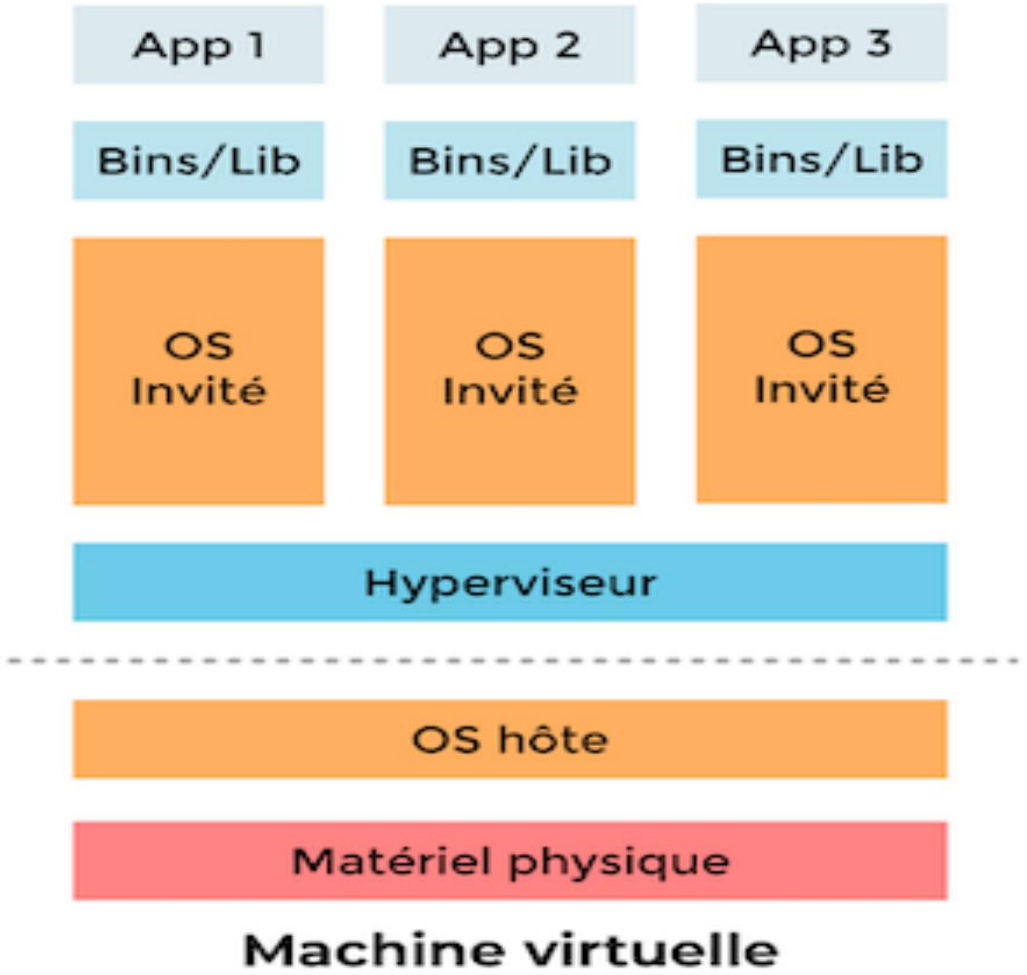
Les conteneurs

Les conteneurs

1) Machine virtuelle:



Machine virtuelle

Les conteneurs

Contraintes des machines virtuelles:

- une machine virtuelle prend du **temps** à démarrer.
- une machine virtuelle **réserve les ressources** (CPU/RAM) sur le système hôte.

Avantages des machines virtuelles:

