

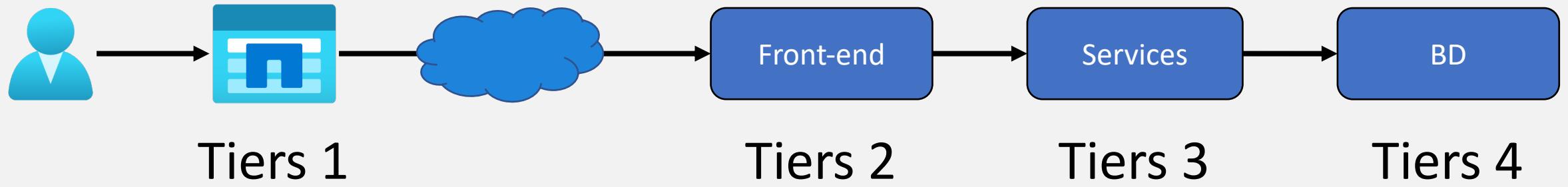
Conteneur



Objectifs

- Modèles de déploiement
- Comprendre l'idée des conteneurs
- Introduction à Docker

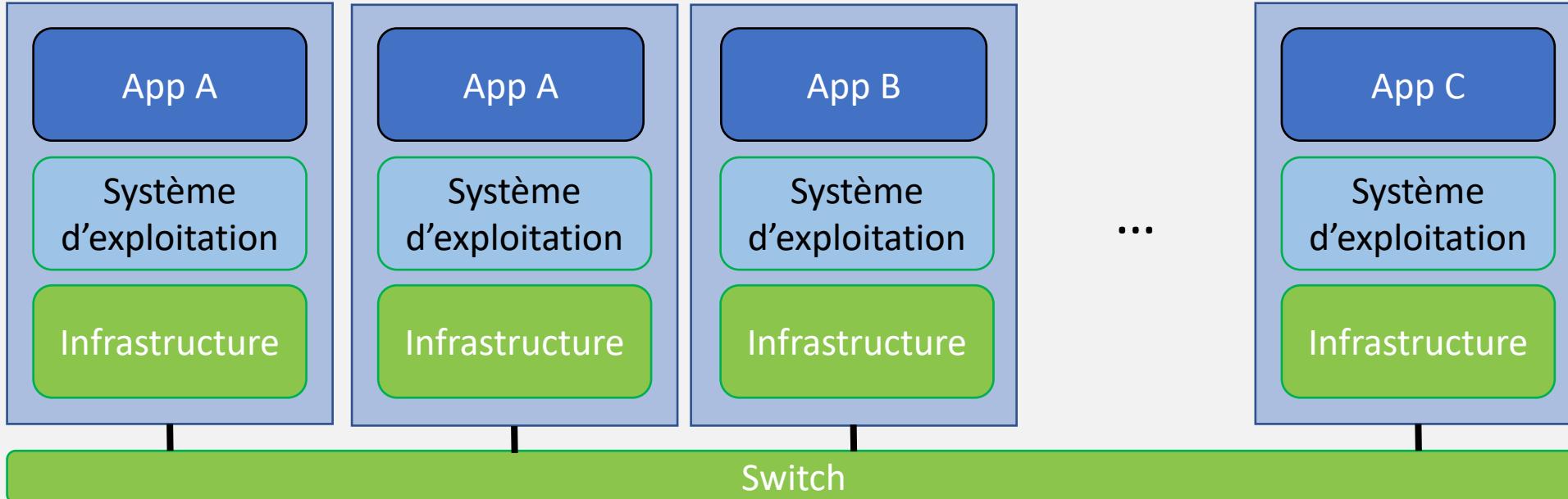
Modèle d'application utilisée en exemple (4-tiers)



