Les containers

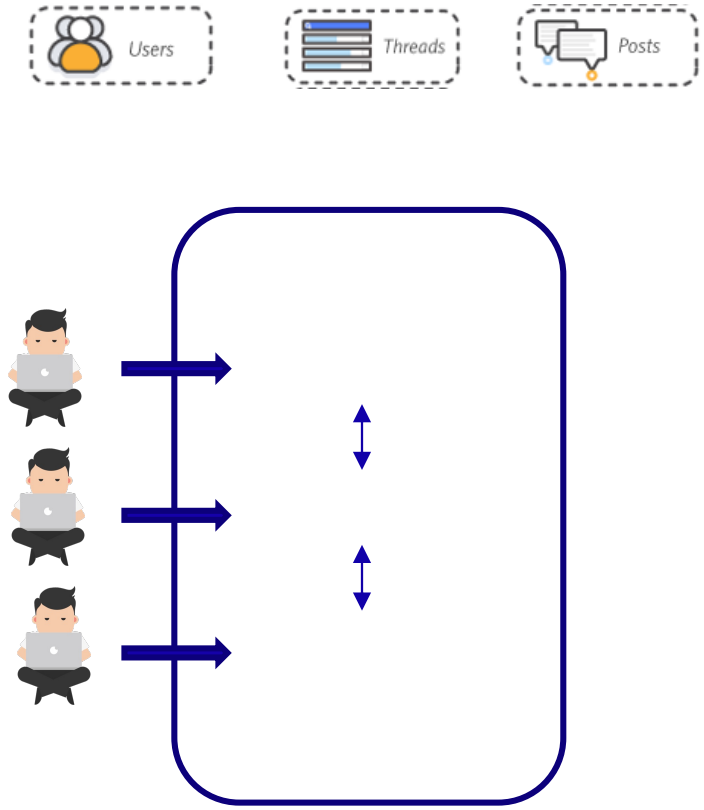
Osman AIDEL

Les Containers

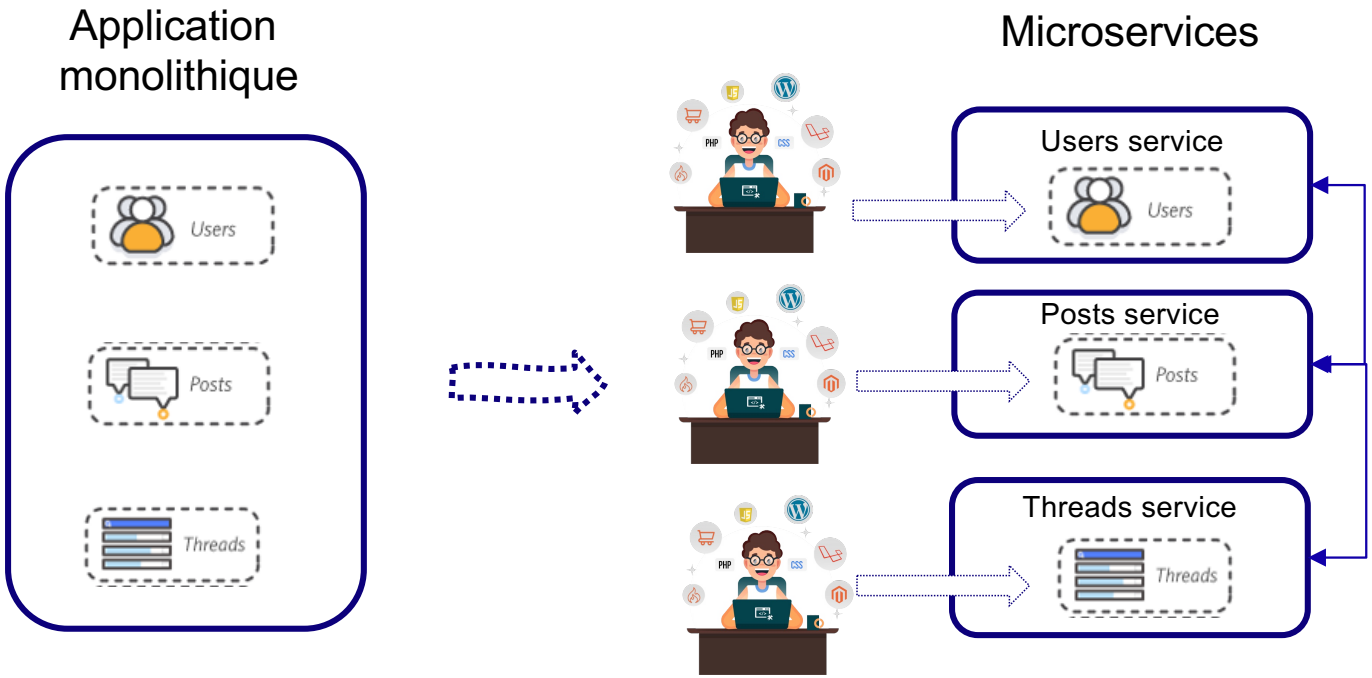
1. Pourquoi les containers ?
2. Docker !!!
3. Comment ça marche
4. Les premières commandes
5. Docker compose
6. L’environnement du module

Pourquoi les containers ?

- Dans les années 2000, les architectures logicielles sont essentiellement monolithiques.
- Les développements et le déploiement sont centralisés sur une seule machine.
- L'application est conçue/codée sur la base d'une seule stack.
- Les équipes de développement doivent faire attention à ne pas affecter le travail des autres équipes.
- Les applications deviennent elles aussi de plus en plus complexes. Il faut assurer la compatibilité des versions entre les différents modules / librairies.
- L'extensibilité verticale n'est pas rentable.
  - Besoin de répondre à des surcharges pour des périodes précises : Black Friday, Noël …
- Une version de production nécessite pour plusieurs modules.



Pourquoi les containers ?

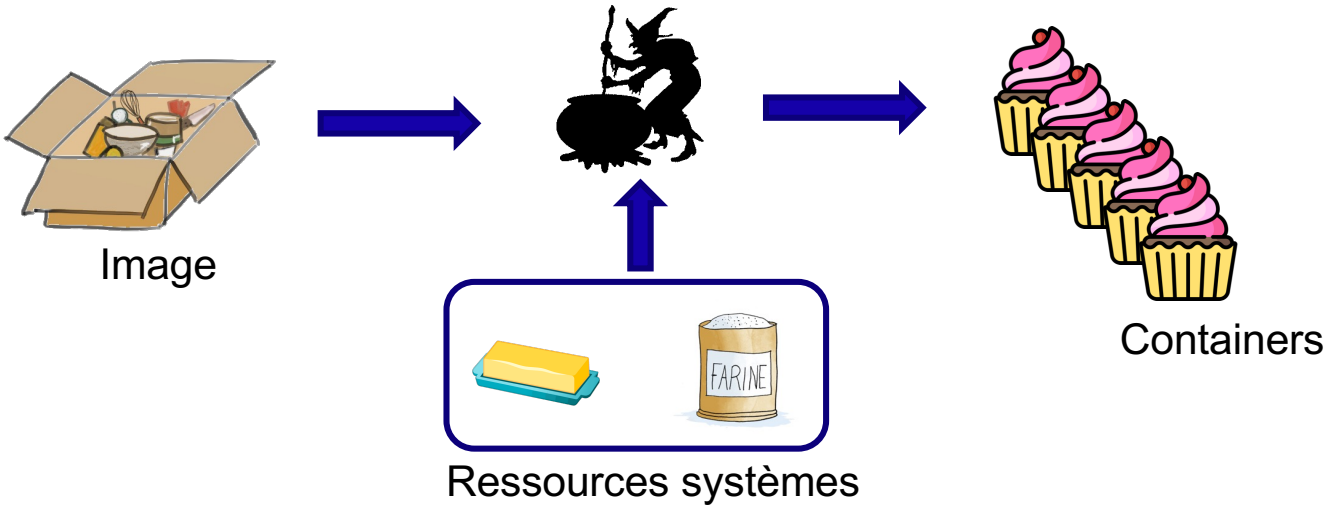


Pourquoi les containers ?