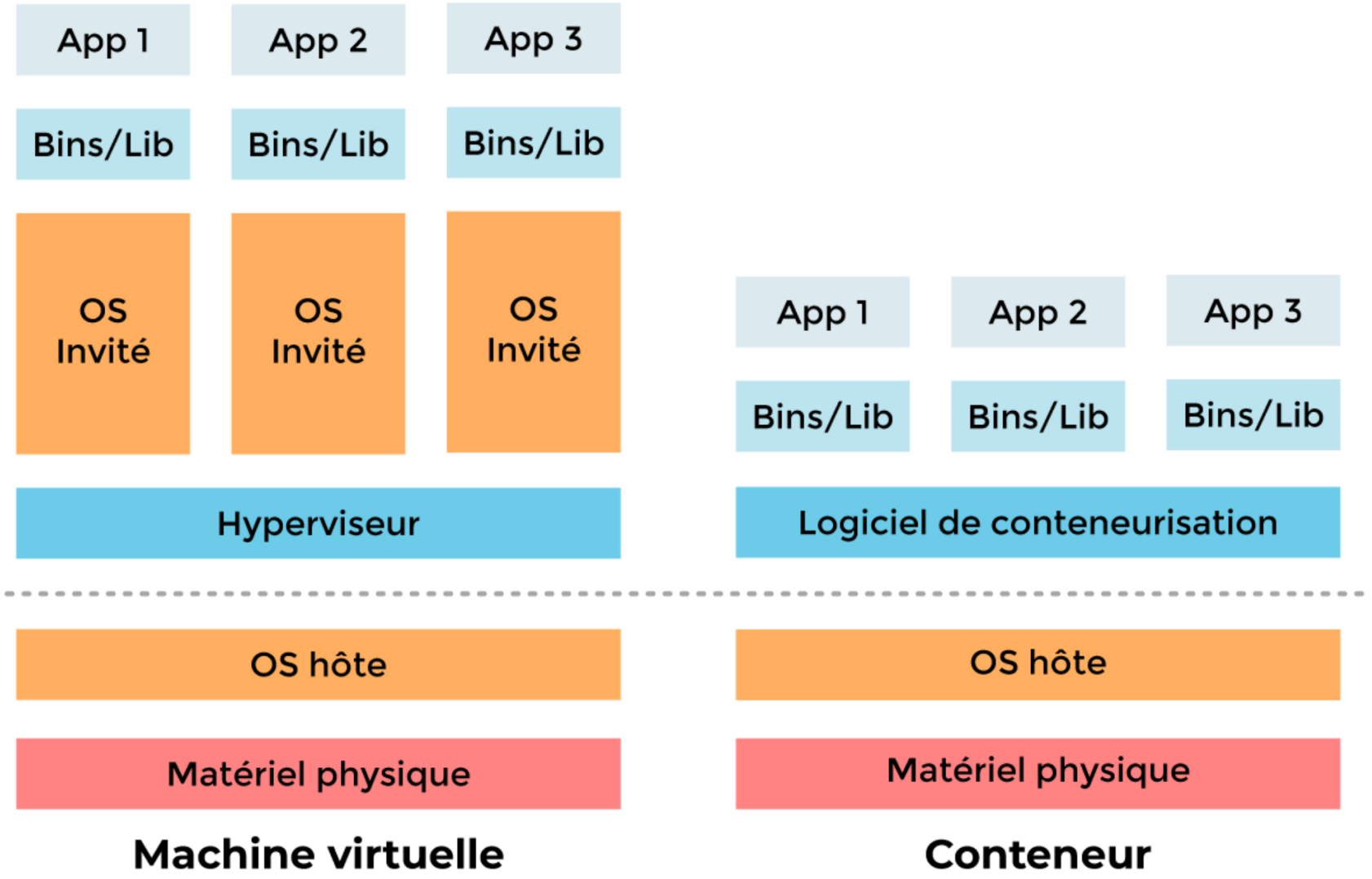
- une machine virtuelle est totalement **isolée** du système hôte ;
- les ressources attribuées à une machine virtuelle lui sont totalement **réservées** ;
- On installer **différents OS** (Linux, Windows, BSD, etc.).

Les conteneurs

2) Le conteneur:

- conteneur Linux est un **processus** ou un ensemble de processus isolés du reste du système, tout en étant **légers**.
- Le conteneur permet de faire de la ***virtualisation légère***, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une **isolation des processus**. Le conteneur partage donc les ressources avec le système hôte.
- Les technologies de conteneur existent depuis de nombreuses années (OpenVZ ou LXC)

Les conteneurs



Les conteneurs

- Une autre différence importante avec les machines virtuelles est qu'un conteneur **ne réserve pas** la quantité de CPU, RAM et disque attribuée auprès du système hôte. Ainsi, nous pouvons allouer 16 Go de RAM à notre conteneur, mais si celui-ci n'utilise que 2 Go, le reste ne sera pas verrouillé.

- Les conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation, ils peuvent **démarrer beaucoup plus rapidement** et plus fréquemment qu'une machine virtuelle sur nos serveurs hôtes, et ainsi réduire encore un peu les frais de l'infrastructure.

- Il y a aussi la possibilité de faire tourner des conteneurs sur le poste des développeurs, et ainsi de réduire les différences entre la production, et l'environnement local sur le poste des développeurs.

