CONTENEURISATION

Module: Cloud Computing

Master IAO
Dept. Informatique

Novembre 2020

Pr. Oussama Mohamed REDA
Intrevenant:
Dr. Y.T.Benjelloune

Rappel sur les Machines Virtuelles

- Historiquement, quand nous avions besoin de serveurs, nous achetions des serveurs physiques avec une quantité définie de CPU, de mémoire RAM ou de stockage sur le disque.
- Or, on avait souvent besoin d'avoir de la puissance supplémentaire pour des périodes de forte charge (fête de Noël, par exemple). Ainsi, vous deviez acheter plus de serveurs pour répondre aux pics d'utilisation. Une solution a ainsi été créée : la ***machine virtuelle***.

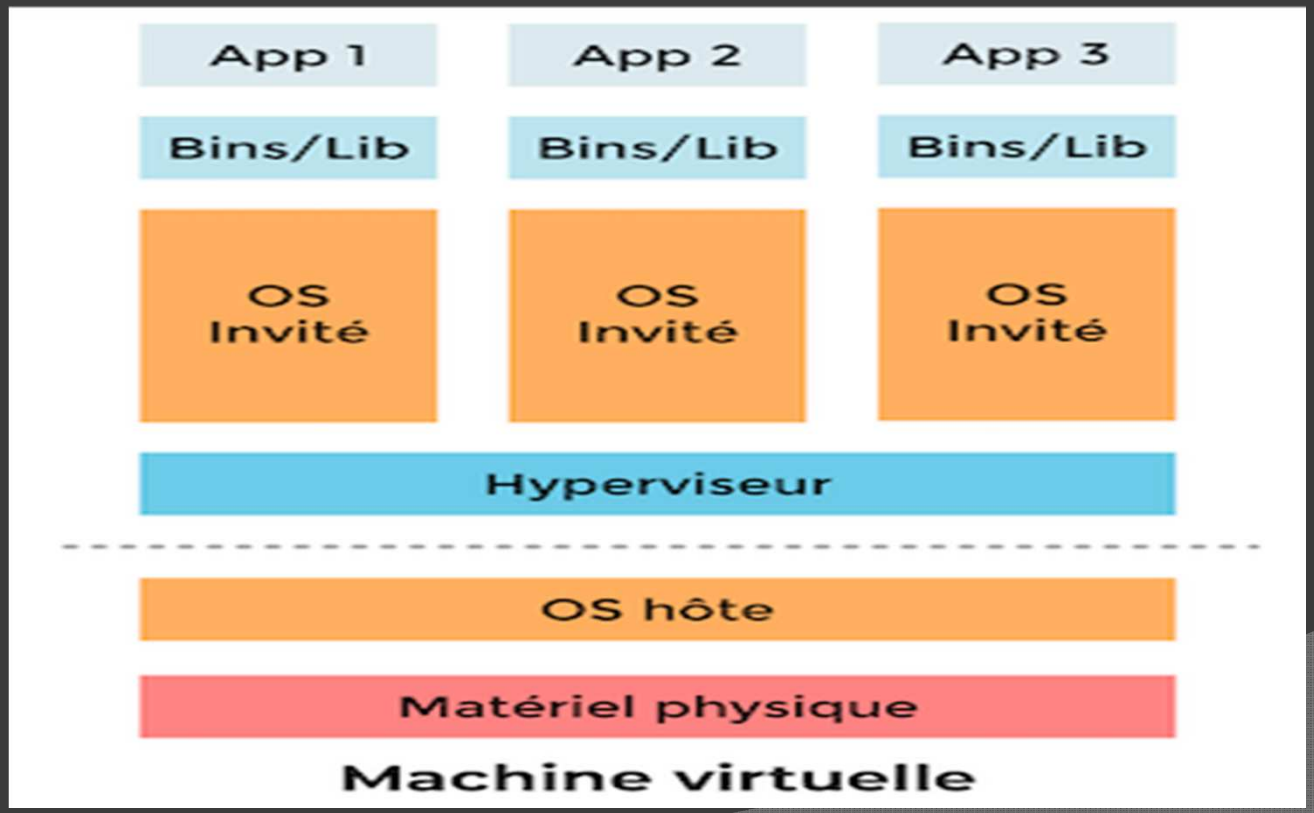
Rappel sur les Machines Virtuelles

- Une VM est un environnement d'exploitation ou d'applications installé sur logiciel, qui permet à l'utilisateur d'emuler son experience de la même façon que sur une machine physique
- Lancer plusieurs environnement d'OS sur la même machine
- Simplifier les backup et les restaurations
- ***Cependant les hyperviseurs de VM reposent sur une emulation hardware et requièrent beaucoup de puissances***
- L'intérêt des VMs est principalement la souplesse et l'optimisation de l'utilisation des ressources matérielles.
- L'organisation en VMs rend plus facile la réaffectation, le changement du dimensionnement, et améliore le taux d'utilisation des dispositifs physiques (disque, mémoire, réseau, etc.).
- ***L'inconvenient des VM est d'être assez gourmandes en ressources, puisqu'il faut, à chaque fois, faire tourner un système d'exploitation complet, avec tout ce que cela implique, en terme d'emprise mémoire notamment.***

Rappel sur les Machines Virtuelles

- Lorsque vous utilisez une machine virtuelle (VM), vous faites ce qu'on appelle de la ***virtualisation lourde***. En effet, vous recréez un système complet dans le système hôte, pour qu'il ait ses propres ressources.
- **L'isolation avec le système hôte est donc totale** ; cependant, cela apporte plusieurs contraintes :
  - une machine virtuelle prend du **temps à démarrer** ;
  - une machine virtuelle **réserve les ressources** (CPU/RAM) sur le système hôte.