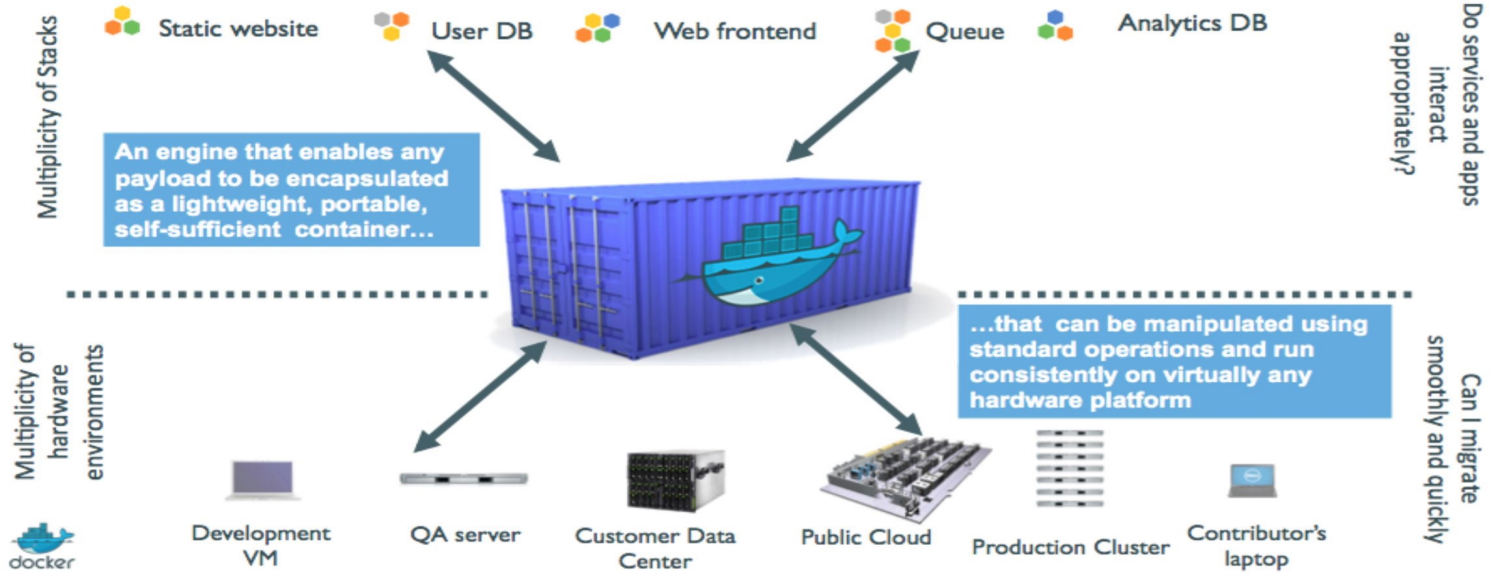
- Fini le temps du « mais ça marche sur ma machine »

Pourquoi les containers ?

- Le principe consiste à regrouper votre application et toutes ses dépendances dans une image.
  - Garantie le fonctionnement de votre application quelque soit l'environnement ou l'application est executée (Windows, Linux …).
- Pour démarrer l'application contenu dans l'image , docker alloue des ressources systèmes (mémoire et RAM) pour exécuter le code applicatif.
- Le container virtualise l'environnement d'execution et s'appuie sur l'OS de la machine hôte
- On appelle container l'instanciation d'une image.



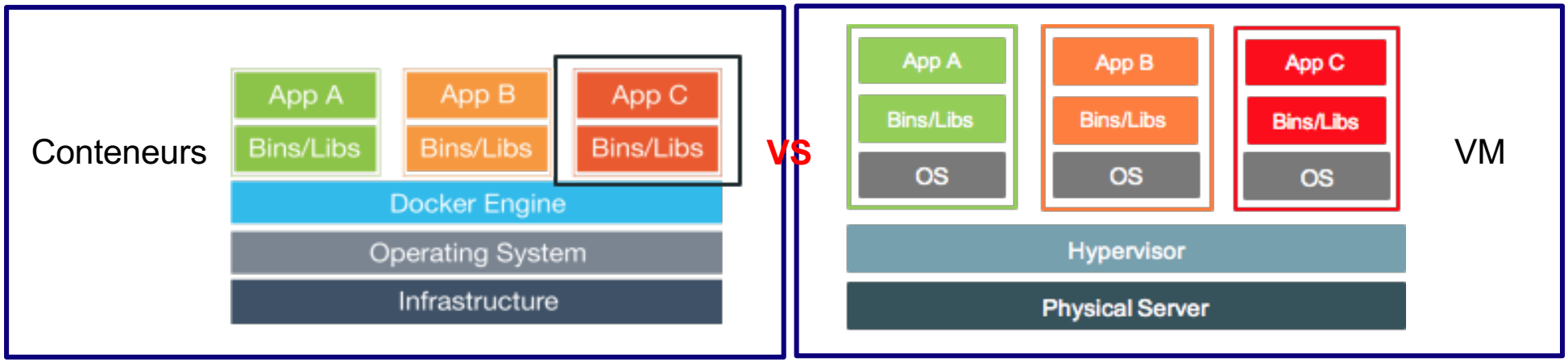
Pourquoi les containers ?



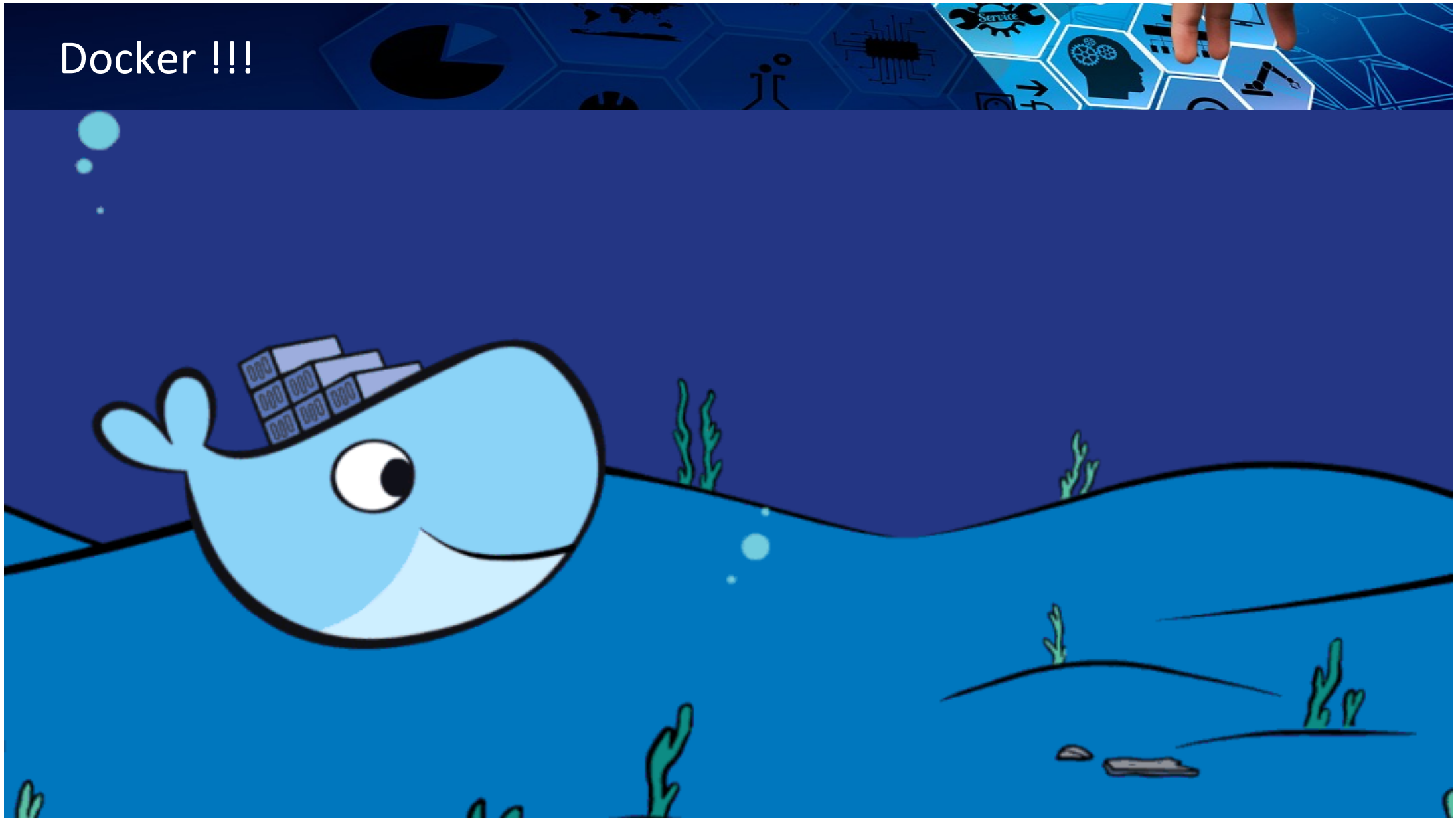
- Encapsuler tous types d'application en embarquant toutes les dépendances – l'application est prête à être livrée au client.
- Le container peut être manipulé comme un objet standard quelle que soit la plateforme d'exécution.
- Les communications entre conteneurs sont simplifiées (création de liens/réseaux).

Pourquoi les containers ?