- conteneurs étant capables de démarrer très rapidement, ils sont souvent utilisés en production pour ajouter des ressources disponibles et ainsi répondre à des besoins de mise à l'échelle, ou de scalabilité. Mais ils répondent aussi à des besoins de préproduction ; en étant légers et rapides au démarrage, il permettent de créer des environnements dynamiques et ainsi de répondre à des besoins métier.

DOCKER

Premier conteneur:

Docker run hello-world

```
➜ docker run hello-world
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/
```

Docker

Démarrer un serveur Nginx avec un conteneur Docker

```
docker run -d -p 8080:80 nginx
```

-d pour **détacher le conteneur** du processus principal de la console. Il vous permet de continuer à utiliser la console pendant que votre conteneur tourne sur un autre processus.

-p pour définir **l'utilisation de ports**. Dans notre cas, nous lui avons demandé de transférer le trafic **du port 8080 vers le port 80 du conteneur. (**http://127.0.0.1:8080 **)**

Docker

- **docker exec –ti id-conteneur bash**

  Permet d’entrer dans le conteneur et effectuer des opérations.

- **docker stop id-conteneur**

  Arrêter le conteneur.

- **docker start id-conteneur**

  Démarrer le conteneur.

- **docker rm id-conteneur**

  Supprimer le conteneur

Docker

- **docker pull <nom image>**

  Récupérer des images sur le Docker Hub sans pour autant lancer de conteneur.

- **docker ps**

  Afficher les conteneurs en cours d'exécution.

- **docker ps -a**

  Afficher tous les conteneurs.

- **docker images -a**

  Afficher toutes les images présentes en local.

Docker

➢ **docker system prune**

va supprimer les données suivantes :

- l'ensemble des **conteneurs** Docker qui ne sont pas en status *running* ;
- l'ensemble des **réseaux** créés par Docker qui ne sont pas utilisés par au moins un conteneur ;
- l'ensemble des **images** Docker non utilisées ;
- l'ensemble des **caches** utilisés pour la création d'images Docker.

Docker

- **docker commit <id conteneur> <image name>**

  Va sauvegarder la nouvelle image.

- **docker save -o <path for generated tar file> <image name>**

  Sauvegarde une image compléte qui pourra être distribuée.

Docker

Le dockerfile:

- **FROM qui vous permet de définir l'image source ;**
- **RUN qui vous permet d'exécuter des commandes dans votre conteneur ;**
- **ADD qui vous permet d'ajouter des fichiers dans votre conteneur ;**
- **COPY qui vous permet d'ajouter des fichiers dans votre conteneur ;**
- **WORKDIR qui vous permet de définir votre répertoire de travail ;**
- **EXPOSE qui permet de définir les ports d'écoute par défaut ;**
- **VOLUME qui permet de définir les volumes utilisables ;**
- **CMD qui permet de définir la commande par défaut lors de l'exécution de vos conteneurs Docker.**

Docker build –t <nom image> .

Docker

Le dockerfile:

```
FROM ubuntu:latest
RUN apt-get update
RUN apt-get install –y nginx
VOLUME /var/www/html
ENTRYPOINT [« nginx »,  «-g », « daemon off ;»]
```

```
docker build –t monnginx:v1.0 .
```

```
Docker run –tid –p 80:80 –name web monnginx:v1.0
```

Docker

Les couches (Layers)

Les images peuvent partager des couches:

Image1

Layer 1

Layer 2

Layer 3

Layer 4

image 2

Layer A

Layer B

Layer D

Les conteneurs peuvent partager des couches:

Conteneur1

Layer 1

Layer 2

Layer 3

Layer 4

Conteneur 2

Layer A

Layer B

Layer D

- Dockerfile

```
FROM ubuntu:latest
RUN apt-get update
RUN apt-get -y install nano
RUN apt-get -y install git
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

- Dockerfile

```
FROM ubuntu:latest
MAINTAINER moi-meme
RUN apt-get update  \
&& apt-get -y install nano \
&& apt-get -y install git \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

Docker Volumes

- Pas de persistance de données
- Volumes : montage entre la machine host (qui héberge docker) et le conteneur

Docker Volumes

• docker volume create	Crée un volume
• docker volume inspect	Inspecte le volume
• docker volume ls	Liste les volumes sur le host
• docker volume prune	Supprime tous les volumes inutilisés
• docker volume rm	Supprime le volume