- Par ailleurs cette solution présente aussi de nombreux avantages :
  - une machine virtuelle est totalement **isolée** du système hôte ;
  - les ressources attribuées à une machine virtuelle lui sont totalement **réservées** ;
  - vous pouvez installer **différents OS** (Linux, Windows, BSD, etc.).
- Mais il arrive très souvent que l'application qu'elle fait tourner ne consomme pas l'ensemble des ressources disponibles sur la machine virtuelle. Ainsi est né un nouveau système de virtualisation plus léger : les **conteneurs**.

Architecture d'une Machine Virtuelle



Les Conteneurs

- **Virtualisation d'OS**: les applications de l'os ont la perception qu'ils sont dans leur SE avec leurs librairies, et tous les fichiers necessaires à l'exection des processus : code, routines, bibliotheques et parametres,
- Pour les conteneurs c'est différent, on n'installe pas d'OS à proprement parler, mais un rootfs (le / d'un unix/Linux) qui est appelé image, qui contient les bibliothèques et les binaires nécessaires. Le noyau quant à lui, est partagé avec le système hôte. Nous pouvons évidemment limiter les ressources des conteneurs.
- La virtualisation consiste à exécuter de nombreux SE sur un seul et meme système , cependant les containers se partagent le même noyau de SE et isolent les processus de l'application du reste du système.
- Package déclaratif
- Unité de déploiement universelle: les applications peuvent etre deploye sur n'importe quel infrastructure (machine locale, serveur , Cloud computing)
- Les containers sont plus efficient que les hyperviseurs en termes de consommation des ressources **systemes, du fait que l'hyperviseur virtualise le hardware mais le conteneur vistrualise le SE**.
- Un conteneur est simplement utilisé dans une VM pour uniformiser une application entre les différents environnements (prod, préprod, intégration, etc.). Il arrive même de trouver dans une VM un seul conteneur.
- Le plus gros défaut des conteneurs, c'est le fait que ce n'est pas *cross-platform*. On lance des conteneurs Linux sous Linux, des conteneurs BSD sous BSD ou des conteneurs Windows sous Windows.

