16 15:28:11 [notice] 1#1: start worker process 22
2024/08/16 15:28:11 [notice] 1#1: start worker process 23
2024/08/16 15:28:11 [notice] 1#1: start worker process 24
2024/08/16 15:28:11 [notice] 1#1: start worker process 25
2024/08/16 15:28:11 [notice] 1#1: start worker process 26
2024/08/16 15:28:11 [notice] 1#1: start worker process 27
```



# Le mode interactif

- Création du container
  - `docker run -p 8080:8080 --name nginx nginxdemos/nginx-hello`
- Utilisation du mode interactif pour interagir avec la console (sh)
  - `docker exec -it nginx /bin/sh`
- Options
  - `-i` : mode interactif permet d'interagir avec la console
  - `-t` : met en place l'interface du terminal (tty)

```
warstrolo@dev-xubuntu-01:~$ docker exec -it nginx /bin/sh
/ $ hostname
1ba71073f06d
/ $ ls
bin          docker-entrypoint.sh  lib          opt          run          sys          var
dev          etc                  media        proc         sbin         tmp
docker-entrypoint.d  home                mnt          root         srv          usr
/ $
```

## Le mode détaché

- Création du container
  - `docker run -d -p 8080:8080 --name nginx nginxdemos/nginx-hello`
- Utilisation du mode détaché qui permet au container de s'exécuter en arrière-plan
- Options
  - `-d` : mode détaché

```
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$ docker run -d -p 8080:8080 --name nginx nginxdemos/nginx-hello
28319a0272d89728edfb0be74bee733481a18ba48ffad94d93e374dbcae7734
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$
```



# TP : Configurer un conteneur en ligne de commande





# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**



# Création de conteneur personnalisé

- Produire l'image de l'état d'un conteneur.
- Qu'est-ce qu'un fichier Dockerfile ?
- Automatiser la création d'une image.
- Mise en œuvre d'un conteneur.
- Conteneur hébergeant plusieurs services : supervisor.
- TP : Créer un container personnalisé.

# Produire l'image de l'état d'un conteneur

- La commande docker commit permet de créer une image à partir de l'état d'un container
- Usage
  - `docker commit [options] nom_container [nom_image][:tag]`

# Qu'est-ce qu'un fichier Dockerfile ?

- Le fichier Dockerfile permet de spécifier les instructions de création d'une image
- Format :
  - # Commentaire
  - INSTRUCTION [argument]
- Les instructions par convention sont écrites en majuscule (bien que pas obligatoire)
- Les instructions sont exécuté dans l'ordre



## Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- ADD [options] <source...> <destination> : Ajoute un fichier local, distant ou un répertoire
  - La source est une archive tar : ADD décompresse l'archive et ajoute son contenu à la destination
  - La source est un répertoire git : clone le répertoire et l'ajoute à la destination
  - La source est une url : télécharge le fichier et l'ajoute à la destination
- ARG <argument> [=valeur\_par\_default] : Permet de passer un argument lors de la création de l'image
  - L'option "--build-arg <argument=valeur>" doit être passé dans les options lors de la commande "docker build"

# Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- CMD : Définie la commande par défaut utilisée pour lancer le container
  - Particularité : Peut-être réécrite lors de l'exécution du container
    - `docker run image [CMD]`
  - Formes
    - CMD ["executable", "argument1", "argumentN"...] : spécifie l'executable
    - CMD ["argument1", "argumentN"] : utilise l'executable par défaut avec les arguments spécifiés
- COPY : Copie la / les source(s) vers la destination
  - Forme : COPY [options] <source...> <destination>

# Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- ENTRYPOINT : Spécifie l'exécutable par défaut
  - Forme : ENTRYPOINT ["executable", "argument1", "argumentN"]
  - Différence avec CMD : Les instructions passées par CMD sont ajoutés après celles spécifiées par ENTRYPOINT
- ENV : Définie une variable d'environnement
  - Forme : ENV <nom>=<valeur>
- EXPOSE : Définie les ports utilisés par le container
  - Note :
    - Ne publie pas les ports par défaut
    - Sans options au moment de l'exécution ne sert à rien d'autre qu'à documenter
    - Avec l'option -P : Docker publie automatiquement les ports



# Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- FROM : Créer une étape de construction à partir d'une image existante
  - Forme : FROM <nom\_image>[:tag] [AS nom\_utilisation]
  - Note :
    - Première instruction donnée lors de la définition du Dockerfile
    - le nom d'utilisation est utilisé lorsque l'image a besoin de plusieurs images différentes pour se construire (par exemple une image qui permet la compilation et une autre image pour l'exécution)



# Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- RUN : Exécute une commande lors de la construction de l'image
  - Formes :
    - RUN [option] <commande>
    - RUN [option] [<commande1>, <commandeN>]
- SHELL : Change la console utilisée par default
  - Forme : SHELL [<executable>, [paramètre]]
  - Note : Cette console est utilisée par l'instruction RUN
- USER : Change l'utilisateur qui exécute les commandes
  - Forme : USER <utilisateur>[:groupe]

# Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- WORKDIR : Change le répertoire d'exécution
  - Forme : WORKDIR <répertoire>
  - Note :
    - Si le répertoire n'existe pas, il est automatiquement créé
    - Définie le répertoire par défaut pour les commandes
      - RUN
      - CMD
      - ENTRYPOINT
      - COPY
      - ADD
    - Si le chemin est relatif, il sera relatif au précédent répertoire d'exécution

# Automatiser la création d'une image

```
# On utilise l'image de base alpine:latest
FROM alpine:latest

# On installe bash
RUN apk add --no-cache bash

# On copie le script "write-file.sh"
COPY write-file.sh /write-file.sh

# On autorise l'exécution du script
RUN chmod +x /write-file.sh

# On définit le script comme la commande d'exécution par default
CMD ["/write-file.sh"]
```

# Mise en œuvre d'un conteneur (construction de l'image)

On exécute la commande “docker build”

1. “docker build <contexte>” : Créer l'image
2. “docker build -t <nom\_image>[:tag] <contexte>” : Créer l'image et la nomme
3. “docker build -f <chemin\_dockerfile> <contexte>” : Créer l'image en utilisant un Dockerfile extérieur à la racine du contexte

Note :

- le contexte est le chemin ou se situe le contexte
  - sans l'option -f celui-ci doit contenir le fichier Dockerfile a la racine