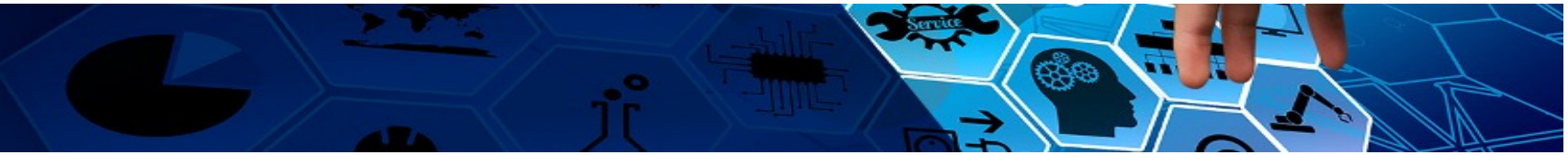
- Une Machine Virtuelle dispose de son propre OS lequel est dupliqué pour chaque Machine Virtuelle.
- Avec les containers, le principe est d'isoler l'exécution de nombreux processus sur un même hôte. Les conteneurs partagent le même noyau et une grande partie des services de l'OS hôte.
- Les ressources sur la machine physique sont moins sollicité en RAM, CPU et disque.
- Dans une architecture à base de conteneurs:
  - le contenu du conteneur , c'est-à-dire le code et ses dépendances (jusqu'au niveau de l'OS), est de la responsabilité du développeur;
  - la gestion du conteneur et les dépendances que celui-ci va entretenir avec l'infrastructure (soit le stockage, le réseau et la puissance de calcul) sont de la responsabilité de l'exploitant.
- Les containers s'appuient la notion de **cgroups et namespaces**.
  - Cgroups définit les ressources systèmes à fournir à un processus.
  - Namespaces c'est une fonctionnalité du noyau Linux, qui permettent de faire en sorte que les processus ne voient pas les ressources utilisées par les autres.



Docker !!!

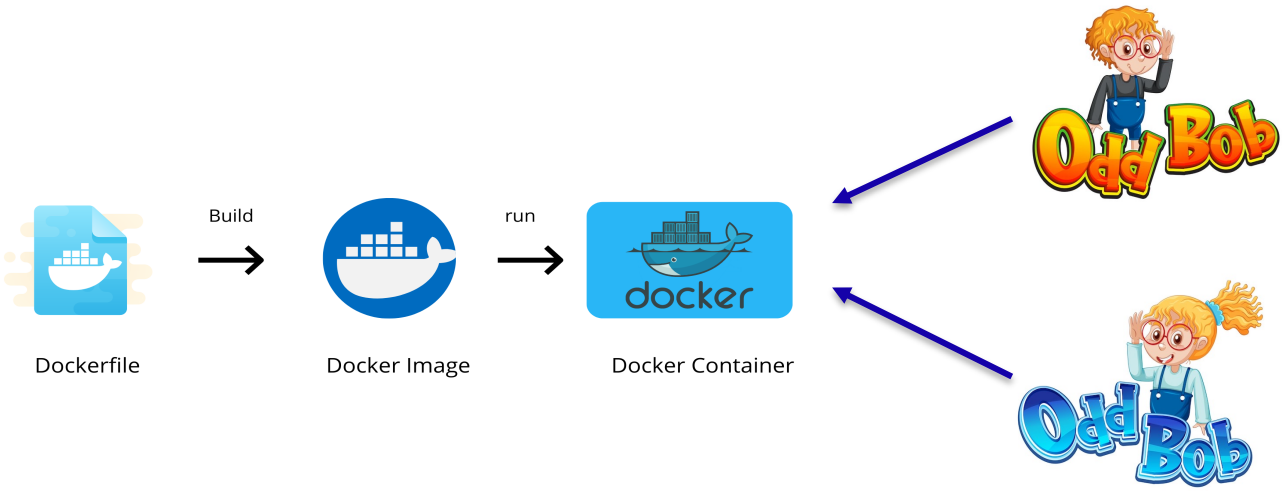


Docker !!!



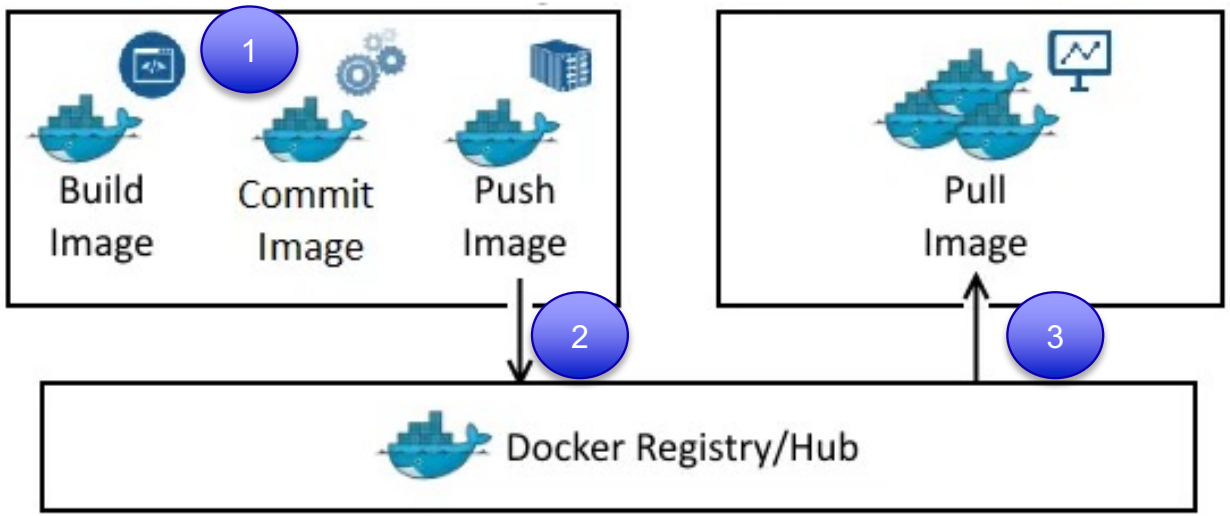
- La première version de Docker date du 20 mars 2013.
- Docker est un système de cloisonnement, il repose sur les « namespaces » et les « cgroups » (control groups) de Linux, qui permettent d'isoler et limiter l'utilisation des ressources de type processeur, mémoire, disque, réseau, etc.
- Outil pour simplifier la création de conteneurs linux en exploitant des technologies noyaux existantes.
- Écrit en GO (langage programmation système crée par Google). Il est sous licence Apache 2.0.
- Environnement d'exécution (CPU, RAM, NET, DISK) tout passe par le « docker-engine » : Ce n'est pas de la virtualisation proprement dite mais de l'isolation de ressources en exploitant le noyau « kernel » de l'hôte.

Comment ça marche ?



- Le dockerfile : le fichier contenant l'ensemble des logiciels à installer : un service par container
- Une image : L'application est « packagée » avec l'ensemble des librairies nécessaires à son fonctionnement.
- Un container : le container est l'exécution d'une image et donc de l'application qu'elle contient. Le container est l'instanciation de l'image. On peut exécuter autant de container que l'on souhaite à partir d'une même image.

Comment ça marche ?



- 1 – Construction de l'image
- 2 – Publication de l'image dans un catalogue publique ou privée.
  - Docker offre un catalogue Docker Hub.
- 3 – Les images sont accessibles aux utilisateurs et peuvent être téléchargées puis instanciées sur une machine (physique ou virtuelle).

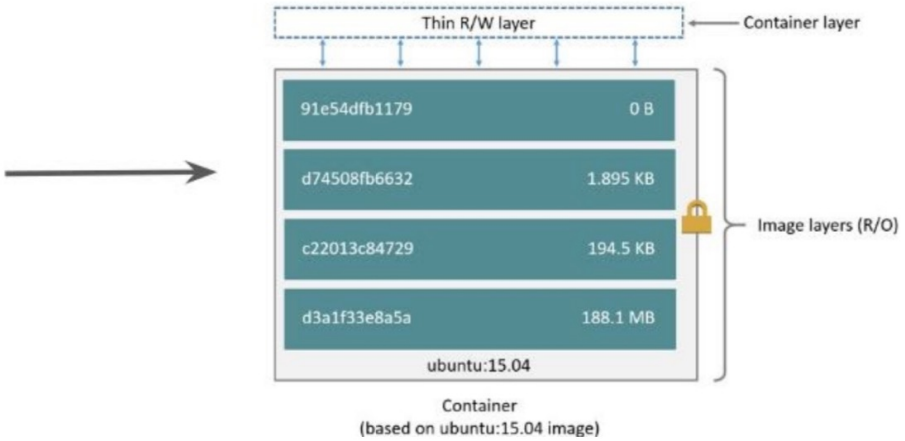
Comment ça marche ?

```
# Construire une image avec un Dockerfile :
docker build -t demo/myimage -f dockerfile

# Instancier un conteneur depuis une image :
docker run -d demo/myimage #(mode détaché « daemon »)
docker run -it demo/myimage #(mode interactif avec shell)

# Télécharger une image à partir des dépots docker
docker pull nginx:latest

# Instancier un conteneur depuis une image existante à partir des dépôts docker
# Si l'image n'est pas disponible, le téléchargement est réalisé automatiquement
docker run nginx:latest
```

Comment ça marche ?