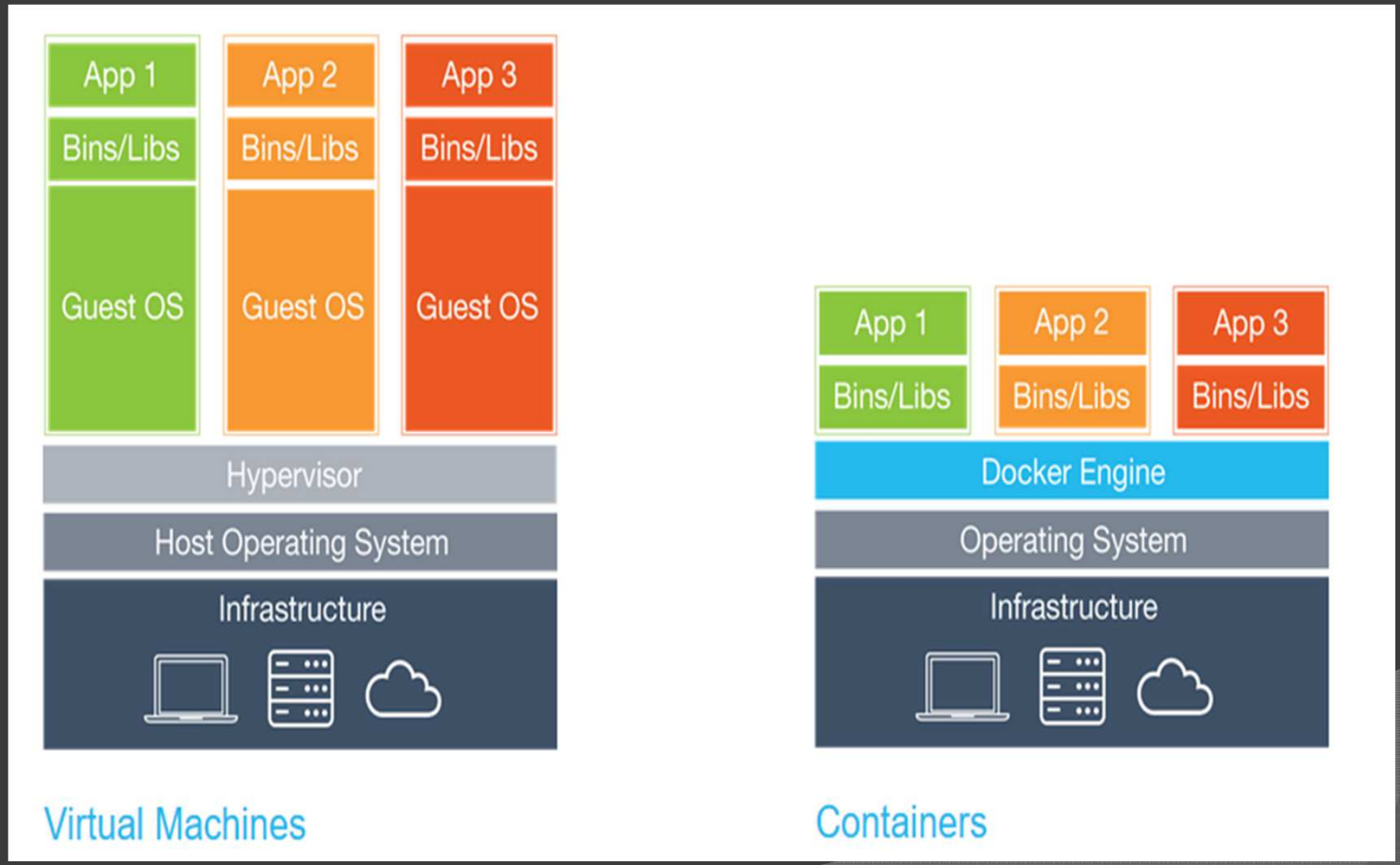
Les Conteneurs

- Un conteneur Linux est un **processus** ou tourne 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 conteneurs, telsque *OpenVZ et LXC, apportent une **isolation importante des processus systèmes*** ; cependant, les ressources CPU, RAM et disque sont totalement partagées avec l'ensemble du système.
- Les conteneurs ***partagent entre eux le kernel Linux*** ; ainsi, il n'est pas possible de faire fonctionner un système Windows ou BSD dans celui-ci.
- Un **conteneur** doit être léger, il ne faut pas ajouter de contenu superflu dans celui-ci afin de le **démarrer rapidement**, mais il apporte une **isolation moindre que la VM.** A contrario, les **machines virtuelles** offrent une très **bonne isolation**, mais elle sont globalement plus **lentes** et bien plus **lourdes**.