```
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$ docker image list
REPOSITORY                                TAG          IMAGE ID       CREATED        SIZE
<none>                                    <none>       2871c9ff59bb   10 seconds ago 128MB
toto                                       latest       9649558da5ac   3 minutes ago  128MB
registry.lameduse.fr/public/lameduse-ressources-dok filewrite    latest       5b1c0e33ea6c   47 hours ago   9.83MB
```



## Mise en œuvre d'un conteneur (publication de l'image)

On exécute la commande “docker push”

1. “docker push <nom\_image>[:tag]” : envoie l'image vers le registre
  - a. si l'image est nommée sans le registre (e.g. image ou repo/image) elle est envoyée par défaut sur le docker hub (requiert une authentification)
  - b. si l'image est nommée avec le registre (e.g. registry.organisation.com/repo/image) elle est envoyé sur le registre spécifié
2. “docker save -o <chemin\_archive> <nom\_image>[:tag]”
  - a. exporte l'image dans l'archive spécifié

# Mise en œuvre d'un conteneur (récupération de l'image)

On exécute la commande “docker push”

1. “docker pull <nom\_image>[:tag]”
  - a. récupère l'image et l'envoie sur le registre interne de l'hôte docker
2. “docker load -i <chemin\_archive>”
  - a. importe l'image depuis l'archive spécifiée
  - b. garde le nom spécifié lors de l'export

# Une image docker en production

Objectif :

- Toute la compilation a été faite avant
- Temps de démarrage très court

En production (avec une plateforme de type k8s) :

- Besoin de plus de puissance, création automatique de plusieurs containers pour supporter le besoin
- Besoin de moins de puissance, suppression de containers automatique pour réduire la consommation

# TP : Créer un conteneur personnalisé





# LaMeDuSe

**UBE : Kubernetes, mise en œuvre**



# Introduction à Kubernetes

1. De la virtualisation à la conteneurisation - Docker & Kubernetes.
2. Installation de Kubernetes
3. Accéder au cluster Kubernetes
4. Déploiement et publication manuelle
5. Détail et introspection du déploiement
6. TP : Déploiement d'une plateforme de test

# Un orchestrateur c'est quoi ?

- Coordination, gestion et automatisation
  - de la configuration (pré-requis, paramètres...)
  - du déploiement (allocation, provisionnement...)
  - de la maintenance (mise à jour, pannes...)

=> Augmentation de l'efficacité opérationnelle  
+ réduction des erreurs

# Etude de cas : “Borg, L’orchestrateur de Google”

- Google utilisait Borg depuis les années 2000
- Problèmes de Borg
  - Très adapté aux besoins internes / difficile à adapter pour d’autres cas d’usage => Pas d’utilisateur extérieur.
  - Complexité de gestion : Exploitation difficile par des équipes non formée et courbe d’apprentissage raide.
  - Pas / peu standardisée.
  - Conception monolithique : Une base de code pour plusieurs fonctions métier.

Problème : Complexité d’utilisation, de maintenance, et d’évolution.



# Kubernetes : La solution de Google

- Automatisation
  - Standardisation des interfaces
  - Standardisation des composants
  - Open-Source = Utilisation par des entreprises autres que Google
  - Faible courbe d'apprentissage
  - Bonne documentation
  - Séparation des composants : Conception en micro-service
- => Facile à mettre en place et à maintenir

# Kubernetes : Concepts généraux

- Pod = Container de container
  - Un pod possède plusieurs containers
  - Un pod correspond a une instance
- Service = Point de connexion vers un ensemble de pod
  - Un service pointe vers un ensemble de pod et distribue le trafic sur cet ensemble
  - Le service possède une IP et un nom DNS
- Node (point de vue plateforme) = Serveur hébergeant Kubernetes
  - Il peut être virtuel ou physique

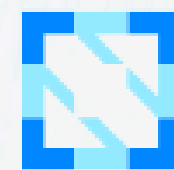
# Qui développe Kubernetes



**Red Hat**



**CoreOS**



**CLOUD NATIVE  
COMPUTING FOUNDATION**

# Solutions d'installation de kubernetes

|              | Développement Local          | Production Cloud Public                                                                 | Production Cloud Privé (3rd party)               | Production Cloud Privé (Sans 3rd party)           |
|--------------|------------------------------|-----------------------------------------------------------------------------------------|--------------------------------------------------|---------------------------------------------------|
| Solutions    | MiniKube                     | Google Kubernetes Engine<br>(AWS) Elastic Kubernetes Engine<br>Azure Kubernetes Service | Rancher Kubernetes Engine 2<br>K3s<br>Open-Shift | KubeAdm                                           |
| Avantages    | Simple d'utilisation         | Simple d'utilisation<br>Support commercial                                              | Simple d'utilisation<br>Support commercial       | Customisable<br>No vendor lock-in                 |
| Inconvénient | Non utilisable en production | Onéreux<br>Vendor Lock-In                                                               | Vendor Lock-in                                   | Complexité d'utilisation<br>Support communautaire |



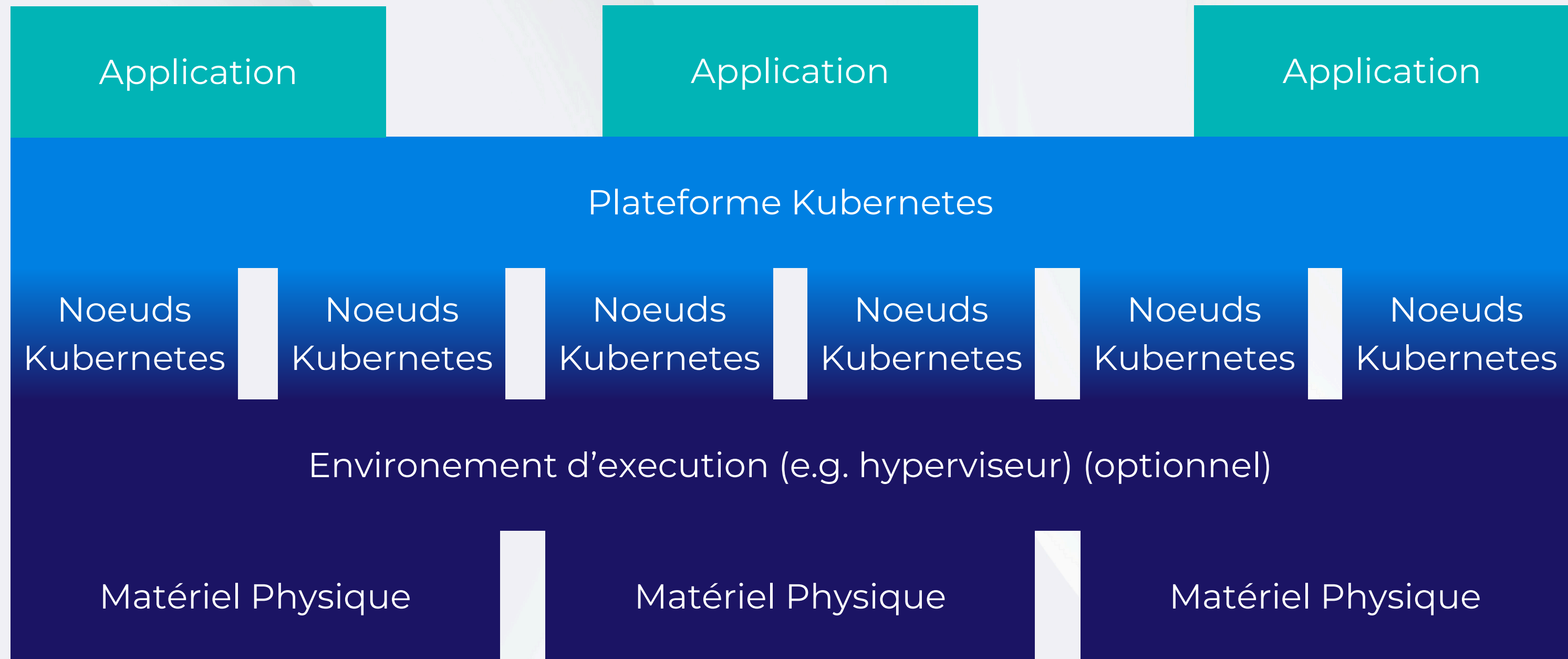
# Installation de Docker

- Dockerd / Containerd
  - Dockerd : moteur de commande de docker
  - Containerd : moteur d'exécution de docker
- Depuis la séparation Dockerd / Containerd
  - Containerd est recommandé

# Moteur de container

- Interface standard CRI (Container Runtime Interface)
  - Containerd : Moteur de Docker
  - CRI-O : Moteur léger développé à la base par Redhat
  - Gvisor : Moteur développé par Google axé sécurité et sandbox

# Niveau d'abstraction vue globale



# Niveau d'abstraction vue physique

Environnement d'exécution (e.g. hyperviseur) (optionnel)

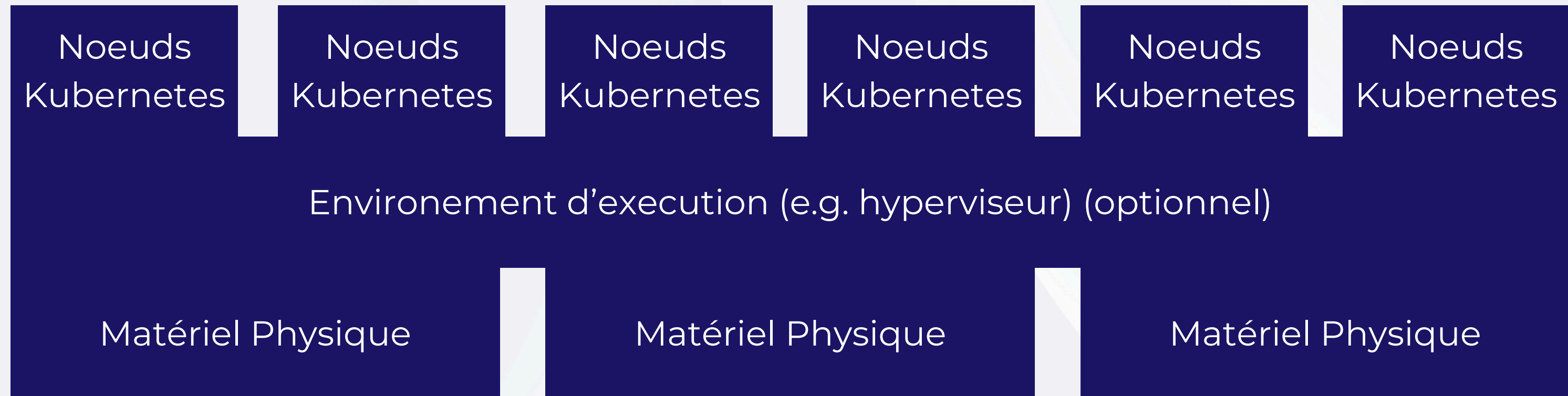
Matériel Physique

Matériel Physique

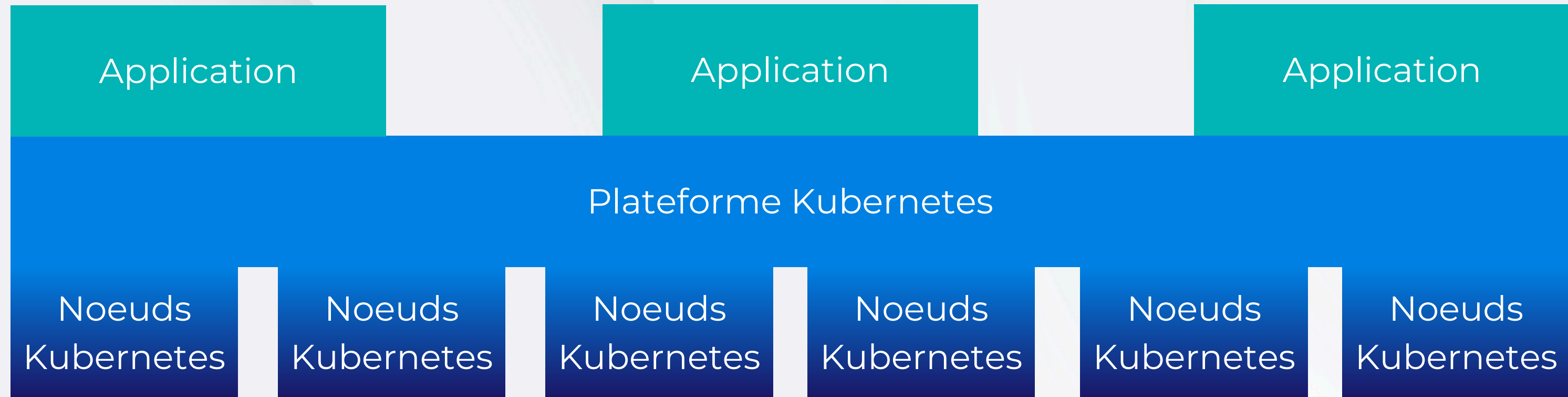
Matériel Physique



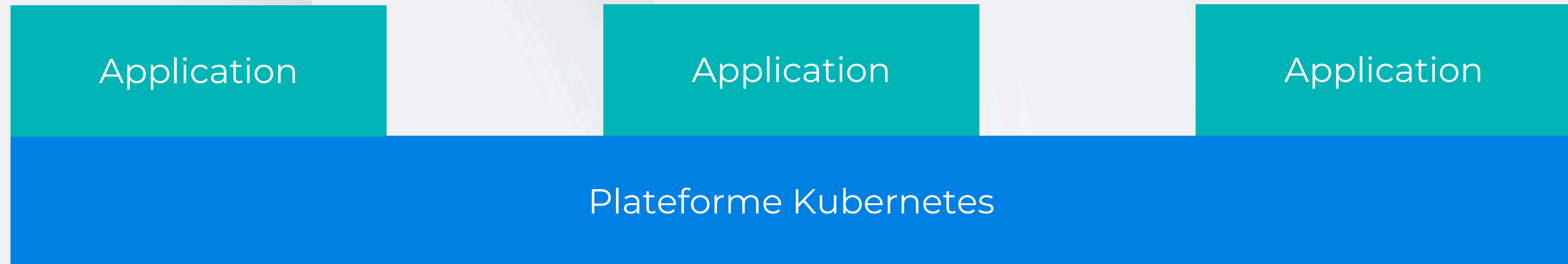
# Niveau d'abstraction vue plateforme d'exécution



# Niveau d'abstraction vue cloud



# Niveau d'abstraction vue applicative



# Accès au cluster kubernetes : CLI (Kubectl)

- kubectl est l'outil en ligne de commande pour interagir avec les clusters Kubernetes.
- Il vous permet de déployer et de gérer des applications sur Kubernetes ainsi que de visualiser et diagnostiquer les ressources et les clusters.



# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl get <type_ressource> [nom_ressource]` :
  - Affiche les informations sur les ressources Kubernetes. Si aucun nom n'est spécifié, toutes les ressources de ce type seront affichées.
  - Exemples :
    - `kubectl get pods` - Liste tous les pods.
    - `kubectl get services` - Liste tous les services.
  - Options :
    - `-o <format>`
      - `yaml` : sortie en yaml
      - `json` : sortie en json
      - `wide` : colonnes étendue
    - `-n <namespace>`

# Accès au cluster kubernetes : CLI (Kubectl)

- kubectl describe <type\_ressource> <nom\_ressource> :
  - Affiche des informations détaillées sur une ressource spécifique, y compris ses événements et son état actuel.
  - Exemple :
    - kubectl describe pod my-pod - Affiche les détails du pod nommé "my-pod".

# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl create -f <fichier_yaml>` :
  - Crée une ressource Kubernetes à partir d'un fichier de configuration YAML.
    - Exemple :
      - `kubectl create -f deployment.yaml` - Crée une ressource à partir du fichier `deployment.yaml`.

# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl apply -f <fichier_yaml>` :
  - Applique les modifications définies dans un fichier YAML en effectuant un strategic merge pour mettre à jour les ressources existantes ou en créer de nouvelles.
  - Exemple :
    - `kubectl apply -f service.yaml` - Applique les configurations du fichier `service.yaml`.



# Accès au cluster kubernetes : CLI (Kubectl)

- kubectl replace -f <fichier\_yaml> :
  - Applique les modifications définies dans un fichier YAML en effectuant un remplacement pour mettre à jour les ressources existantes.
  - Exemple :
    - kubectl replace -f service.yaml - Applique les configurations du fichier service.yaml.

# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl delete <type_ressource> <nom_ressource>` :
  - Supprime une ressource spécifique. Vous pouvez également utiliser un fichier YAML pour supprimer des ressources.
  - Exemples :
    - `kubectl delete pod my-pod` - Supprime le pod nommé "my-pod".
    - `kubectl delete -f deployment.yaml` - Supprime les ressources définies dans le fichier `deployment.yaml`.

# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl logs <nom_pod> [-c <nom_container>]` :
  - Affiche les logs du pod spécifié. Vous pouvez également spécifier un conteneur particulier dans un pod multi-containers.
  - Exemple :