Modèles de déploiement applicatif

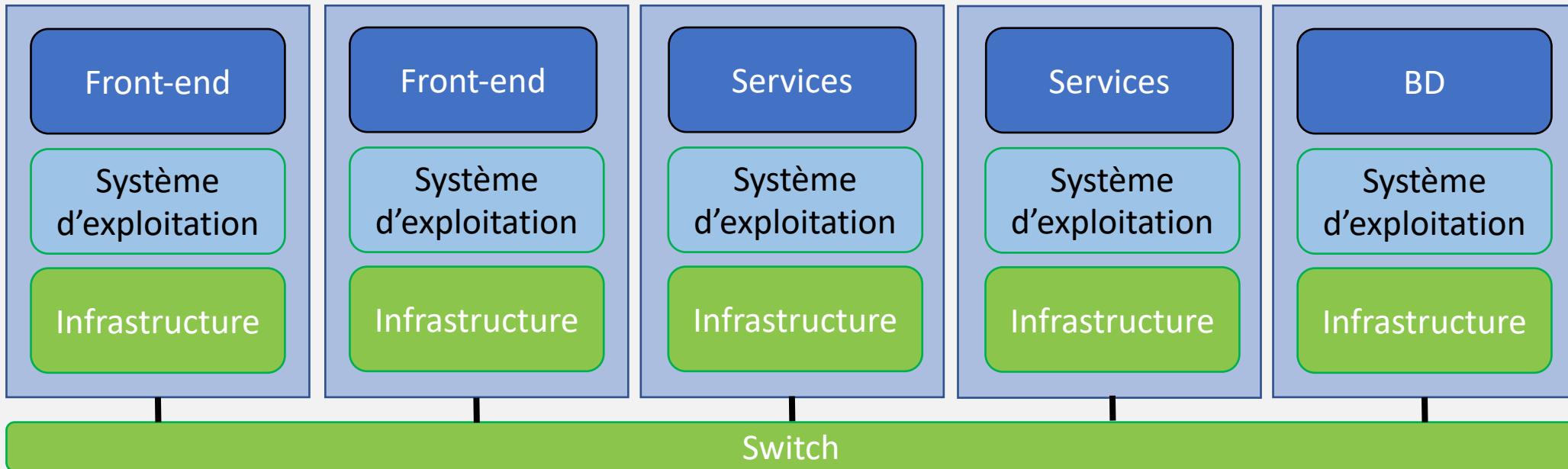
- Modèle 1 : Physique
 - Les applications s'exécutent sur une machine physique
- Modèle 2 : Virtualisation
 - Les applications s'exécutent sur des machines virtuelles
- Modèle 3: Conteneurs
 - Les applications s'exécutent sur un moteur de conteneurs

Modèle 1 – Physique



- Une application est installée par machine physique
- Les applications sont isolées
- Coût élevé / perte de ressource
- Complexité de gestion des dépendances
- Long à déployer