- La création d'une image est réalisée à partir d'une image (FROM ubuntu:15.04)
- Une image est un système de fichier immuable. Pour modifier une image il faut créer une nouvelle image.
- Chaque commande exécutée dans le dockerfile génère des fichiers qui seront stockés dans une couche stockage.
- Télécharger une image, c'est au final télécharger un ensemble de couche spécifique à chaque commande du docker file.
- Un conteneur est au final un ensemble de processus tournant dans un système de fichier en lecture/écriture.
- ATTENTION, contrairement aux images un container peut être modifié dans ce cas, docker associera au container une couche stockage supplémentaire et indépendante de l'image. Par conséquent, si vous réinstanciez un container à partir de l'image vous perdrez vos modifications.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Les premières commandes

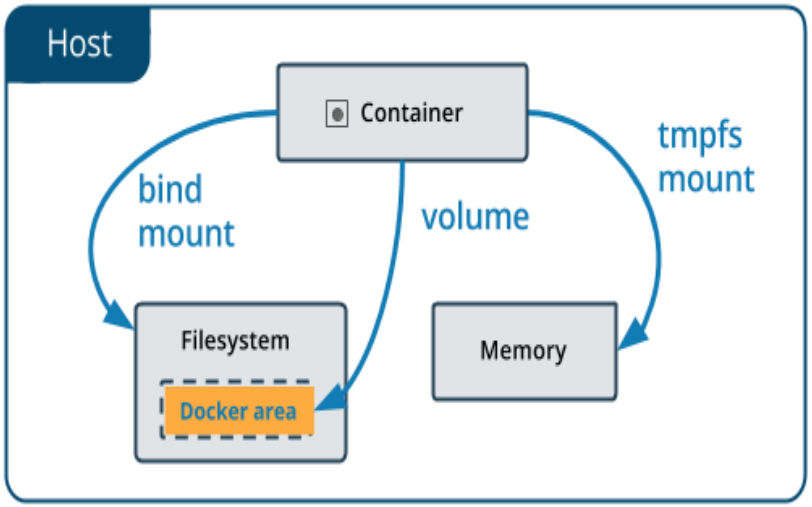
```
# liste les containers en cours d'execution
docker ps

# liste les containers arrêtés
docker ps -a

# L'aide pour utiliser docker
docker --help

# Instanciation d'un container à partir d'une image
docker run -d --name mynginx nginx:latest
* run instancie un container
* -d execute le container en tâche de fond (detach)
* --name : nom associé au container produit
* nginx nom de l'image à instancier
* latest : version de l'image à instancier
```

Les premières commandes

```
#Arrêter le container mynginx
docker stop mynginx

#Une fois qu'un container est instancié il peut être arrêté et redémarré autant de fois que l'on souhaite.
docker start mynginx

docker exec –it mynginx /bin/bash
* exec : execute une commande dans un container
* -ti : demande un terminal interactif
* mynginx : nom du container où exécuter la commande
* /bin/bash : la commande à executer.

# Supprimer le container nginx, celui-ci ne pourra donc plus être démarré ou arrêté. Il faudra instancier un nouveau container à partir de l'image.
docker rm mynginx
```

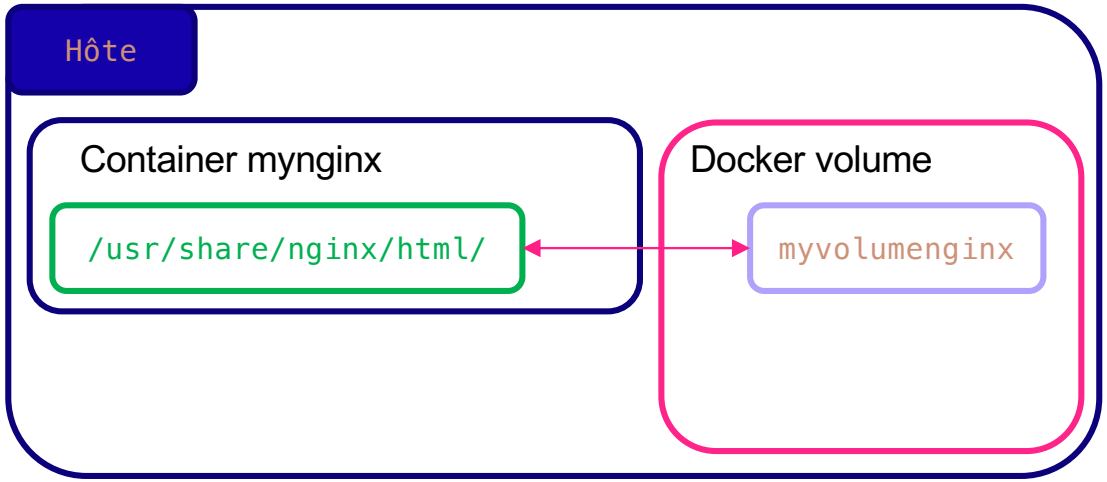
Les volumes

- Les containers sont éphémères et ne sont pas conçus pour préserver des données.
  - La suppression du container supprime toutes les données modifier dans le container.
  - Instancier un container mysql
- Pour conserver des données il est possible d'attacher des volumes persistant de la machine hôte vers les containers.
- Il est possible d'attacher un volume de 3 manières :
  - Bind : on associe un répertoire de la machine hôte à un container. Le contenu de la machine hôté écrase le contenu du répertoire associé au container.
  - Volume : on crée un volume dédié à docker sur la machine hôte. Le point de montage du volume conserve les données du container.
  - Tmpfs : volume temporaire d'espace de travail. Son contenu est supprimé à chaque redémarrage du container.



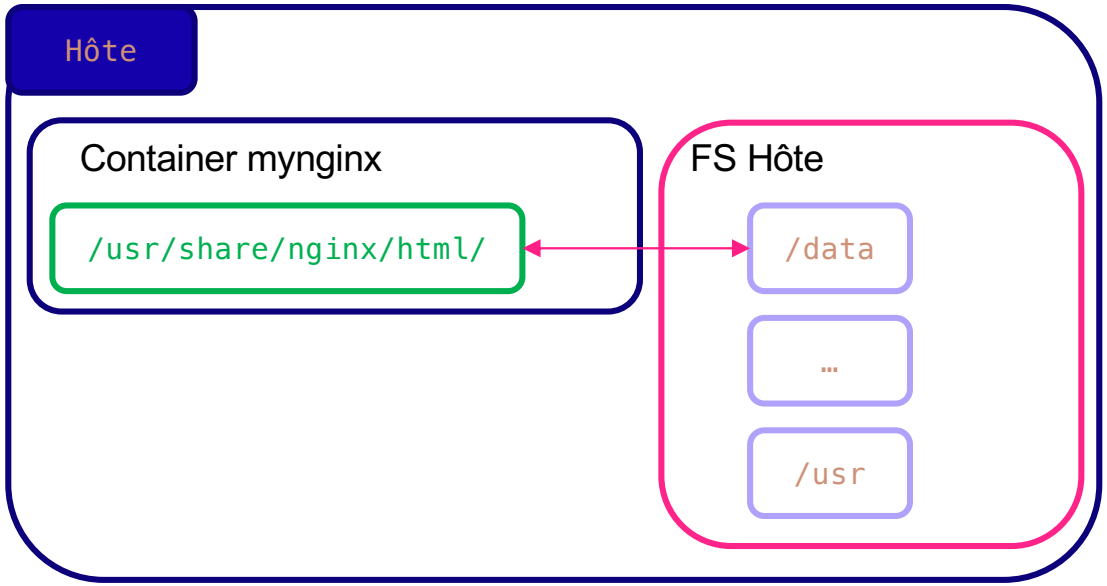
Les volumes : type volume

```
#Création d'un volume
docker volume create myvolumenginx

#liste les volumes disponibles
docker volume ls

#Lancement du conteneur avec un volume
#Le volume myvolumenginx est monté sur le chemin /usr/share/nginx/html/ du container.
#Les fichiers contenus initialement dans /usr/share/nginx/html/ sont conservés et toutes
modifications seront conservées même après suppression du container.
docker run -d -v myvolumenginx:/usr/share/nginx/html/ --name mynginx nginx:latest

#Suppression d'un volume
docker volume rm myvolumenginx
```