    - `kubectl logs my-pod` - Affiche les logs du pod nommé "my-pod".
    - `kubectl logs my-pod -c my-container` - Affiche les logs du conteneur nommé "my-container" dans le pod "my-pod".



# Accès au cluster kubernetes : CLI (Kubectl)

- `kubectl exec -it <nom_pod> -- <commande>` :
  - Exécute une commande dans un conteneur en cours d'exécution. Utilisez `-it` pour obtenir un terminal interactif.
  - Exemple :
    - `kubectl exec -it my-pod -- /bin/sh` - Ouvre un shell interactif dans le pod nommé "my-pod".
  - Note :
    - Option `--container` pour spécifier le container



# Accès au cluster kubernetes : CLI (Kubectl)

- kubectl cluster-info :
  - Affiche des informations sur les services principaux du cluster, y compris l'URL de l'API et des composants comme le dashboard.
  - Exemple :
    - kubectl cluster-info - Affiche des informations sur le cluster Kubernetes.

# Accès au cluster kubernetes : CLI (Kubectl)

- kubectl config current-context :
  - Affiche le contexte Kubernetes actuellement utilisé. Le contexte détermine le cluster, l'utilisateur et le namespace par défaut.
  - Exemple :
    - kubectl config current-context - Affiche le contexte actuel configuré dans kubectl

# Accès au cluster kubernetes : CLI (K9s)

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: dev  
K8s Rev: v1.17.3  
CPU: 5%  
MEM: 17%

<0> Deployments  
<1> Replicasets  
<2> Statefulsets  
<3> Daemonsets  
<4> Pods  
<5> Events

<6> Jobs  
<7> Persistentvolumes  
<8> Cpu  
<9> Mem

<enter> Goto  
<tab> Next  
<backtab> Prev



Pulses

Deployments - 6/0

Replicasets - 7/0

Statefulsets - 0/0

Daemonsets - 2/0

Pods - 23/0

Events - 16/3

Jobs - 3/0

Persistentvolumes - 0/0

Cpu - 191m

Mem - 1320Mi

<pulses>

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: dev  
K8s Rev: v1.17.3  
CPU: 5%  
MEM: 17%

<esc> Back  
<c> Clear  
<f> FullScreen  
<ctrl-s> Save  
<s> Toggle AutoScroll  
<w> Toggle Wrap



Logs(default/nginx)

Autoscroll: On FullScreen: Off Wrap: Off

2020-02-28T04:21:56.849024713Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.849405412Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.849710094Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.849975726Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.850655441Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.850674065Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.850943668Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.851318883Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.851564522Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.852438245Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.852462723Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.852674971Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.853326657Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.853338282Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.853553185Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.853966201Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.854978974Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.854989669Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.854993863Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.85528333Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.855526606Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.855872693Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.856211169Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.856538277Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.856846529Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.857414857Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

2020-02-28T04:21:56.857513637Z 172.17.0.1 - - [28/Feb/2020:04:21:56 +0000] "GET / HTTP/1.1" 200 612 "-" "hey/0.0.1" "-"

<pulses>

<pod>

<logs>

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: dev  
K8s Rev: v1.17.3  
CPU: 5%  
MEM: 17%

<0> all  
<1> kube-system  
<2> default

<a> Attach  
<ctrl-d> Delete  
<d> Describe  
<e> Edit  
<ctrl-k> Kill  
<l> Logs

<ctrl-j> Logs (jq)  
<ctrl-l> Logs <Stern>  
<shift-l> Logs Previous  
<shift-f> Port-Forward  
<s> Shell  
<y> YAML



Pods(all)[23]

| NAMESPACE↑           | NAME                                       | READY | RESTART | STATUS    | CPU | MEM | %CPU/R | %MEM/R | %CPU/L | %MEM/L | IP            | NODE     |
|----------------------|--------------------------------------------|-------|---------|-----------|-----|-----|--------|--------|--------|--------|---------------|----------|
| default              | hello-1582785780-lsrtd                     | 0/1   | 0       | Completed | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.12   | minikube |
| default              | hello-1582785840-rq8h5                     | 0/1   | 0       | Completed | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.12   | minikube |
| default              | hello-1582785900-4zbkf                     | 0/1   | 0       | Completed | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.12   | minikube |
| default              | jaeger-5bbc8c887-cmjj7                     | 1/1   | 1       | Running   | 0   | 7   | 0      | 3      | 0      | 3      | 172.17.0.11   | minikube |
| default              | nginx                                      | 1/1   | 1       | Running   | 0   | 4   | 0      | 0      | 0      | 0      | 172.17.0.10   | minikube |
| default              | nginx-6fbbdbc48c-5kv5p                     | 1/1   | 0       | Running   | 0   | 2   | 0      | 28     | 0      | 14     | 172.17.0.15   | minikube |
| default              | nginx-6fbbdbc48c-7xn7j                     | 1/1   | 0       | Running   | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.7    | minikube |
| default              | nginx-6fbbdbc48c-bmqqj                     | 1/1   | 0       | Running   | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.13   | minikube |
| default              | nginx-6fbbdbc48c-jf944                     | 1/1   | 0       | Running   | n/a | n/a | n/a    | n/a    | n/a    | n/a    | 172.17.0.12   | minikube |
| default              | nginx-6fbbdbc48c-xwjnb                     | 1/1   | 0       | Running   | 0   | 3   | 0      | 39     | 0      | 19     | 172.17.0.14   | minikube |
| kube-system          | coredns-6955765f44-2pkvx                   | 1/1   | 1       | Running   | 3   | 7   | 3      | 10     | 0      | 4      | 172.17.0.2    | minikube |
| kube-system          | coredns-6955765f44-wr88k                   | 1/1   | 1       | Running   | 3   | 7   | 3      | 10     | 0      | 4      | 172.17.0.3    | minikube |
| kube-system          | etcd-minikube                              | 1/1   | 1       | Running   | 20  | 29  | 0      | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kube-system          | fluentd-elasticsearch-vnt25                | 1/1   | 1       | Running   | 1   | 51  | 1      | 25     | 0      | 25     | 172.17.0.5    | minikube |
| kube-system          | kube-apiserver-minikube                    | 1/1   | 1       | Running   | 47  | 227 | 18     | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kube-system          | kube-controller-manager-minikube           | 1/1   | 2       | Running   | 20  | 35  | 10     | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kube-system          | kube-proxy-sqs9s                           | 1/1   | 1       | Running   | 0   | 14  | 0      | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kube-system          | kube-scheduler-minikube                    | 1/1   | 2       | Running   | 4   | 12  | 4      | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kube-system          | metrics-server-6754dbc9df-t8x2n            | 1/1   | 1       | Running   | 0   | 13  | 0      | 0      | 0      | 0      | 172.17.0.8    | minikube |
| kube-system          | metrics-server-6754dbc9df-tz7kh            | 1/1   | 1       | Running   | 0   | 10  | 0      | 0      | 0      | 0      | 172.17.0.6    | minikube |
| kube-system          | storage-provisioner                        | 1/1   | 2       | Running   | 0   | 14  | 0      | 0      | 0      | 0      | 192.168.64.15 | minikube |
| kubernetes-dashboard | dashboard-metrics-scraper-7b64584c5c-5tjsh | 1/1   | 1       | Running   | 0   | 5   | 0      | 0      | 0      | 0      | 172.17.0.4    | minikube |
| kubernetes-dashboard | kubernetes-dashboard-79d9cd965-wbzxv       | 1/1   | 1       | Running   | 0   | 11  | 0      | 0      | 0      | 0      | 172.17.0.9    | minikube |

<pulses>

<pod>

1.26

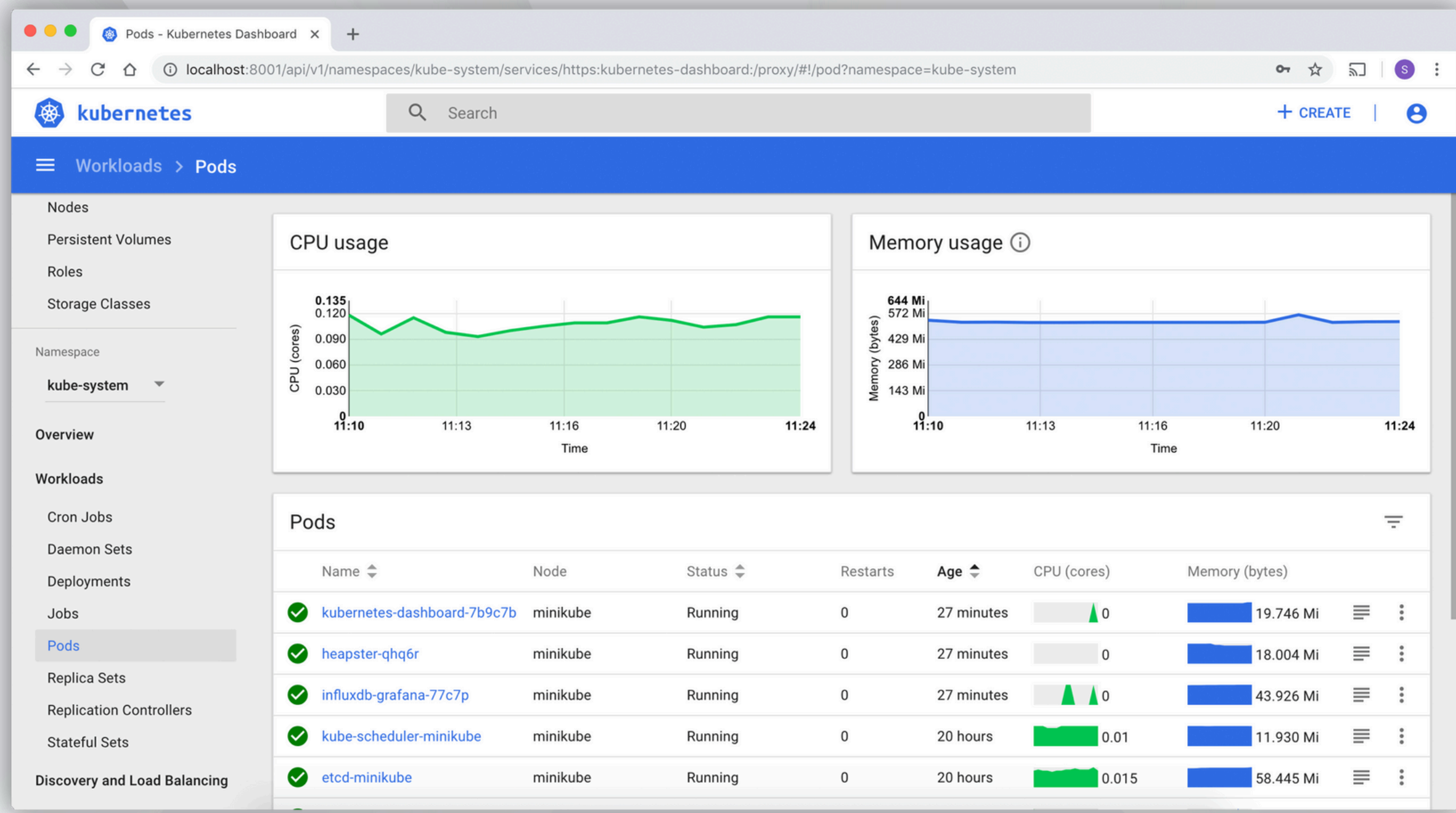


LaMeDuSe

WWW.LAMEDUSE.FR

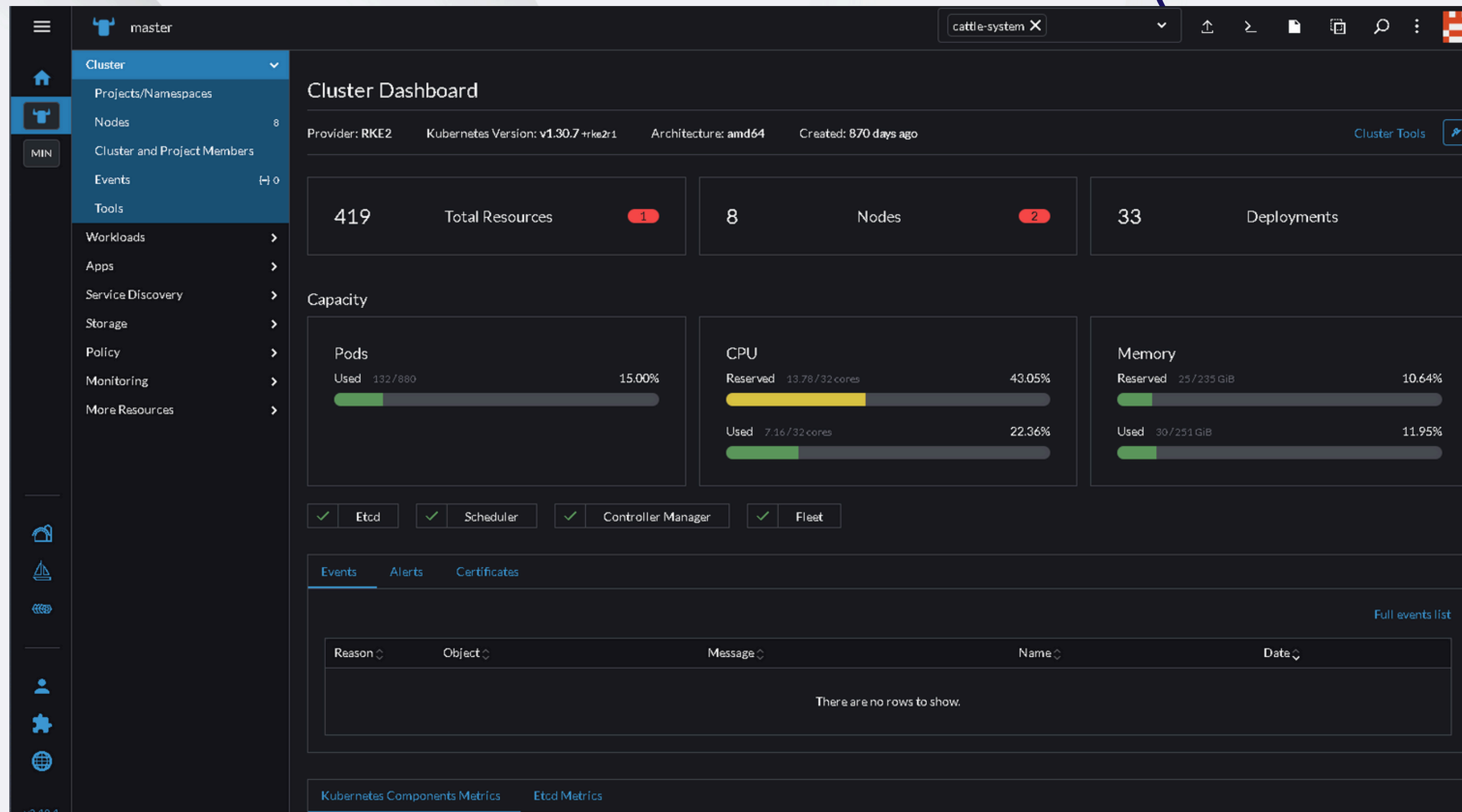


# Accès au cluster kubernetes : GUI (Dashboard)





# Accès au cluster kubernetes : GUI Rancher (Dashboard)



# Création des ressources de base - Déploiement

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: my-app-image:v1
```