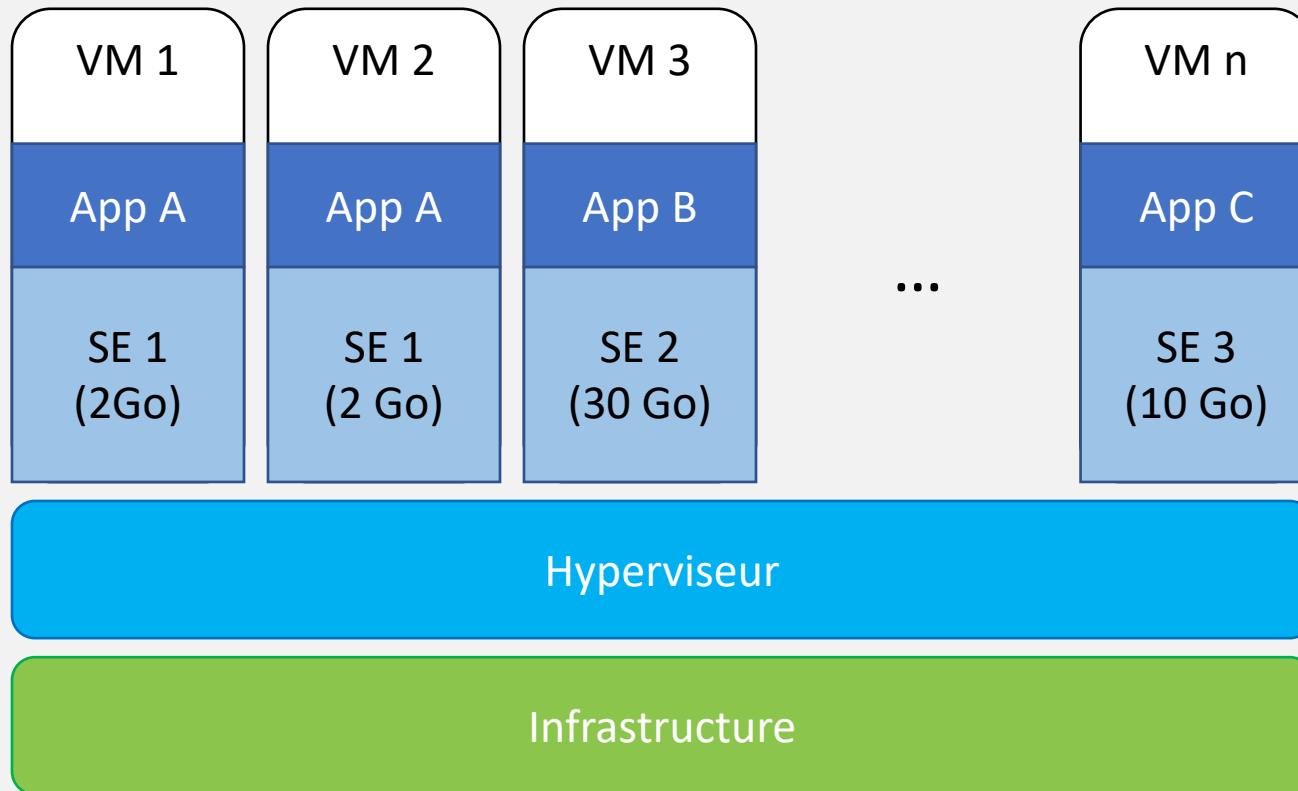
Modèle 1 – Physique



Automatisation :

- Scripts (Puppet, chef, ansible)
- Packages
- Etc.