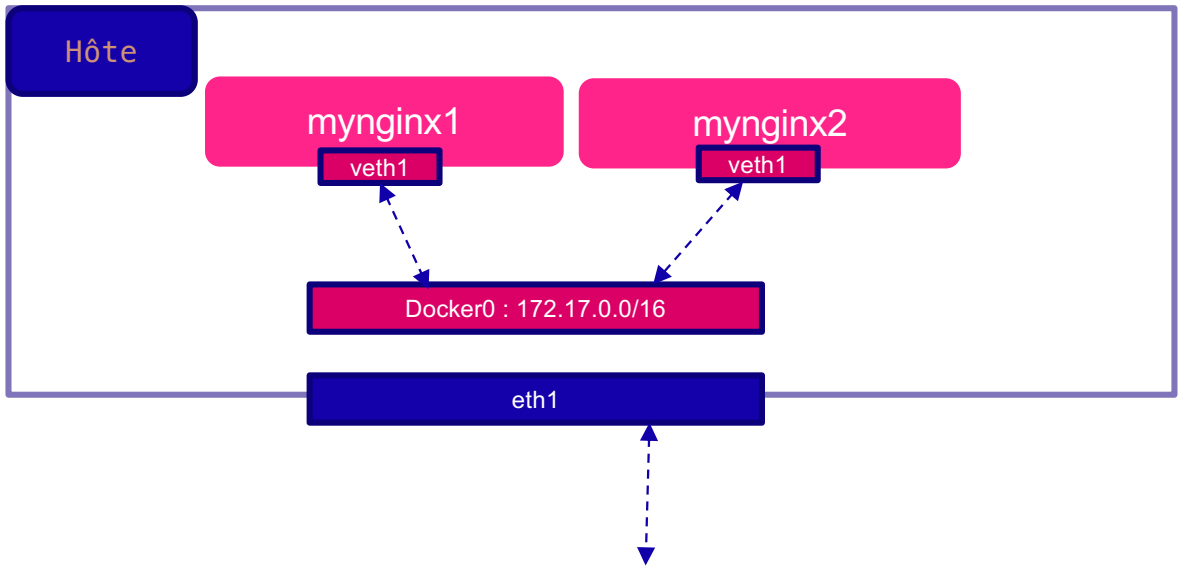
Les volumes : type bind

```
#Exemple avec un montage Bind
#Le répertoire /usr/share/nginx/html/ du container sera remplacé par le
répertoire /data de la machine hôte.

docker run -d --hostname --mount
type=bind,src=/data/,destination=/usr/share/nginx/html/ --name mynginx
nginx:latest
```



Le réseau

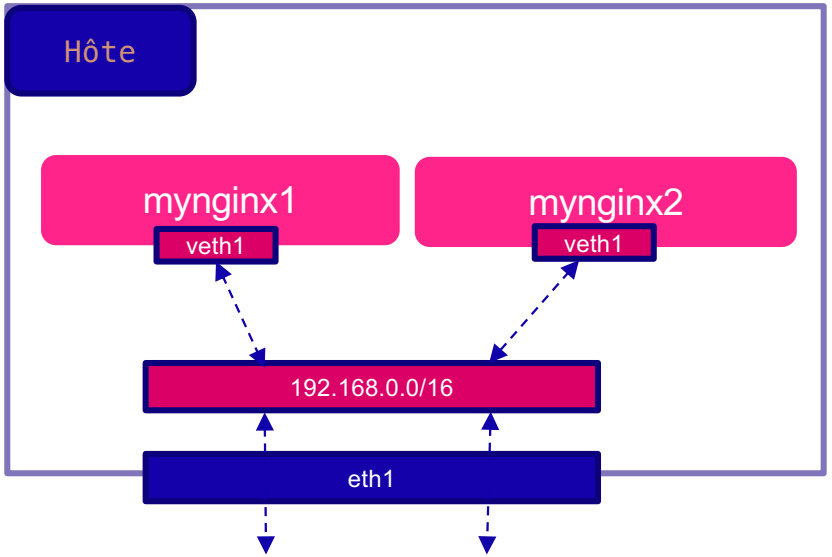
- Docker dispose de son propre réseau : docker0 mais pas de résolution de nom
- Par défaut, docker est configuré pour cloisonner les containers au niveau réseau.
- Chaque container se voit associer une nouvelle IP à chaque instanciation et redémarrage.



Le réseau

- Pour bénéficier d'un DNS il faut créer son propre réseau.
- Chaque container se voit associer une nouvelle IP à chaque instanciation et redémarrage.
- L'utilisation est très pratique et Il est recommandé de toujours faire référence à un container par son nom.

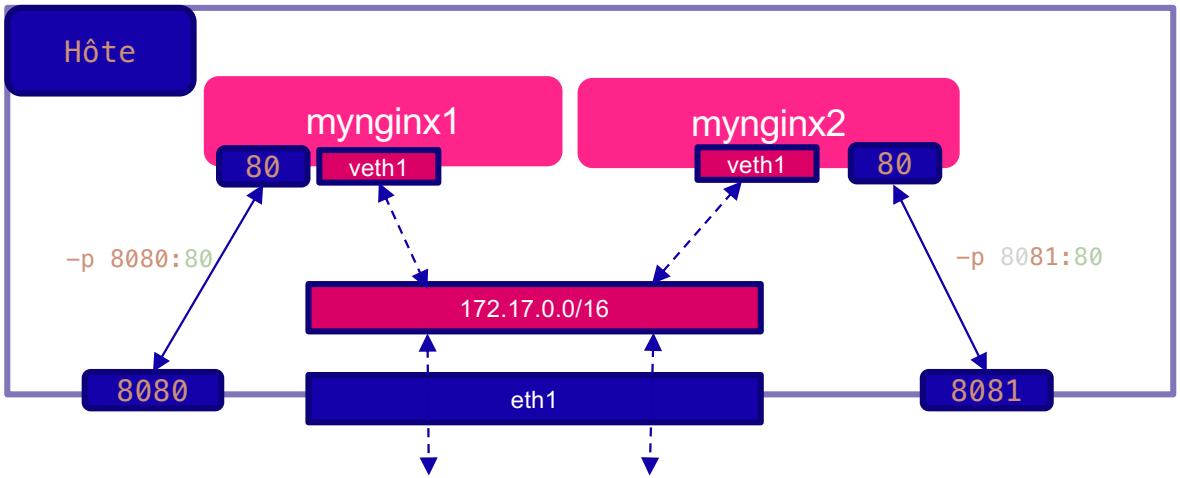
```
# Liste les réseaux disponibles
docker network ls

# Création d'un réseau mynetwork
docker network create --driver=bridge --subnet=192.168.0.0/16 mynetwork

# Liste les informations sur un réseau
docker network inspect mynetwork

# Connecter un container à notre réseau mynetwork
docker network connect mynetwork mynginx1
docker network connect mynetwork mynginx2

# Suppression d'un réseau mynetwork
docker network rm mynetwork
```



Le réseau

- Lorsqu'un container écoute sur un port (expose), Il est possible d'associer le port d'un container au port de la machine hôte. On parle de publication de ports.

```
docker run -d --name mynginx1 -p 8081:80 nginx:latest
docker run -d --name mynginx2 -p 8080:80 nginx:latest
```



Les commandes pratiques

```
# Lister les objets docker disponible sur l'hôte
docker container ls
docker image ls
docker volume ls
docker network ls

# Afficher le stdout d'un container
docker logs mynginx

# Inspecter la configuration d’un objet docker
docker inspect <OBJECT ID>
docker container inspect <container name>
docker volume inspect < volume name>
docker network inspect <network name>


# purge tous les containers, réseau, images et volumes non utilisés
docker system prune -a --volumes
```

Docker Compose

- Docker Compose est un **outil destiné à définir et exécuter des applications Docker à plusieurs conteneurs**.
- Docker Compose s'appuie sur un fichier YAML pour configurer les services de votre application.
- La configuration du stockage, du réseau et des containers est centralisé en un seul fichier.
- Le démarrage de container s'en voit simplifier par l'utilisation d'une seule commande docker-compose.