# Création des ressources de base - Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

# Scalabilité d'un déploiement. (yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 5 #modifier le nombre de réplicas
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: my-app-image:v1
```



# Scalabilité d'un déploiement. (commande)

Sinon par commande :

- “kubectl scale deployment my-app --replicas=5”

# Les fichiers descriptifs - Déploiement

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: nginx
          ports:
            - containerPort: 80
          env:
            - name: NAME
              value: "value"
```

# Stratégie de mise à jour sans interruption (update/rollback).

- Mise à jour sans interruption :
  - Kubernetes permet de déployer des mises à jour d'applications sans interrompre les services.
  - Cela est réalisé à l'aide de déploiements gérés par des stratégies de mise à jour, garantissant que les utilisateurs continuent d'accéder aux applications pendant les modifications.

# Stratégie de mise à jour sans interruption (update/rollback).

- Rolling Update : Cette stratégie met à jour progressivement les pods avec une nouvelle version de l'application, en remplaçant les anciens pods par de nouveaux, un ou plusieurs à la fois.
  - Avantages : Pas de temps d'arrêt, possibilité de surveiller chaque étape.
  - Fonctionnement :
    - Un nouveau pod est créé avec la mise à jour.
    - Un ancien pod est supprimé.
    - Le processus continue jusqu'à ce que tous les pods soient remplacés.



# Stratégie de mise à jour sans interruption (update/rollback).

- Rollback : Si une mise à jour cause des problèmes, Kubernetes permet de revenir rapidement à une version antérieure du déploiement.
- Pourquoi faire un rollback ? : Corriger une erreur, éviter un downtime prolongé.
- Fonctionnement :
  - Kubernetes garde un historique des déploiements. Un rollback remet la configuration à une version antérieure.