Modèle 2 – Virtualisation des machines physiques



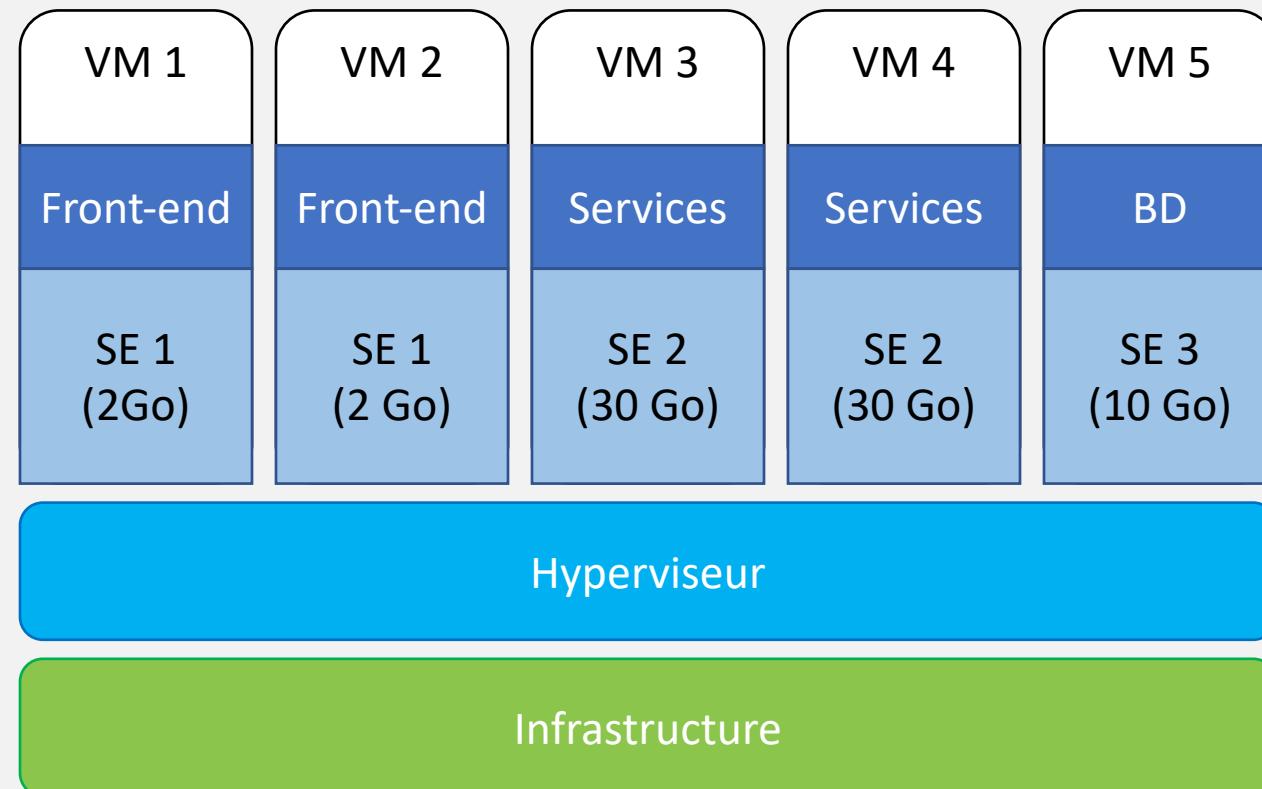
1 hyperviseur par hôte :

- n machines virtuelles (VM) avec des systèmes d'exploitation différents
- Par VM :
 - Un système d'exploitation complet (x Go)
 - Une instance de l'application / service