Les avantages des conteneurs

- **Ne réserve que les ressources nécessaires**
  - 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.
- **Démarrez rapidement vos conteneurs**
  - 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.
- **Donnez plus d'autonomie à vos développeurs**
  - En dehors de la question pécuniaire, il y a aussi la possibilité de faire tourner des conteneurs sur le poste des développeurs, et ainsi de réduire les différences entre la "sainte" production, et l'environnement local sur le poste des développeurs.
- Les conteneurs permettent de **réduire les coûts**, d'augmenter la **densité de l'infrastructure**, tout en améliorant le cycle de déploiement.

Container VS VM



Solutions de conteneurisation

- Docker
- Kubernates

Docker

- Docker
- Docker-compose
- Docker-swarm

DOCKER

- Docker (ou, très précisément, le *docker engine*) est un programme qui va nous permettre de créer des conteneurs Linux et d'y installer des environnements prêts à l'emploi, les *images*.
- Ainsi, Docker propose une solution beaucoup plus légère, basée sur la capacité du système Linux à créer des espaces isolés auxquels on affecte une partie des ressources de la machine-hôte.
- Ces espaces, ou *containers* partitionnent en quelque sorte le système-hôte en sous-systèmes étanches, au sein desquels le nommage (des processus, des utilisateurs, des ports réseaux) est purement local.
- Par exemple : On peut par exemple faire tourner un processus *apache* sur le port 80 du conteneur A, un autre processus *apache* sur le port 80 du conteneur B, sans conflit ni confusion.
- Tous les nommages sont en quelque sorte interprétés par rapport à un container donné (notion *d'espace de nom*)

Caractéristiques de Docker

- Le déploiement : puisque Docker a pour vocation de conteneuriser des applications, il devient simple de créer un conteneur pour notre application, et la dispatcher où bon nous semble. Un conteneur qui fonctionne sur une machine avec une distribution X, fonctionnera sur une autre machine avec une distribution Y.
- Emulation: Docker permet d'émuler un système distribué de serveurs.
- Le développement : cela permet de facilement avoir le même environnement de développement qu'en production, si ça marche quelque part, ça marchera partout. Cela permet également de pouvoir sur la même machine, tester avec plusieurs versions d'un même logiciel. Par exemple pour une application PHP, on pourrait facilement tester sur plusieurs versions de PHP, puis plusieurs versions de nginx et d'autres serveurs web.
- Installer des applications : étant donné que Docker propose une multitude d'outils, vous allez voir à quel point il est facile et rapide d'installer une application, bien souvent une seule ligne de commande suffit pour avoir par exemple notre nextcloud fonctionnel. (nextcloud est sous forme d'image qui est téléchargé et installe dans notre conteneur)
- Docker s'exécute *toujours* dans un environnement de type Unix: Linux ou Mac OS X. Si vous avez une machine sous Windows, Docker propose une machine virtuelle Linux ultra-légère au sein de laquelle il s'exécute.

Les composants de docker

- Docker Engine
- Docker File
- Docker image
- Docker container
- Docker-Compose
- Docker-Swarm

Les composants du Docker : Docker Engine

- *Docker engine* ou *moteur docker* est le programme qui gère les conteneurs; il s'exécute dans la Docker VM,
- *Docker VM* désigne le système Linux où Docker s'exécute, qui peut donc être soit directement votre machine si vous êtes sous Linux, soit une VM Linux s'exécutant sur votre machine Windows ou Mac OS.
- Le Docker Engine est un outil client-serv*eur* sur lequel repose la technologie de container pour prendre en charge les tâches de création d'applications basées container.
- Le moteur crée un processus daemon server-side permettant d'héberger les images, les containers, les réseaux et les volumes de stockage.
- Ce daemon fournit aussi une interface SLI client-side permettant aux utilisateurs d'interagir avec le daemon via l'API de la plateforme.