# Stratégie de mise à jour sans interruption (update/rollback).

- Déclencher un rollback :
  - Revenir à la version précédente :
    - `kubectl rollout undo deployment my-app`
  - Rollout vers une version spécifique :
    - Voir les révisions disponibles :
      - `kubectl rollout history deployment my-app`
    - Rollback vers une révision spécifique :
      - `kubectl rollout undo deployment my-app --to-revision=<numéro_de_revision>`
    - Vérifier l'état après rollback :
      - `kubectl get deployment my-app`

◦

# TP: Mise en place d'un environnement de test



# LaMeDuSe

UBE : Kubernetes, mise en œuvre

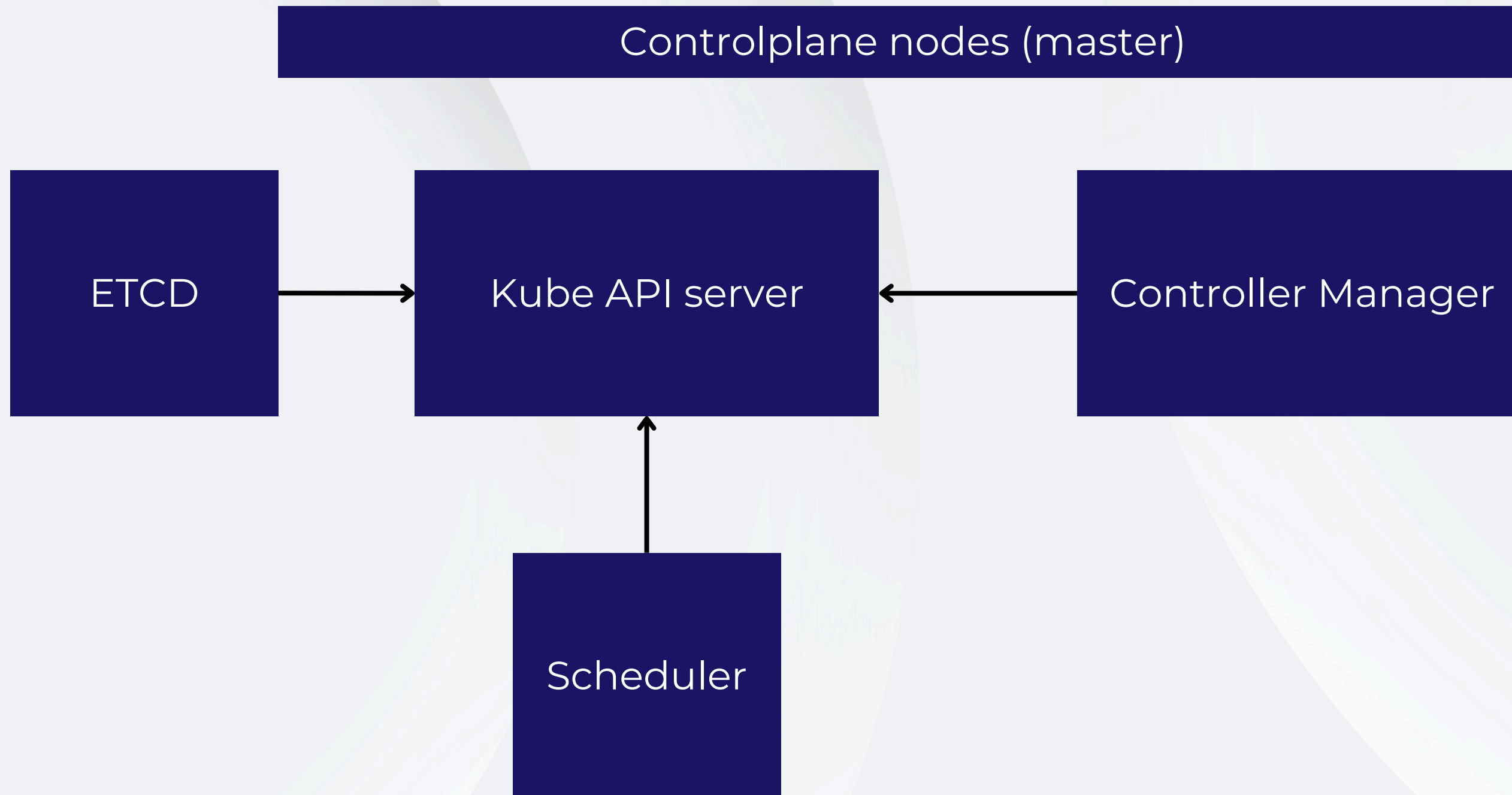




# Architecture Kubernetes

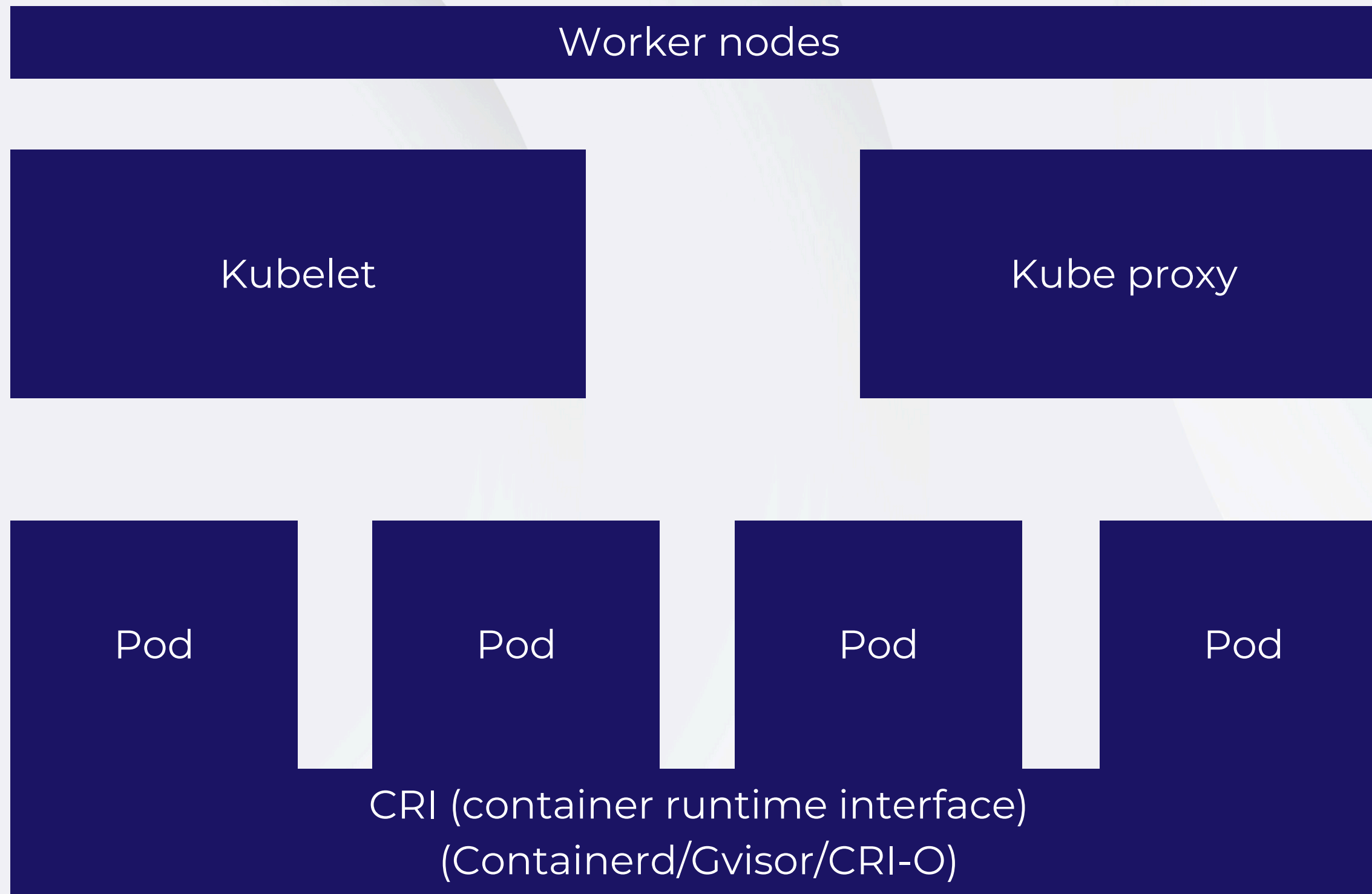
1. Composants du master node : API server, scheduler, controller manager, etc.
2. Architecture d'un noeud : Kubelet, le moteur de conteneur (docker), Kube-proxy.
3. Objets Kubernetes : volume, service, pod, etc.
4. Objet statefull, objet stateless.
5. Solution du deployment.
6. TP : Utilisation de deployment.