Docker Volumes

- docker volume create monvolume

- docker run -tid -p 80:80 -v monvolume:/usr/share/nginx/html --name web nginx:latest

Registry docker

- Registry cloud : docker hub, gitlab…
- Registry privé : n’importe quel serveur

Registry docker

- Contrôler le stockage des images
- Contrôle total du pipeline de distribution d'images
- Intégration étroite du stockage et la distribution des images dans le flux de travail de développement interne

Registry docker

- Docker hub
  - C’est la registry par défaut.
  - Gratuit et payant.
  - Limitation du plan gratuit:
    - Dépôts publics illimités
    - 1 seul dépôt privé

Registry docker

docker run –tid –p 80:80 –name web nginx:latest

Se connecter á votre compte docker hub

docker login

username: moufarreh

password: ********

Registry docker

docker tag nginx moufarreh/nginx:v1.0

docker push moufarreh/nginx:v1.0

Registry docker

- **Création du Registry privé**
- sur un serveur avec docker engine installé.
- La registry privé est un conteneur á partir d'une image:

```
docker run -d -p 5000:5000 --restart=always --name mon-registry registry:2
```

Registry docker

- **<u>Déposer une image dans le Registry privé</u>**

  - Créer une image personnalisée (alpine + git)

```
FROM alpine:latest

RUN apk add --no-cache git
```

  - Construire l'image

```
docker build -t alpinegit .
```

Registry docker

- Tagger l’image

```
docker tag alpinegit <SERVER NAME REGISTRY>:<PORT SERVER REGISTRY>/<CONTAINER NAME>
```

Soit:

```
docker tag alpinegit localhost:5000/alpinegit
```

- Pousser l’image dans la registry privé:

```
docker push localhost:5000/alpinegit
```

Registry docker

- Visualiser les images disponibles dans le Registry privé

```
curl -X GET http://localhost:5000/v2/_catalog
```

```
docker pull  localhost:5000/alpinegit
```

Docker-compose

Docker-compose

Docker-compose

- Orchestration des conteneurs
- Définir le comportement des conteneurs
- Exécuter des applications docker á conteneurs multiples
- Une seule commande pour démarrer tous les conteneurs de la config définie dans le docker-compose.yaml

Docker-compose

- Exemple de postgresql

docker run -tid -p 5432:5432 -e POSTGRES_PASSWORD="monpasse" -e PGDATA="/var/lib/postgresql/data" --name pgsql -v alten-pgdata:/var/lib/postgresql/data postgres:13

Docker-compose

```
version: "3"
services:
   sgbd:
      image:  postgres:13
      restart: always
      container_name: pgsql
      ports:
         - "5432:5432"
      volumes:
         - alten-pgdata:/var/lib/postgresql/data/
      environment:
         - POSTGRES_PASSWORD=monpasse
         - PGDATA=/var/lib/postgresql/data/
volumes:
      alten-pgdata :
         external: true
```

Docker-compose

- Un répertoire avec docker-compose.yaml
- Commandes similaires à docker
- Lancement du service:
  - docker-compose build
    uniquement construction des images
  - docker-compose up
    build et run des images
  - docker-compose up –d
    mode détaché (docker run –d)
  - docker-compose down
    stop et remove des services

Docker-compose

- docker-compose ps
  - État des services
- docker-compose start
- docker-compose stop
- docker-compose restart
- docker-compose rm

Docker-compose

- docker-compose pull

  maj des images
- docker-compose scale SERVICE=3

  lancer 3 instances de SERVICE

Docker-compose

docker-compose.yaml (.yml)

- version: "3"

  version de compose compatible avec la version du docker engine.

- services:
- image:
- restart:
  - restart: "no"
  - restart: always
  - restart: on-failure
  - restart: unless-stopped

Des questions ?

Docker Swarm

Docker Swarm

- **Qu'est-ce que Docker Swarm ?**

Docker Swarm est un outil de gestion d'orchestration de conteneurs qui permet de créer et de gérer un cluster de conteneurs Docker pour le déploiement d'applications. Il permet de répartir les conteneurs sur plusieurs hôtes pour une haute disponibilité et une mise à l'échelle horizontale.

Docker Swarm utilise des services pour définir l'état souhaité des applications.