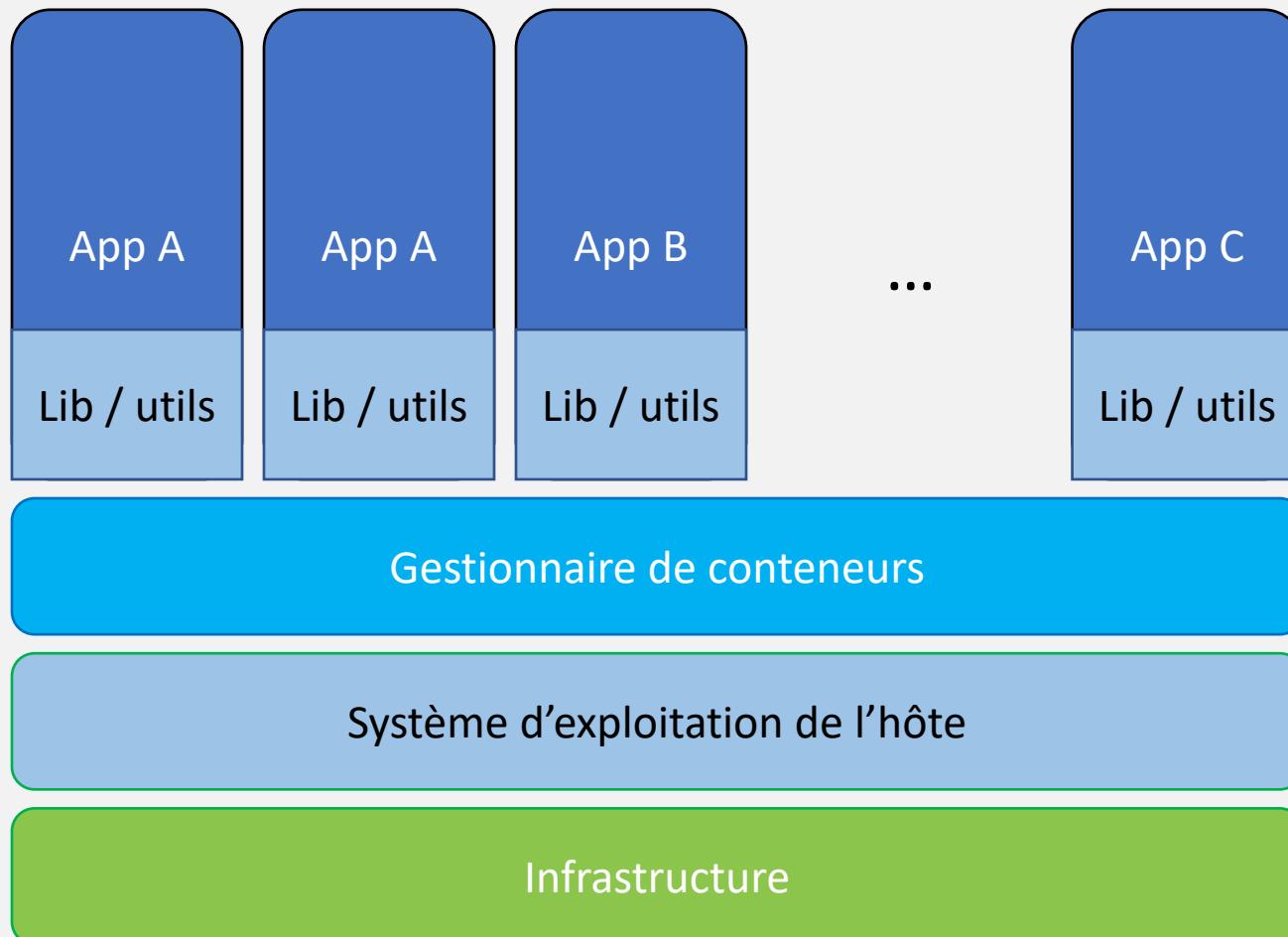
Idées :

- Déplacer les VMs d'un hôte à un autre
 - Répartir la charge
 - Reprise en cas de défaillances matérielles
- Isoler les applications
 - Si dépassement de ressource : peu d'impacts sur les autres applications
 - Si piratage : reste consigné à la VM
- Mise à l'échelle verticale et horizontale facile