# Composants du master node

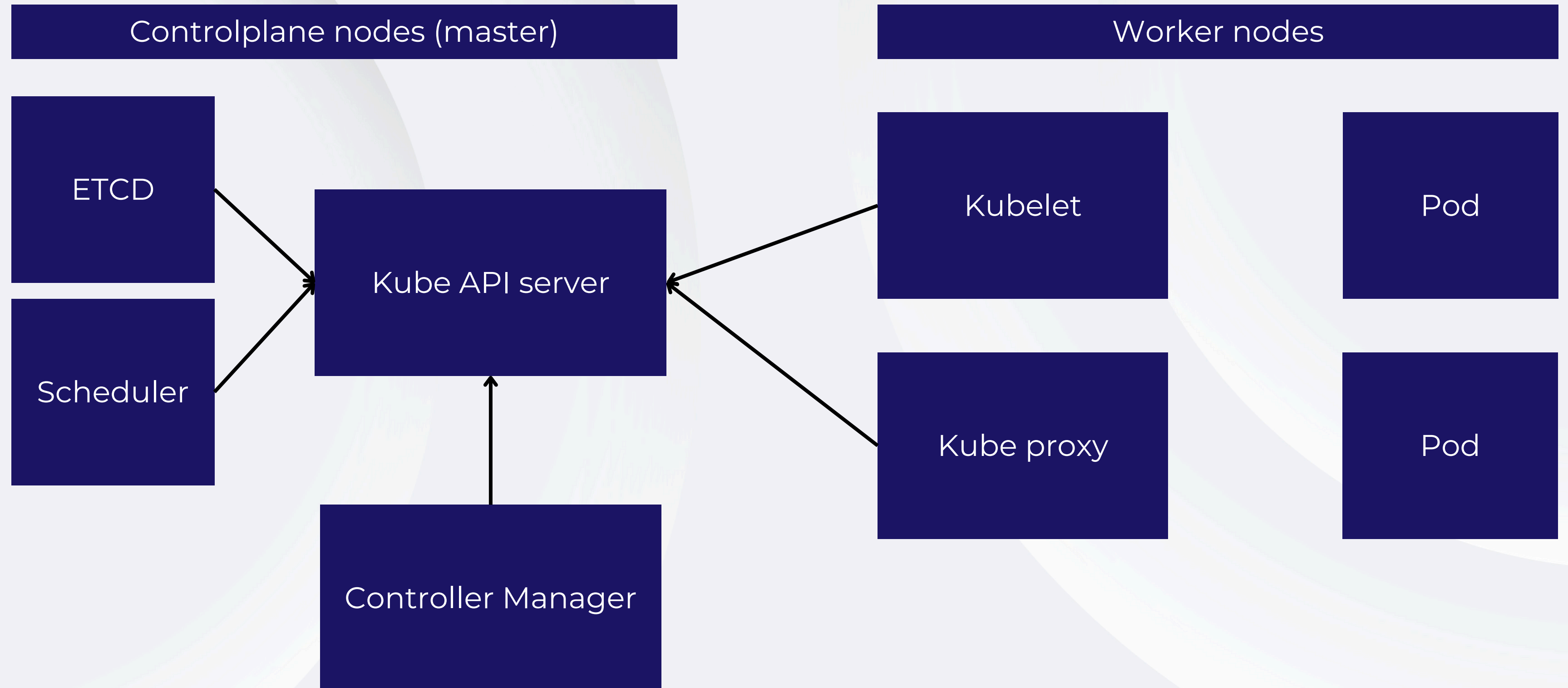


ETCD peut être déployé à l'extérieur du cluster kubernetes ou dans celui-ci

# Architecture d'un noeud



# Architecture Kubernetes





# Accès au cluster kubernetes : APIs

- L'API de Kubernetes est le cœur du système Kubernetes.
- Elle permet aux utilisateurs, applications et outils de communiquer avec le cluster.
- Toutes les interactions avec le cluster, que ce soit via kubectl, des outils tiers ou des services internes, passent par cette API.

# Accès au cluster kubernetes : APIs

- L'API expose différents types de ressources qui représentent des objets dans le cluster, tels que des Pods, Services, Deployments, ConfigMaps, etc.
- Chaque ressource a une représentation en JSON ou YAML, avec des champs spécifiques pour configurer son état et son comportement.

# Accès au cluster kubernetes : APIs

- L'API de Kubernetes est organisée en groupes et versions pour assurer la compatibilité et l'évolution du système.
  - core/v1 : Le groupe API de base contenant les ressources fondamentales (ex : Pods, Services).
  - apps/v1 : Contient des ressources pour les déploiements, les DaemonSets, etc.
  - batch/v1 : Utilisé pour les Jobs et CronJobs.
- Chaque groupe API peut évoluer indépendamment avec des versions comme v1, v1beta1, etc.



# Accès au cluster kubernetes : APIs

L'API de Kubernetes supporte les opérations CRUD (Create, Read, Update, Delete) sur les ressources.

- POST : Créer une nouvelle ressource.
- GET : Lire les détails d'une ressource ou lister des ressources.
- PUT/PATCH : Mettre à jour une ressource existante.
- DELETE : Supprimer une ressource.

Ces opérations sont exécutées via des appels HTTP RESTful.



# Accès au cluster kubernetes : APIs

Les endpoints de l'API Kubernetes sont des URL spécifiques utilisées pour interagir avec les ressources.

- `/api/v1/namespaces/{namespace}/pods` : Lister ou créer des pods dans un namespace donné.
- `/apis/apps/v1/deployments` : Gérer les déploiements.
- `/api/v1/nodes` : Obtenir des informations sur les nœuds du cluster.

Les requêtes peuvent être effectuées en utilisant des outils comme curl, ou à travers des SDKs dans différents langages de programmation.

# Accès au cluster kubernetes : APIs

L'accès à l'API Kubernetes est protégé par des mécanismes d'authentification et d'autorisation.

- Authentification : Via des tokens, certificats client ou OAuth.
- Autorisation : Basée sur des rôles (RBAC - Role-Based Access Control) qui définissent quelles actions un utilisateur ou un service peut effectuer sur quelles ressources.

# Accès au cluster kubernetes : APIs

- Les labels sont des paires clé-valeur attachées aux ressources Kubernetes.
- Ils permettent de filtrer et de sélectionner des ensembles de ressources via des sélecteurs (label selectors).
- Exemple : Sélectionner tous les pods avec le label app=frontend.
  - Endpoint : `/api/v1/namespaces/{namespace}/pods?labelSelector=app=frontend`.
- Les labels sont essentiels pour la gestion des déploiements, la mise en place de services, et l'organisation des ressources



## Accès au cluster kubernetes : APIs

- Les annotations sont similaires aux labels, mais servent à attacher des métadonnées non modifiables par les systèmes internes de Kubernetes.
- Elles sont utilisées pour stocker des informations supplémentaires sur les ressources, comme des traces, des configurations ou des informations liées à des outils tiers.



# Objets Kubernetes : volume, service, pod, etc.

- Pod
  - Unité de base dans Kubernetes, représentant un ou plusieurs conteneurs qui partagent des ressources (stockage, réseau).
  - Utilisé pour exécuter des applications conteneurisées.
- Deployment
  - Gère des ensembles de pods répliqués, permettant des mises à jour sans interruption et le scaling.
  - Fournit des fonctionnalités comme les rolling updates et les rollbacks.

# Objets Kubernetes : volume, service, pod, etc.

- Service
  - Expose les pods pour permettre la communication réseau, interne ou externe.
  - Types principaux : ClusterIP, NodePort, LoadBalancer.
- Namespace
  - Permet de diviser les ressources d'un cluster en environnements logiques isolés.
  - Utilisé pour organiser et gérer des applications distinctes ou des équipes.

# Objets Kubernetes : volume, service, pod, etc.

- ConfigMap
  - Stocke des configurations sous forme de paires clé-valeur, indépendamment du code d'application.
  - Injecte des configurations dans les pods via des variables d'environnement ou des volumes.
- Secret
  - Stocke des informations sensibles comme des mots de passe, des clés SSH, des certificats.
  - Similaire à ConfigMap mais sécurisé (base64 encodé).



# Objets Kubernetes : volume, service, pod, etc.

- PersistentVolume (PV)
  - Représente un stockage physique dans le cluster, utilisé pour persister les données des pods.
  - Exemples : stockage local, disques réseau, NFS, etc.
- PersistentVolumeClaim (PVC)
  - Demande de stockage faite par les utilisateurs pour consommer des PersistentVolumes.
  - Les pods utilisent les PVC pour accéder au stockage.



# Objets Kubernetes : volume, service, pod, etc.

- ServiceAccount
  - Fournit une identité pour les pods afin d'interagir avec l'API Kubernetes.
  - Utilisé pour limiter les permissions d'accès aux ressources du cluster.
- Role et RoleBinding
  - Role : Définit des permissions spécifiques (lecture, écriture) sur des ressources dans un namespace.
  - RoleBinding : Associe un rôle à un utilisateur ou un groupe.

# Objets Kubernetes : volume, service, pod, etc.

- ClusterRole et ClusterRoleBinding
  - ClusterRole : Similaire à Role mais s'applique à l'ensemble du cluster.
  - ClusterRoleBinding : Associe un ClusterRole à des utilisateurs ou groupes à l'échelle du cluster.

# Objets Kubernetes : volume, service, pod, etc.

- ReplicaSet
  - Assure un nombre spécifié de répliquas de pods en cours d'exécution à tout moment.
  - Géré principalement par des déploiements.
- DaemonSet
  - Exécute un pod sur chaque nœud du cluster (ou sous-ensemble de nœuds).
  - Utilisé pour des services de cluster comme les agents de surveillance ou de logging.



# Objets Kubernetes : volume, service, pod, etc.

- StatefulSet
  - Gère des applications nécessitant un état stable ou un identifiant réseau unique.
  - Utilisé pour des bases de données, des caches, ou des applications à état.
- Job et CronJob
  - Job : Exécute une tâche jusqu'à sa complétion (pods temporaires).
  - CronJob : Planifie des Jobs à exécuter à des intervalles réguliers (similaire aux cron jobs Linux).



# Objet statefull, objet stateless

- Stateful : Applications qui maintiennent un état ou des données persistantes entre les sessions ou les redémarrages.
  - Base de donnée, stockage de donnée
- Stateless : Applications qui ne conservent pas d'état entre les requêtes et peuvent être recréées ou déplacées sans impact.
  - Serveur web, serveur applicatif

## Solution du deployment.

- Deployment : gère la création et la mise à jour de groupes de pods sans état (stateless).
- StatefulSet : utilisé pour les applications stateful qui nécessitent des identités stables et un stockage persistant.
- DaemonSet : garantit que tous (ou certains) nœuds exécutent une copie d'un pod donné.
- ReplicaSet : assure un nombre spécifique de pods en cours d'exécution à tout moment.
- Job : utilisé pour les tâches courtes et exécutées une seule fois (one-off).
- CronJob : planifie des tâches périodiques ou à exécution unique basées sur des expressions cron.

## Types de services.

- ClusterIP (par défaut)
  - Usage : Expose le service à l'intérieur du cluster, accessible uniquement par d'autres services ou pods.
  - Utilisation courante : Communication interne entre les microservices.