Docker Swarm

- **Principaux concepts de Docker Swarm**
  - **Cluster Docker Swarm :** Un ensemble de machines, ou nœuds, qui exécutent des conteneurs Docker. Il y a des nœuds managers et des nœuds workers.
  - **Nœuds Managers :** Ils gèrent le cluster et ses services. Ils élisent un leader pour assurer une haute disponibilité.
  - **Nœuds Workers :** Ils exécutent les conteneurs des services déployés.
  - **Services et Tâches :** Un service définit un état souhaité, comme le nombre d'instances d'un conteneur. Chaque instance est une tâche dans le cluster.
  - **Réseau Swarm :** Permet la communication entre les conteneurs sur différents nœuds.
  - **Équilibrage de Charge :** Swarm peut équilibrer la charge entre les conteneurs d'un service.
  - **Mise à Jour Progressive :** Permet de mettre à jour les services avec un minimum d'interruption.

Docker Swarm

- **Mise en place d'un Cluster Swarm**
  - ➢ **Initialisation du Cluster Swarm**

```
$ docker swarm init --advertise-addr 192.168.0.10
Swarm initialized: current node (m05wlmwqhvue979pjl9y2855v) is now a manager.
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
46e6ywf2n42e346nkn77ojn0ddts53nh9hphqbap60dy5krpl4-dt86dfzb97lhc4vqqjswvcvmn
192.168.0.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

  - ➢ **Ajouter des nœuds workers (sur les machines workers, utiliser le token fourni par le manager) :**

```
 docker swarm join --token SWMTKN-1-
46e6ywf2n42e346nkn77ojn0ddts53nh9hphqbap60dy5krpl4-dt86dfzb97lhc4vqqjswvcvmn
192.168.0.10:2377
```

Docker Swarm

- **commandes les plus couramment utilisées dans la gestion d'un environnement Docker Swarm.**

Catégorie	Commande	Description
Gestion du Swarm	`docker swarm init`	Initialise un nouveau swarm.
	`docker swarm join`	Rejoint un swarm existant.
	`docker swarm leave`	Quitte le swarm.
	`docker swarm update`	Met à jour la configuration du swarm.
	`docker swarm join-token`	Gère les tokens pour rejoindre le swarm.
Gestion des Nœuds	`docker node ls`	Liste tous les nœuds du swarm.
	`docker node inspect`	Affiche des détails sur les nœuds.
	`docker node update`	Met à jour les paramètres d'un nœud.
	`docker node promote`	Élève un nœud worker au rang de manager.
	`docker node demote`	Rétrograde un nœud manager en worker.
	`docker node rm`	Supprime un nœud du swarm.
Gestion des Services	`docker service create`	Crée un nouveau service.
	`docker service ls`	Liste les services.
	`docker service inspect`	Affiche des détails sur un service.
	`docker service update`	Met à jour un service.
	`docker service scale`	Échelonne un service.
	`docker service ps`	Liste les tâches d'un service.
	`docker service logs`	Affiche les logs d'un service.
	`docker service rm`	Supprime un service.

Catégorie	Commande	Description
Gestion des Stacks	`docker stack ls`	Liste toutes les stacks.
	`docker stack deploy`	Déploie une stack.
	`docker stack services`	Liste les services d'une stack.
	`docker stack ps`	Liste les tâches d'une stack.
	`docker stack rm`	Supprime une stack.
Gestion des Secrets	`docker secret create`	Crée un nouveau secret.
	`docker secret ls`	Liste les secrets.
	`docker secret inspect`	Affiche des détails sur un secret.
	`docker secret rm`	Supprime un secret.
Gestion des Configs	`docker config create`	Crée une nouvelle configuration.
	`docker config ls`	Liste les configurations.
	`docker config inspect`	Affiche des détails sur une configuration.
	`docker config rm`	Supprime une configuration.

Docker Swarm

- Déploiement d'un Service NGINX

```
docker service create --name my_nginx_service --publish 80:80 nginx
```

- Scale d'un Service

```
docker service scale my_nginx_service=3
```

Augmenter le nombre de réplicas de my_nginx_service à 3

Docker Swarm

- Déploiement d'une Stack
  - Pour déployer une stack, vous utilisez généralement un fichier docker-compose.yml.

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
```

```
docker stack deploy -c docker-compose.yml my_stack
```

Docker Swarm

- Scale d'une Stack
  - Le scaling d'une stack se fait en modifiant le fichier docker-compose.yml. Par exemple, pour augmenter le nombre de réplicas du service web à 3,

```
version: '3'
services:
  web:
    image: nginx
    deploy:
      replicas: 3
    ports:
      - "80:80"
```

```
docker stack deploy -c docker-compose.yml my_stack
```