Modèle 2 – Virtualisation des machines physiques



Modèle 3 – Virtualisation des applications / services



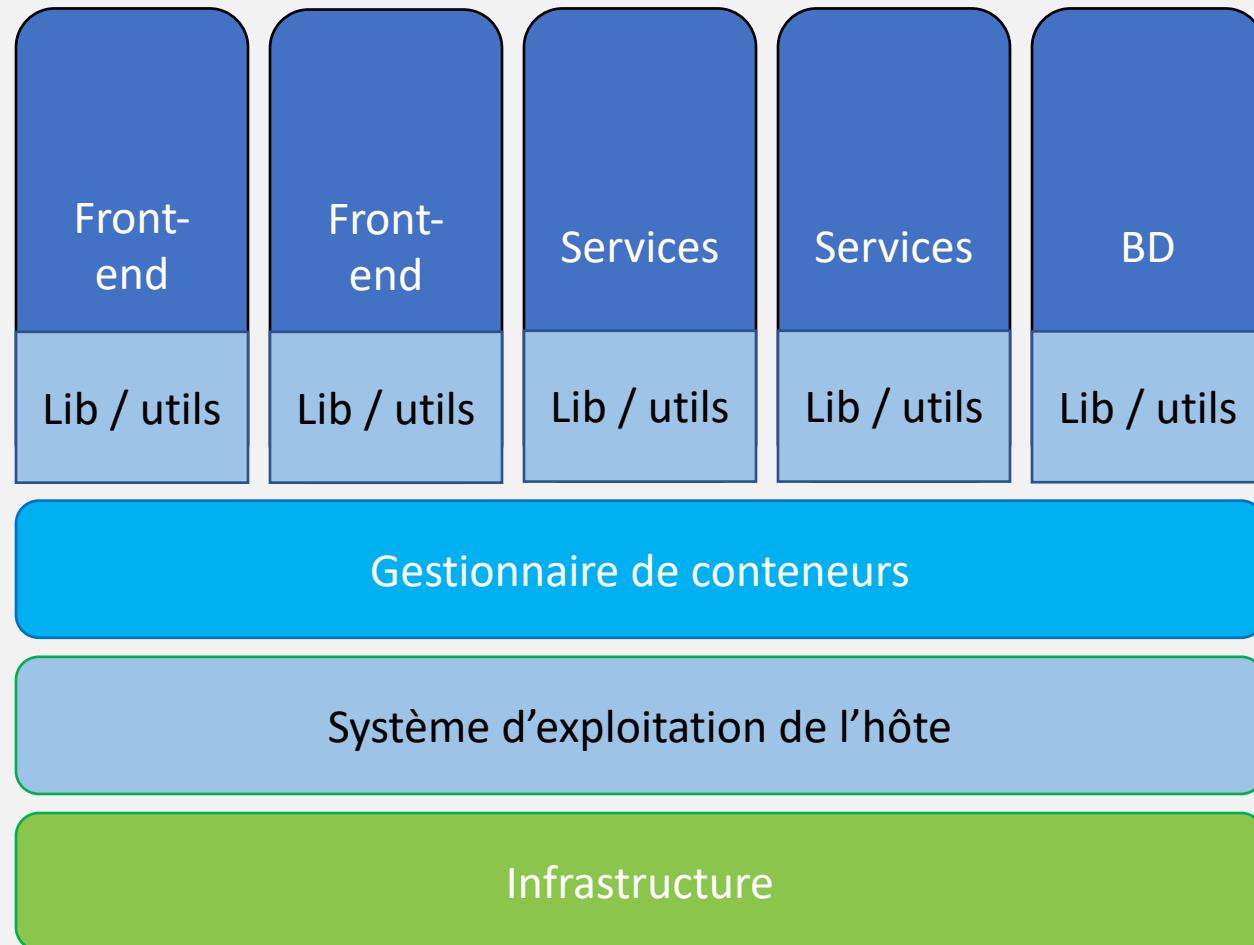
1 hôte :

- 1 système d'exploitation
- Plusieurs conteneurs / hôte
- 1 application / conteneur (= instance d'une image)

Idées :

- Déplacer les conteneurs d'un hôte à un autre
 - Répartir la charge
 - Reprise en cas de défaillances matérielles
- Isoler les applications
 - Si dépassement de ressource : peu d'impacts sur les autres applications
 - Si piratage : reste consigné au conteneur
- Mise à l'échelle verticale et horizontale facile
- Léger : le conteneur ne contient que les dépendances de l'application et l'application (quelques Mo), le conteneur ne contient pas le système d'exploitation

Modèle 3 – Virtualisation des applications / services



Conteneur

- Une image de conteneur contient une application avec toutes ses dépendances
- Une image de conteneur est donc une façon de distribuer une application
- Un conteneur est une instance d'image
- Un conteneur est un tout (unité) indépendant (applicatif et système d'exploitation)
- Un conteneur démarre rapidement

Docker

- Les conteneurs de docker s'exécute dans le moteur docker :
 - Docker est très utilisé dans l'industrie
 - Ils sont légers
 - Par défaut, les conteneurs sont isolés de l'hôte
 - Ils ne peuvent pas accéder aux ressources de l'hôte
 - Ils ont tous des adresses IP
- Les images sont organisées en couches d'images :
 - La couche de base est nommée « scratch »
 - Les autres couches sont des images qui se basent sur scratch ou sur la couche précédente, un peu comme de l'héritage en POO