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-clusterip
spec:
  type: ClusterIP
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
```

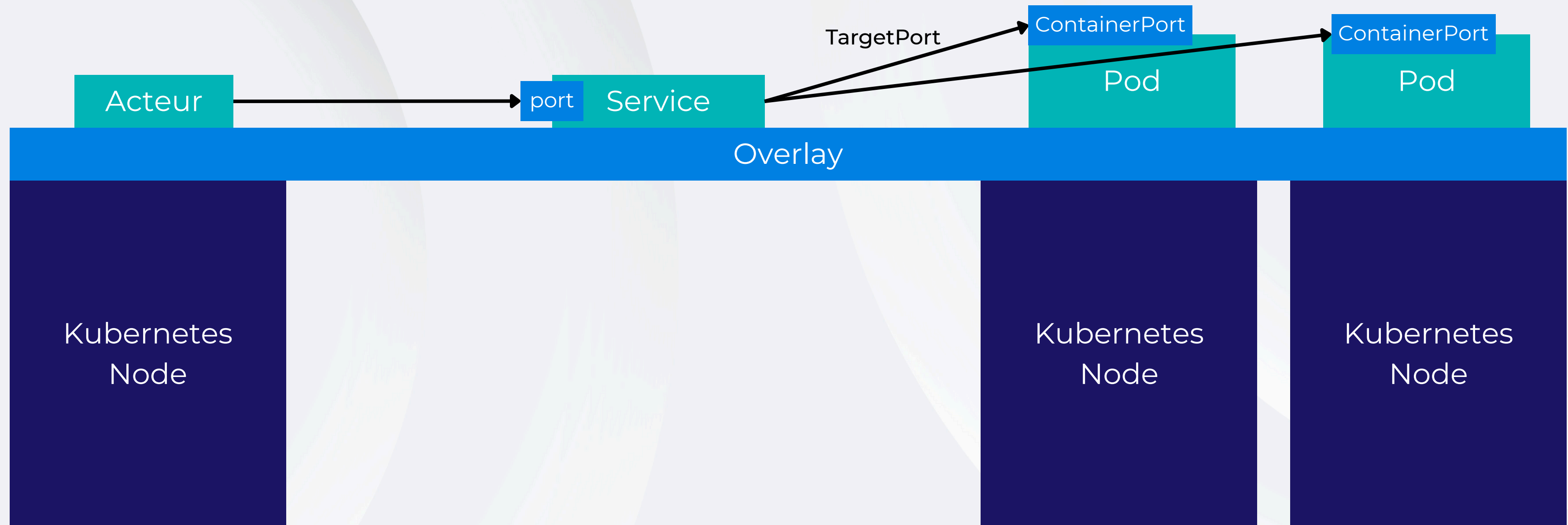
Service = configuration de la couche réseau

Acteur = initiateur de connexion

TargetPort doit être égal à ContainerPort

## Types de services.

- ClusterIP (par défaut)





## Types de services.

- Headless (ClusterIP sans IP)
  - Un service headless est un service Kubernetes qui n'a pas d'adresse IP virtuelle (clusterIP: None). Contrairement à un service classique, il ne fait pas de load balancing ni de proxy entre les clients et les Pods.

Usage : découverte de pods, clusterisation des pods (statefulset), communication directe.

```
apiVersion: v1
kind: Service
metadata:
  name: my-headless-service
spec:
  clusterIP: None # <--- devient un svc headless
  selector:
    app: my-app
  ports:
    - name: http
      port: 80
      targetPort: 8080
```

## Types de services.

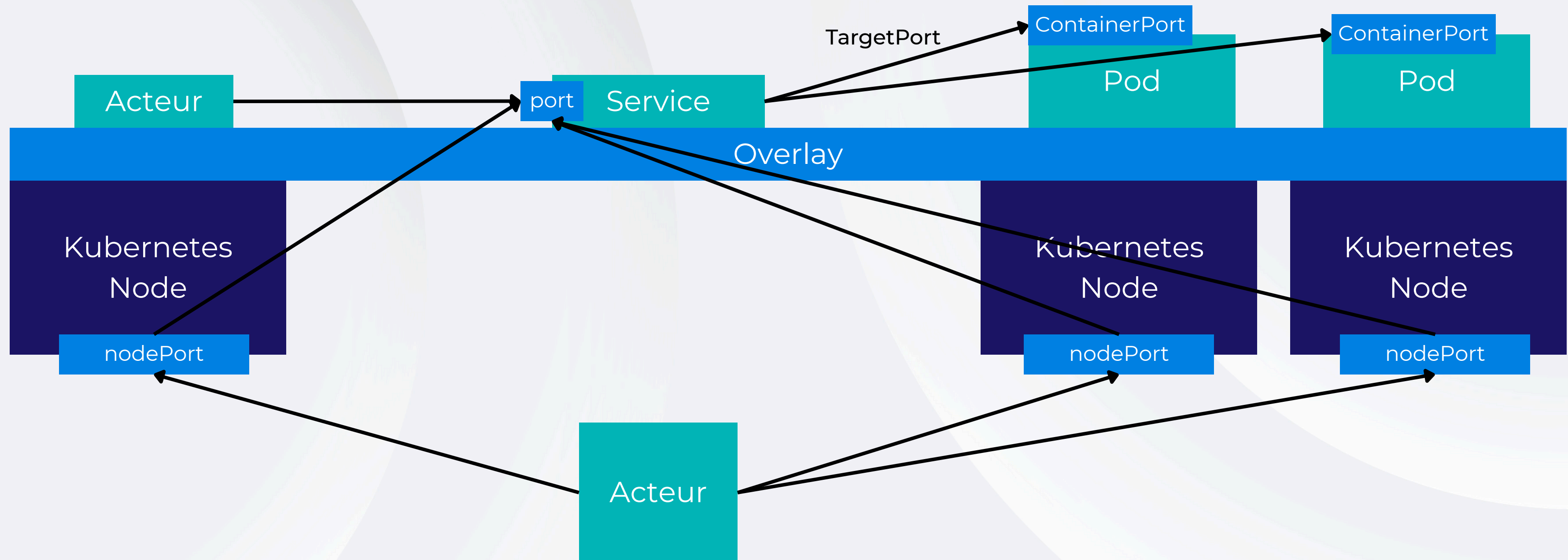
- NodePort
  - Expose le service sur une IP de chaque nœud du cluster, accessible via un port statique (port du nœud).
  - Exposer des services à l'extérieur du cluster pour le développement ou des tests.

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-nodeport
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30007 # Port entre 30000 et 32767
```

nodePort écouté par le kubeproxy  
choisi par vous (si défini) ou par kubernetes sinon

## Types de services.

- NodePort



## Types de services.

- ExternalName
  - Mappe un service à un nom DNS externe, redirige les requêtes vers des services en dehors du cluster.
  - Accéder à des services externes (bases de données, API externes).