Docker Compose

docker-compose.yml

```yaml
version: '3'
services:
   site_web:
         image: 'bitnami/apache:latest'
         container_name: 'site_web'
         ports:
                  - '80:8080'
         networks:
                  - ngnet
         volumes:
                  - ./httpd.conf:/opt/bitnami/apache2/conf/httpd.conf
                  - logs:/opt/bitnami/apache2/logs
volumes:
   logs:

networks:
   ngnet:
```

Docker Compose

```
# Création automatique des volumes, réseau et containers avec téléchargement des images.
# Avec l'option -d les containers sont exécutés en tâche de fond
docker-compose up -d

# Arrêt de tous les containers
docker-compose stop

# Démarrage de tous les containers instanciés
docker-compose start

# Suppression des containers arrêtés
docker-compose rm

# Arrêt et suppression des réseaux et des containers
docker-compose down

# Afficher l'output des containers
docker-compose logs
```

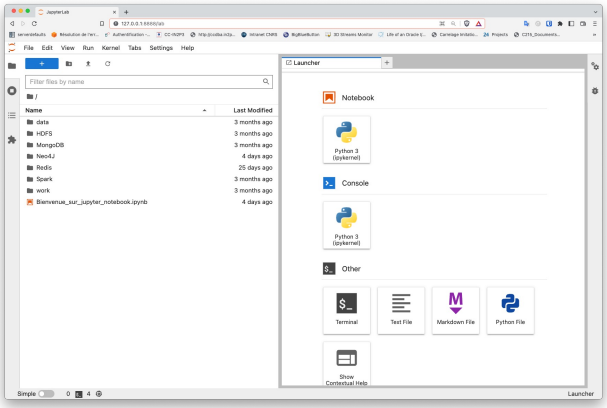
Liste complète des commandes docker-compose :
https://docs.docker.com/engine/reference/commandline/compose_down/

L’environnement du module

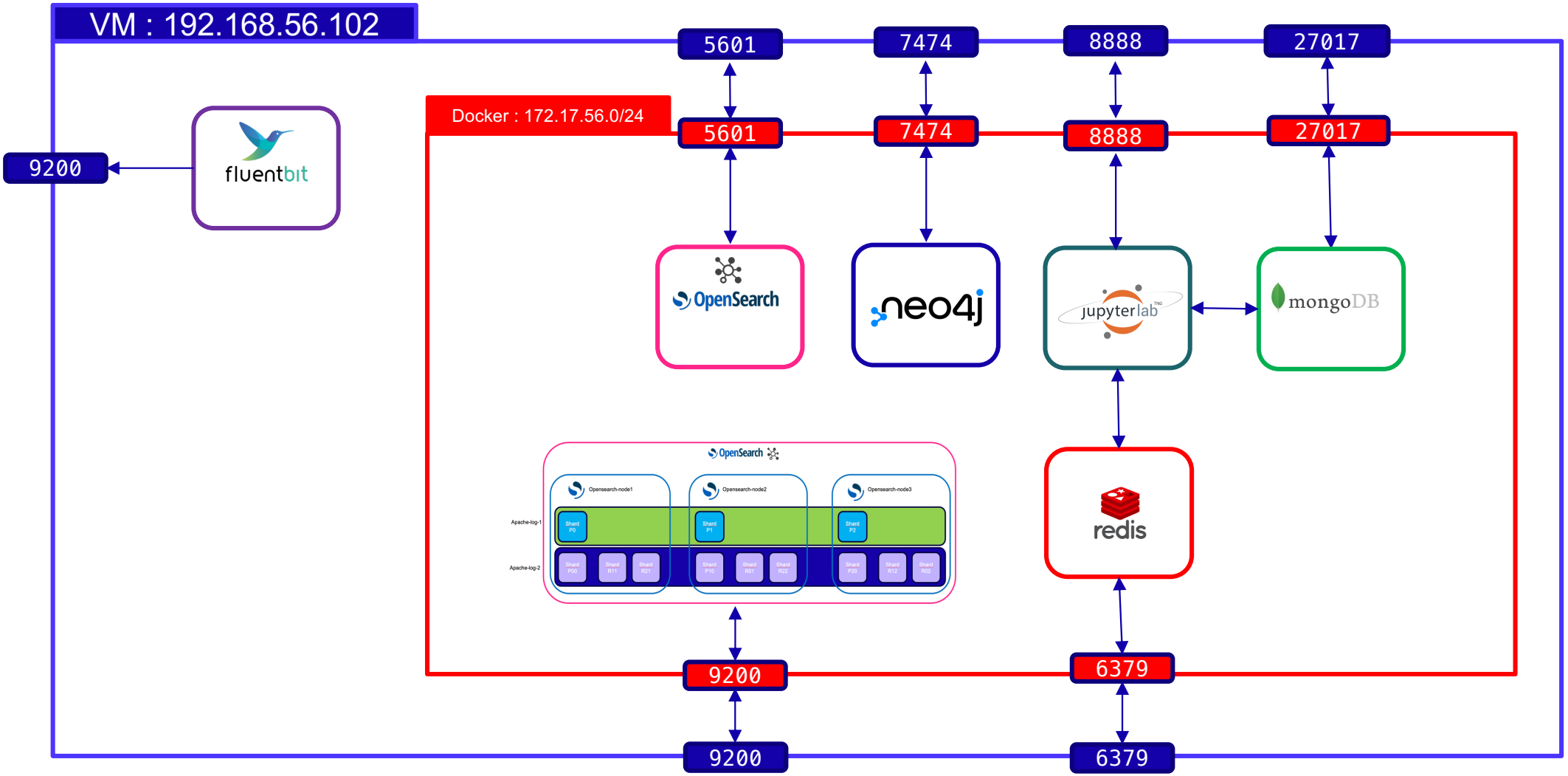
- Les TP sont disponibles à partir du container Jupyter.

```
(base) ccoaidel:BIGDATA oaidel$ docker start jupyter
jupyter
(base) ccoaidel:BIGDATA oaidel$ docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS         PORTS                                              NAMES
2dfaf96f4427   guizmo38/jupyter-training:latest   "tini -g -- start-no…"   3 weeks ago   Up 9 seconds   0.0.0.0:4040->4040/tcp, 0.0.0.0:8888->8888/tcp   jupyter
(base) ccoaidel:BIGDATA oaidel$
```