Docker – Quelques commandes

- `docker run [-rm] [autres options] <nom_image> [arguments]`
 - Télécharge l'image si elle n'existe pas en local
 - Crée un conteneur à partir de l'image et l'exécute
- `docker ps [-a]` : affiche les conteneurs actifs (-a : ou tous)
- `docker image <cmd>` (gestion des images) :
 - ls : liste les images locales
 - rm <nom_image> : supprime une image local si possible
 - history <nom_image> : affiche l'historique d'une image (commandes)

Dockerfile

- Les images sont créées à partir du fichier « Dockerfile »

```
FROM scratch

COPY premier-programme /
ENTRYPOINT ["premier-programme"]
```

- Le fichier premier-programme a été préalablement créé :

```
> cat ./premier-programme.cpp
#include <iostream>

int main(int argc, char** argv) {
    std::cout << "Bonjour à tous !" << std::endl;
}
PS /Users/pfl/tmp/Dockerdemo> g++ ./premier-programme.cpp -o ./premier-programme -static
```

Dockerfile

- L'image est créée à partir du Dockerfile
- Utilisez la commande `docker build --tag <nom_image>:<version>` .

```
> docker build --tag premier-programme:latest .
Sending build context to Docker daemon 51.71kB
Step 1/3 : FROM scratch
    --->
Step 2/3 : COPY premier-programme /
    ---> ef6a20c69526
Step 3/3 : ENTRYPOINT ["/premier-programme"]
    ---> Running in 137f676d2d6c
Removing intermediate container 137f676d2d6c
    ---> e0e83c6eaf36
Successfully built e0e83c6eaf36
Successfully tagged premier-programme:latest
```

- Test :

```
> docker run --rm premier-programme
Bonjour à tous !
```

Dockerfile

- Les images sont créées à partir du fichier « Dockerfile »

```
> docker image history premier-programme
IMAGE          CREATED          CREATED          SIZE          COMMENT
BY             14 minutes ago   /bin/sh -c #(nop)  ENTRYPPOINT ["/premier-pro...  0B
e0e83c6eaf36    14 minutes ago   /bin/sh -c #(nop) COPY file:d502050d334ff3dd...  23kB
ef6a20c69526
```

Dockerfile

- On peut aussi partir d'une image existante :

```
FROM busybox

ENTRYPOINT ["echo"]
CMD ["Bonjour à tous"]
```

```
> docker build --tag echo:latest .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM busybox
--> edabd795951a
Step 2/3 : ENTRYPOINT ["echo"]
--> Running in e8da44107e81
Removing intermediate container e8da44107e81
--> 4b9b29f503bd
Step 3/3 : CMD ["Bonjour à tous"]
--> Running in 04545cc53647
Removing intermediate container 04545cc53647
--> 5c368523eb4a
Successfully built 5c368523eb4a
Successfully tagged echo:latest
```

docker image history echo				
IMAGE	CREATED	CREATED BY	SIZE	COMMENT
5c368523eb4a	About a minute ago	/bin/sh -c #(nop) CMD ["Bonjour à tous"]	0B	
4b9b29f503bd	About a minute ago	/bin/sh -c #(nop) ENTRYPOINT ["echo"]	0B	
edabd795951a	2 days ago	/bin/sh -c #(nop) CMD ["sh"]	0B	
<missing>	2 days ago	/bin/sh -c #(nop) ADD file:4e5169fa630e0afed..	1.22MB	

Dockerfile

```
FROM busybox

ENTRYPOINT ["echo"]
CMD ["Bonjour à tous"]
```

```
> docker run --rm echo
Bonjour à tous
> docker run --rm echo "Ici un nouvel argument !"
Ici un nouvel argument !
```