Les composants du docker

- Le DockerFile est un fichier de configuration qui contient toutes les instructions pour créer une image, comme des métadonnées (Mainteneur, label, etc.), ou même les commandes à exécuter pour installer un logiciel.
- DockerFile decrit ce qui doit etre installé dans le systeme.
- Par analogie avec Java c'est le fichier source
- Un "Dockerfile" peut être inclus dans d'autres "Dockerfile", et être à la base de plusieurs images différentes.
- Par exemple, si tous vos projets utilisent MySQL comme SGDB, vous pouvez créer un "Dockerfile" qui installe MySQL, et ensuite créer un autre "Dockerfile" pour chacun de vos projets.
- Si vous mettez à jour votre "Dockerfile" MySQL, vos images projets pourront êtres mises à jour en même temps. Vous pouvez aussi utiliser une même image pour créer plusieurs conteneurs différents, mais avec les mêmes propriétés (pensez aux instances de classes)

Les composants du docker:
Image Docker

- Une image Docker représente le système de fichiers, sans les processus.
- Elle contient tout ce que vous avez décidé d'y installer (Java, une base de donnée, un script que vous allez lancer, etc…), mais est dans un état inerte. Les images sont créées à partir de fichiers de configuration, nommés "Dockerfile".
- Docker va un peu plus loin en proposant des installations pré-configurées, empaquetées de manière à pouvoir être placées très facilement dans un conteneur. On les appelle des *images*.
- On peut ainsi trouver des images avec :
  - Web (Apache, njinx),
  - serveurs NoSQL (mongodb, cassandra),
  - moteurs de recherche (ElasticSearch, Solr).

L'installation d'une image se fait très simplement, et soulage considérablement des tâches parfois pénibles d'installation directe.

- Par analogie avec Java, l'image docker est le fichier compilé de Java

Les composants du docker: Le conteneur

- Un conteneur est l'exécution ou le déploiement d'une image : il possède la copie du système de fichiers de l'image, ainsi que la capacité de lancer des processus.
- En gros, le conteneur est un OS leger, avec lequel vous pouvez interagir.
- Dans ce conteneur, vous allez donc pouvoir interagir avec les applications installées dans l'image, exécuter des scripts, faire tourner un serveur, etc.
- Un conteneur Docker peut donc être vu comme un sous-système Linux autonome, mobilisant très peu de ressources car l'essentiel des tâches système est délégué au système Linux dans lequel il est instancié.
- On dispose donc virtuellement d'un moyen de multiplier à peu de frais des pseudo-machines dans lesquelles on pourrait installer « à la main » des logiciels divers et variés.
- Pour faire l'analogie avec le monde Java (ou le monde des langages objets en général), container est une instance de la classe.

Les images docker, constituant un pseudo système distribué

Une image se place dans un conteneur. On peut placer la même image dans plusieurs conteneurs et obtenir ainsi un système distribué. Examinons la Figure_ montrant une configuration complète, avec un système-hôte Windows. Nous avons tous les composants à l'œuvre, essayons de bien comprendre par les étapes suivantes:

1- Le système hôte a créé une VM Docker (donc, une machine virtuelle Linux dans laquelle s'exécute le *Docker Engine*).

2- Docker a téléchargé (nous verrons comment plus tard) les images de plusieurs systèmes de gestion de données: MySQL, MongoDB (un système NoSQL que nous étudierons), et Solr.