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-externalname
spec:
  type: ExternalName
  externalName: my.external.service.com
```



## Types de services.

- LoadBalancer
  - Service qui as pour vocation d'être utiliser avec un composant externe pour rentrer du trafic (une gateway)

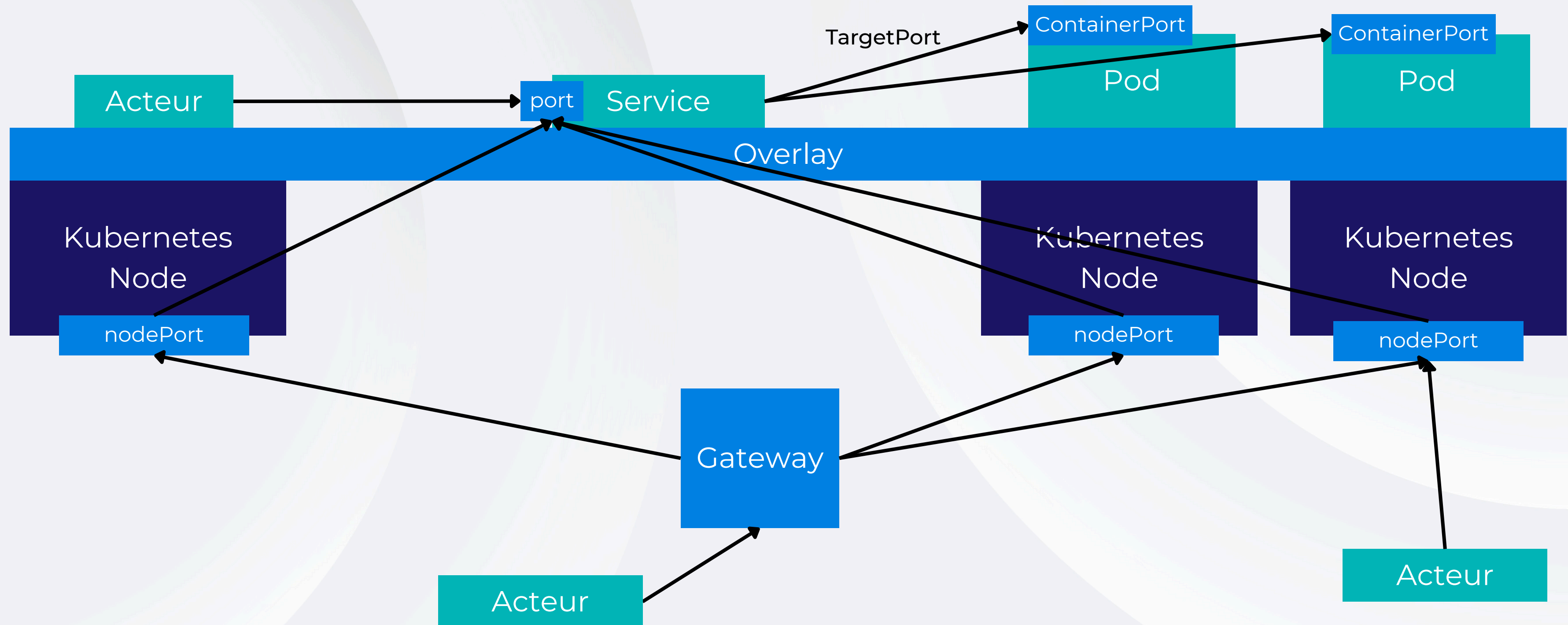
```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80      # exposed port
      targetPort: 5678 # container port
```

Attention : Service non fonctionnel sans implémentation externe

## Types de services.

- LoadBalancer

Le “service provisioner” est alerté qu’un service loadbalancer a été créé et envoie un signal à la gateway





# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**



# Ressources des TP

- 1.TP : Créer une machine virtuelle pour réaliser un maquettage
- 2.TP : Configurer un conteneur en ligne de commande
- 3.TP : Créer un conteneur personnalisé
- 4.TP : Mettre en œuvre une application multiconteneur
- 5.TP : Construire et utiliser son propre registre
- 6.TP : Administrer les conteneurs
- 7.TP : Création d'un cluster Swarm.
- 8.TP : Gestion des nœuds dans le cluster.
- 9.TP : Publication et mise à l'échelle d'un service dans le cluster.



# 1- TP: Créer une machine virtuelle pour réaliser un maquettage

Sujet 1 : Mise en place d'une machine virtuelle pour accueillir docker

1. Création de la VM sous VMware Workstation
  - a. Utiliser l'ISO Alma Linux
  - b. Allouer 20 go de disque
2. Paramétrage de la machine virtuelle
  - a. Allouer 4 go de ram
  - b. Activer la virtualisation Intel VT-x / AMD-V
3. Installation de Docker
  - a. Télécharger le script d'installation
  - b. Démarrer et activer le service docker

## 2- TP: Configurer un conteneur en ligne de commande

Sujet 1 : Exécuter les deux containers (filewriter & filereader)

1. Création du volume partagé
2. Exécution du container fileread
3. Exécution du container filewrite

Notes :

- [registry.lameduse.cloud/public/lameduse-ressources-dok\\_filewrite](https://registry.lameduse.cloud/public/lameduse-ressources-dok_filewrite)
- [registry.lameduse.cloud/public/lameduse-ressources-dok\\_fileread](https://registry.lameduse.cloud/public/lameduse-ressources-dok_fileread)

## 2- TP: Configurer un conteneur en ligne de commande

Sujet 1bis : Exécuter les deux containers (netwriter & netreader)

1. Création du réseau partagé
2. Exécution du container netread
3. Exécution du container netwrite

Notes :

- `registry.lameduse.cloud/public/lameduse-ressources-dok_netwrite`
- `registry.lameduse.cloud/public/lameduse-ressources-dok_netread`

## 2- TP: Configurer un conteneur en ligne de commande

Sujet 2 : Exécuter nginxdemos/hello

### Objectifs

- Le container doit être accessible sur `http://localhost:9000`
- Port interne du container : 80



## 2- TP: Configurer un conteneur en ligne de commande

Sujet 3 : Exécuter nginxdemos/hello (avancé)

### Objectifs

- Le container doit être accessible sur `http://localhost:9000`
- Interdiction d'utiliser la publication de port “-p”

## 2- TP : Configurer un conteneur en ligne de commande

Sujet 3 Solution : Exécuter nginxdemos/hello (avancé)

### Objectifs

- Le container doit être accessible sur `http://localhost:9000`
- Interdiction d'utiliser la publication de port “-p”

### Solutions

- Solution 1 :
  - L'option `--network` contient des réseaux prédéfinis comme “host” qui permet au container d'utiliser directement le réseau de l'hôte
- Solution 2 :
  - Utilisation de l'option `-P` pour automatiquement publier les ports exposés

## 3- TP : Créer un conteneur personnalisé

Sujet 1 : Créer un container d'une application ReactJS

### Objectifs

- Cloner le repo git (<https://github.com/LaMeDuSe/lameduse-ressources.git>)
- Aller dans le répertoire “dok\_cra\_example”
- Créer le fichier Dockerfile (Ou utiliser celui existant)
  - Lire le readme dans le répertoire (il contient des informations pour le faire)
- Construire l'image sous le nom de “application\_react” avec le tag “demo”
- Lancer le container et publier le port 8080 (port interne : 80)



## 3- TP : Créer un conteneur personnalisé

### Sujet 2 : Créer le Dockerfile d'une application Reactjs

#### Objectifs

- Créer le fichier Dockerfile du sujet 1
  - image de build : "node:18"
  - image d'exécution : "nginx:alpine"
  - COPY --from=nom\_image <source> <destination> : copie des fichiers d'une image à une autre
  - Commande nginx : "nginx -g daemon off;"



# 4- TP : Mettre en œuvre une application multiconteneur

## Sujet 1 : Créer un docker compose pour wordpress

### Notes

- variable pour la base de donnée
  - MYSQL\_ROOT\_PASSWORD=somewordpress
  - MYSQL\_DATABASE=wordpress
  - MYSQL\_USER=wordpress
  - MYSQL\_PASSWORD=wordpress
- variable pour le serveur wordpress
  - WORDPRESS\_DB\_HOST=db
  - WORDPRESS\_DB\_USER=wordpress
  - WORDPRESS\_DB\_PASSWORD=wordpress
  - WORDPRESS\_DB\_NAME=wordpress
- Images
  - mariadb:10.6.4-focal
  - wordpress:6.6.1-php8.2-apache

## 4- TP : Mettre en œuvre une application multiconteneur

Sujet 1 (solution) : Créer un docker compose pour wordpress

Solution :

- [https://github.com/LaMeDuSe/lameduse-ressources/tree/main/dok\\_docker\\_compose\\_wordpress](https://github.com/LaMeDuSe/lameduse-ressources/tree/main/dok_docker_compose_wordpress)

# 5- TP : Construire et utiliser son propre registre

Sujet 1 : Mettre en place son propre registre

Objectif :

- Exécuter l'image registry:2
- Publier le port 5000
- Construire et envoyer une image sur le registre

## 6- TP : Administrer les conteneurs

Sujet 1 : Mettre en place un environnement wordpress en utilisant un bindmount

Objectif :

- Créer le docker-compose
- Utiliser un bind-mount pour stocker les données de la base de donnée
- Utiliser la commande docker compose down -v



## 6- TP : Administrer les conteneurs

### Sujet 2 : Déployer fileread et filewrite avec Docker Desktop

Objectif :

- Re-déployer Fileread et Filewrite en utilisant Docker Desktop

## 6- TP : Administrer les conteneurs

### Sujet 3 : Déployer netread et netwrite avec Docker Desktop

Objectif :

- Re-déployer netread et netwrite en utilisant Docker Desktop

# 7- TP : Création d'un cluster Swarm.

## Sujet 1 : Créer un cluster Docker Swarm

Objectif :

- Créer un nœud maître docker swarm
- Ajouter un nœud de travail
- Profiter de votre nouveau cluster !

## 7- TP : Création d'un cluster Swarm.

Sujet 1 (solution) : Créer un cluster Docker Swarm

Objectif :

- Créer un nœud maître docker swarm
  - `docker swarm init`
- Ajouter un nœud de travail
  - `docker swarm join-token worker`
  - (sur le nœud à ajouter) “`docker swarm join <token> <hôte>`”
- Profiter de votre nouveau cluster !



## 8- TP : Gestion des nœuds dans le cluster

Sujet 1 : Mettre en pause un nœud de travail

Objectif :

- Mettre en pause un nœud avec la commande
  - “docker node update <node>”

## 8- TP : Gestion des nœuds dans le cluster

Sujet 1 (solution) : Mettre en pause un nœud de travail

Objectif :

- Mettre en pause un nœud avec la commande
  - “docker node update --availability pause <node>”

Note :

- Si vous avez regardé la documentation, vous avez vu les options suivantes
  - Available : nœud disponible, prêt à recevoir de la charge
  - Pause : le nœud ne prend plus de nouveau service
  - Drain : le nœud est vidé de tout les services

## 9- TP : Publication et mise à l'échelle d'un service dans le cluster

Sujet 1 : Créer un déploiement nginx et lui ajouter 2 répliquas

Objectif :

- Déployer un service nginxdemos/hello
  - publier le port 9000
- Ajouter 2 répliquas au service

Question :

- Avez vous rencontré une erreur ? Si oui pourquoi ?

## 9- TP : Publication et mise à l'échelle d'un service dans le cluster

Sujet 1 (solution) : Créer un déploiement nginx et lui ajouter 2 répliquas

Objectif :

- Déployer un service nginxdemos/hello
  - publier le port 9000
  - commande : “docker service create [--name <nom>] <image>”
- Ajouter 2 répliquas au service
  - commande : “docker service scale <nom=NB\_replicas>”

Question :

- Avez vous rencontré une erreur ? Si oui pourquoi ?



## 0.4- TP : Additions

### Sujet 1 : Déployer apache + php

Objectif :

- Déployer apache + php dans une machine virtuelle debian
- Déployer apache + php dans un container docker

```
<!DOCTYPE html>
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

## 0.7- TP : Additions

Sujet 2 : Déployer portainer + une base de donnée

Objectif :

- déployer une instance portainer
- deployer une base de données en utilisant portainer

## 0.8- TP : Additions

Sujet 3 : Déployer deux instance WordPress avec docker swarm

Objectif :

- déployer une instance swarm
- deployer une instance traefik
- deployer une instance WordPress dont le titre sera blog
- deployer une instance WordPress dont le titre sera site principal
- Configurer l'utilisation de traefik sur les deux instance



# LaMeDuSe

**DOK : Docker, créer et administrer  
ses conteneurs virtuels d'applications**





# Sources

Sources :

- <https://github.com/containerd/containerd/blob/main/core/runtime/v2/README.md>

## Q&A

Q: A quoi corresponds le contexte ?

A: Le contexte est l'endroit où se situent tous les fichiers de votre application nécessaires à la construction de votre container.

Le contexte est un répertoire qui souvent contient à sa racine un fichier dockerfile.

Q: Où se situent les volumes ?

A: `/var/lib/docker/volumes`

# Docker Desktop + Ubuntu 24.04

Fix: `sudo sysctl -w kernel.apparmor_restrict_unprivileged_userns=0`

## Instruction Docker avancée

- HEALTHCHECK Check a container's health on startup.
- LABEL Add metadata to an image.
- MAINTAINER Specify the author of an image.
- ONBUILD Specify instructions for when the image is used in a build.
- STOPSIGNAL Specify the system call signal for exiting a container.
- VOLUME Create volume mounts.



# Création d'un fichier YAML de configuration (Options)

```
version: '3.8' # Version du format de fichier Docker Compose.
services:
  app:
    image: nginx:latest # Spécifie l'image Docker à utiliser pour le conteneur.
    build:
      context: ./app # Répertoire pour la construction de l'image.
    command: "nginx -g 'daemon off;'" # Commande à exécuter dans le conteneur.
    environment:
      - DEBUG=true # Variable d'environnement pour le conteneur.
    ports:
      - "8080:80" # Mappage des ports (hôte:conteneur).
    volumes:
      - app-data:/usr/share/nginx/html # Volume monté dans le conteneur.
    networks:
      - webnet # Réseau auquel se connecter.
    restart: always # Politique de redémarrage du conteneur.
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost/"] # Test de la santé du service.
    logging:
      driver: json-file # Pilote de journalisation.
    user: "1000:1000" # Utilisateur pour exécuter le conteneur.
    working_dir: /usr/src/app # Répertoire de travail dans le conteneur.
  db:
    image: postgres:13 # Image Docker pour le service de base de données.
    environment:
      POSTGRES_PASSWORD: example # Mot de passe pour PostgreSQL.
    volumes:
      - db-data:/var/lib/postgresql/data # Volume pour les données de la base de données.
    networks:
      - webnet # Réseau auquel se connecter.
# Définition des volumes utilisés par les services
volumes:
  app-data: # Volume pour l'application.
  db-data: # Volume pour la base de données.
# Définition des réseaux personnalisés
networks:
  webnet: # Réseau pour la communication entre services.
    driver: bridge # Pilote de réseau.
```