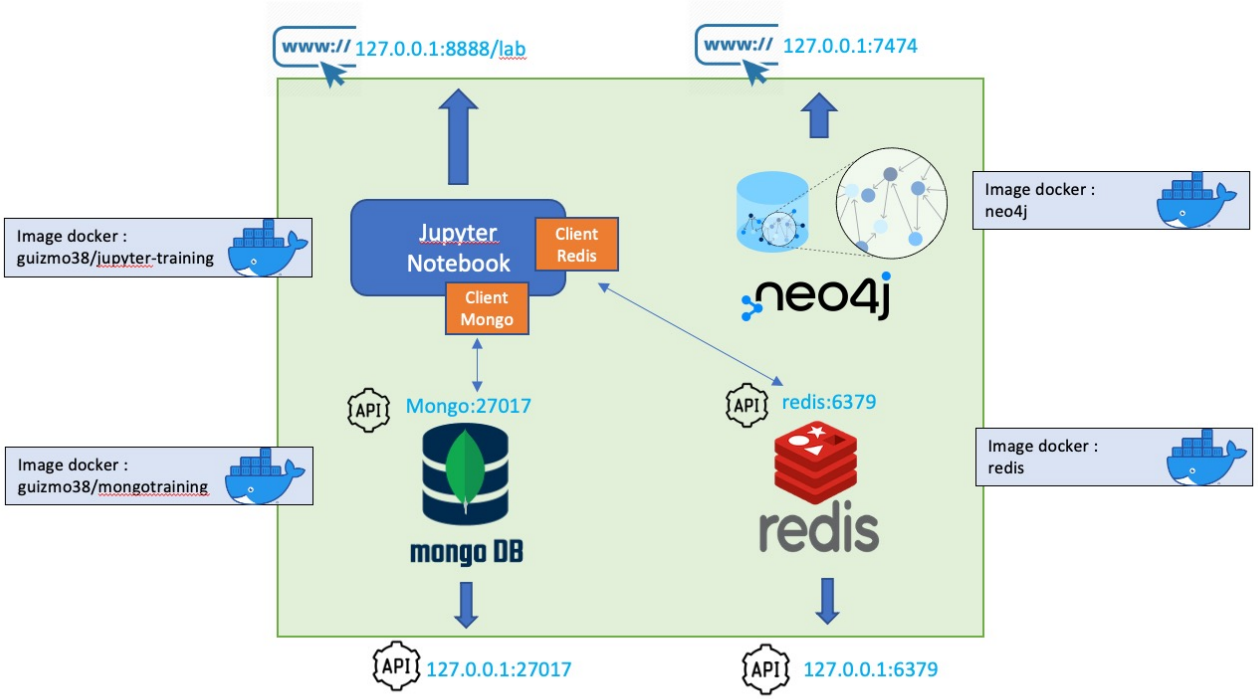
- Go to http://127.0.0.1:8888/lab



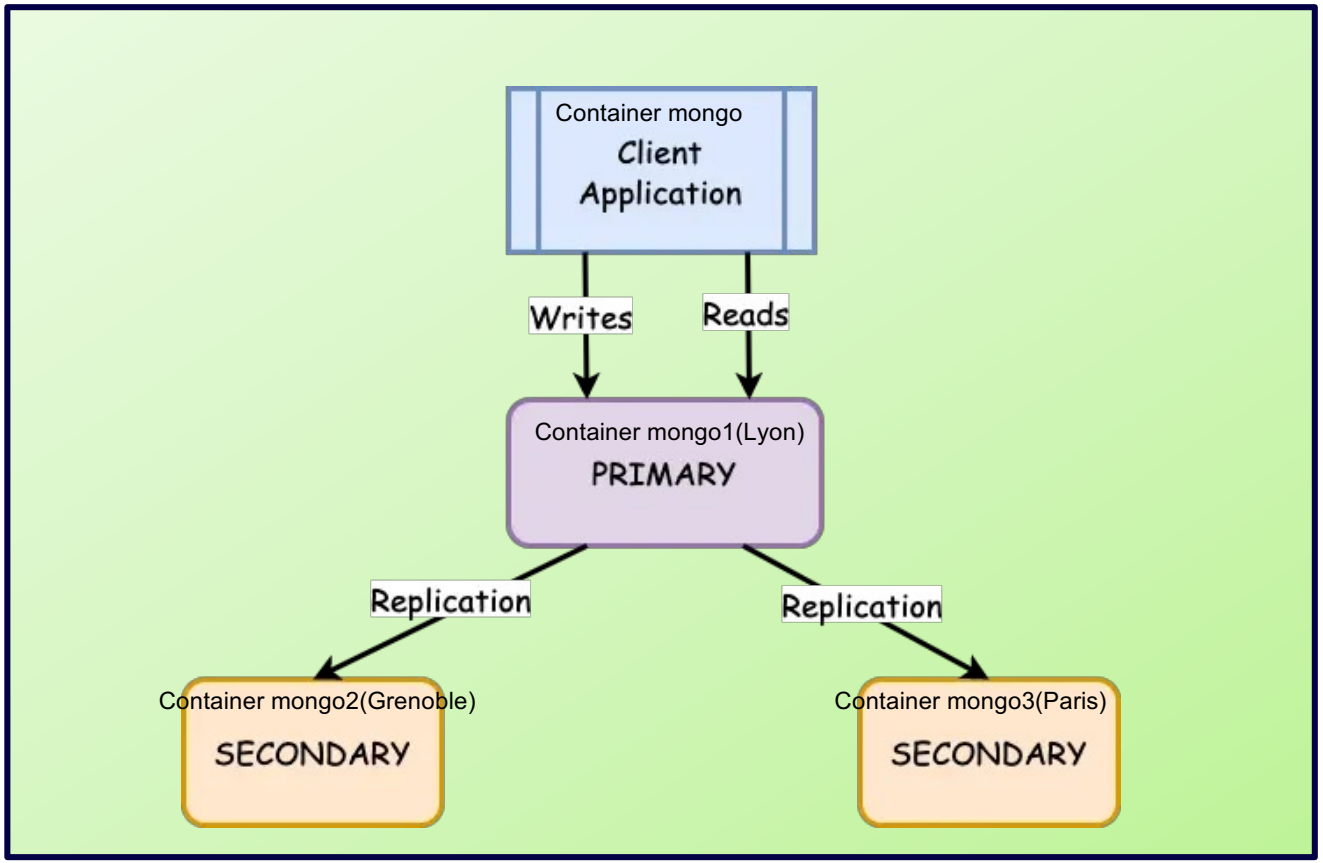
L’environnement du module



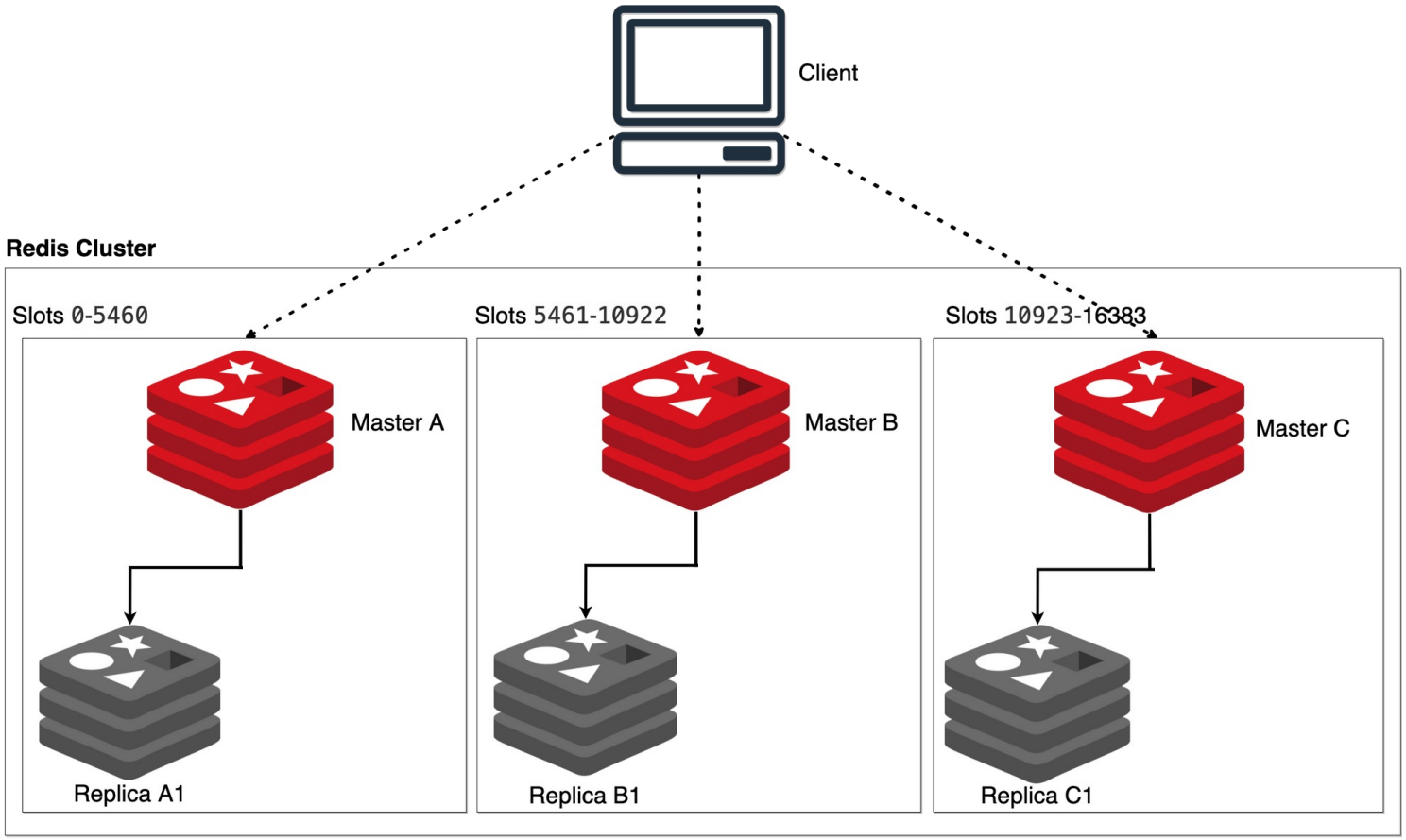
L’environnement du module



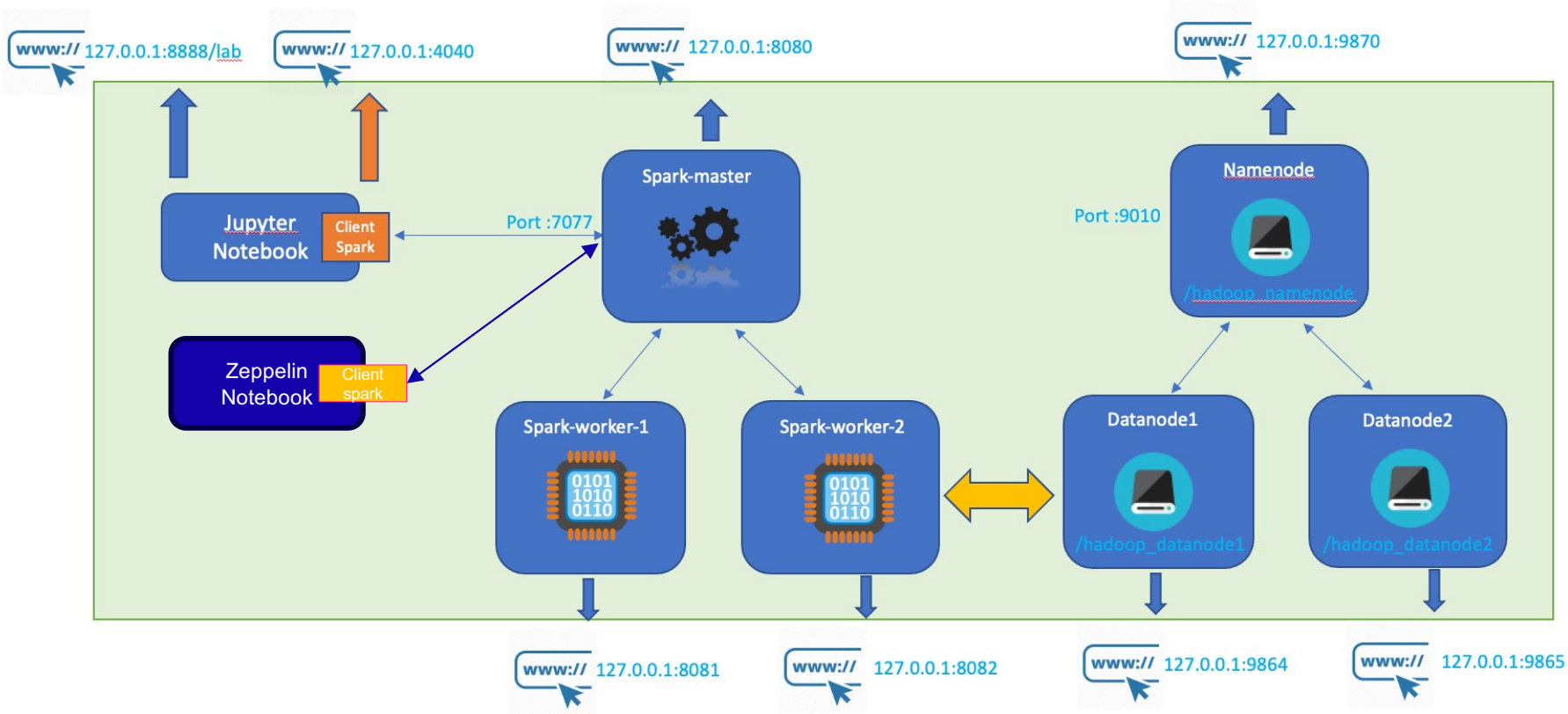
TP ReplicaSet



TP Redis sharding

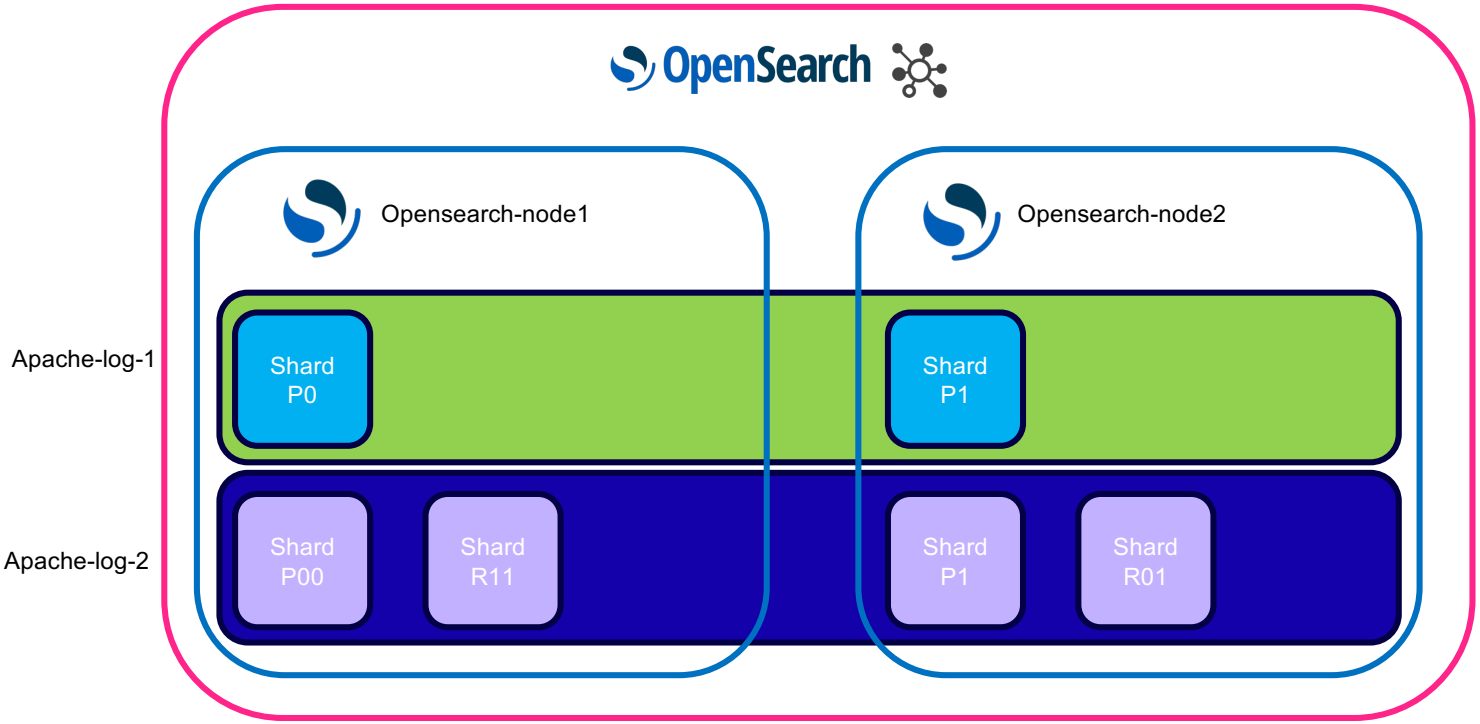


L’environnement du module BIG DATA

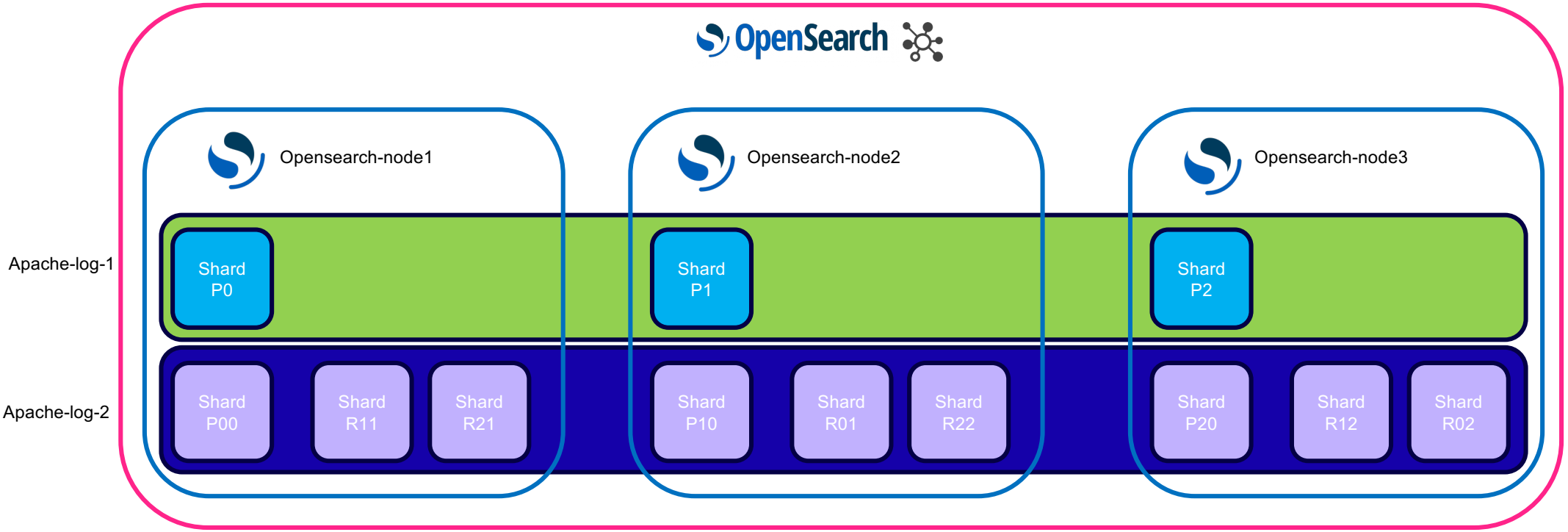


TP Docker

TP Sharding


OpenSearch
Opensearch-node1
Opensearch-node2
Apache-log-1
Apache-log-2
Shard P0
Shard P1
Shard P00
Shard R11
Shard P1
Shard R01

TP Sharding


OpenSearch
Opensearch-node1
Opensearch-node2
Opensearch-node3
Apache-log-1
Apache-log-2
Shard P0
Shard P1
Shard P2
Shard P00
Shard R11
Shard R21
Shard P10
Shard R01
Shard R22
Shard P20
Shard R12
Shard R02

Les commandes pratiques

- Sauvegarde
- Docker save –o destination/monimage.tar grafana:latest
- Tar –C dtar/ -xvf monimage.tar
- Docker load –i destination/monimage.tar

- /etc/systemd/

- docker run -it --entrypoint bash ubuntu