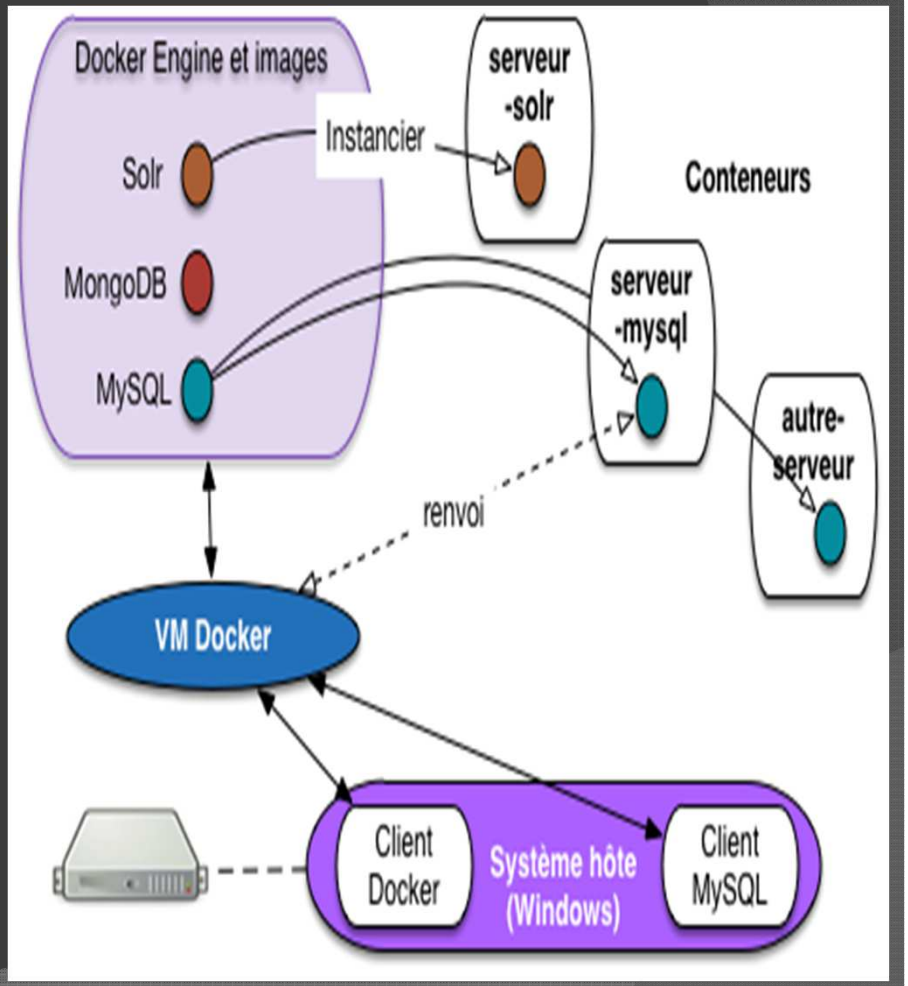
3- Ces images ont été *instanciées* dans des conteneurs A, B et C. L'instanciation consiste à installer l'image dans le conteneur et à l'exécuter. Nous avons donc deux conteneurs avec l'image MySQL, et un troisième avec l'image Solr.

l'ensemble constitue donc un système distribué virtuel, le tout s'exécutant sur la machine-hôte et gérable très facilement grâce aux utilitaires Docker. Nous avons par exemple dans chaque conteneur un serveur MySQL.

Maintenant, on peut se connecter à ces serveurs à partir de la machine-hôte avec une application cliente (par exemple phpMyAdmin) et tester le système distribué. (objet du TP)

On peut instancier l'image de MongoDB dans 4 conteneurs et obtenir un *cluster* MongoDB en quelques minutes.

Evidemment, les performances globales ne dépasseront pas celle de l'unique machine hébergeant Docker. Mais pour du développement ou de l'expérimentation, c'est suffisant, et le gain en temps d'installation est considérable.



Les composants du Docker: Docker Compose

- Le composant Docker Compose permet de définir la composition des services au sein d'un container dédié.
- Docker compose est un outil anciennement tiers puis racheté par docker, qui permet de composer une stack ou une infrastructure complète de conteneurs.
- Celui-ci permet de simplifier la création, l'interconnexion et la multiplication de conteneurs.
- En gros nous créons un fichier avec extension yml (Docker-compose.yml) qui nous permettra de gérer un ensemble de conteneurs en quelques commandes.

Les composants du Docker: Docker Hub

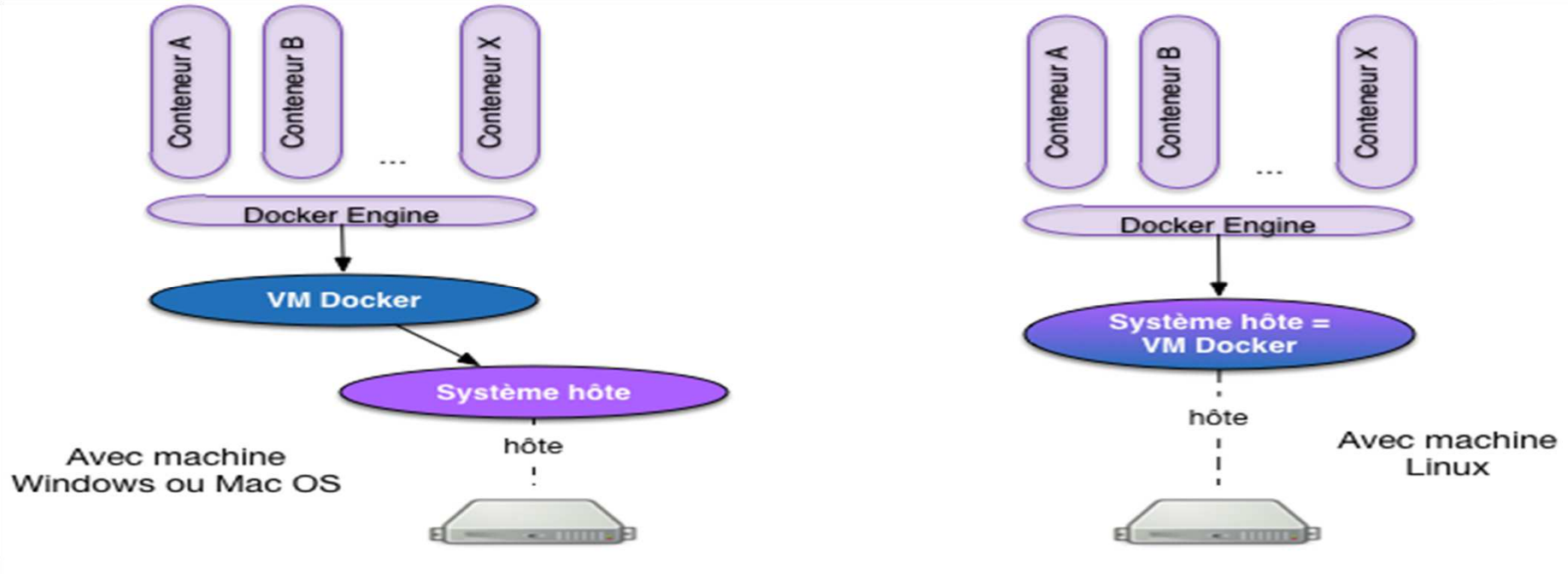
- Le Docker Hub est un outil SaaS permettant aux utilisateurs de publier et de partager des applications basées container via une bibliothèque commune.

Le Client Docker

- Le *client Docker* est l'utilitaire grâce auquel on transmet au moteur les commandes de gestion de ces conteneurs.
- Il peut s'agir soit de la ligne de commande (*Docker CLI*) ou de *kitematic (docker sur mac)*.

L'architecture Docker, sous différents systèmes d'exploitation

À gauche, la machine hôte est sous Windows. Docker (engine) va commencer par lancer une VM Linux, et s'exécuter en tant que programme dans cette VM. Grâce à l'interface de commande du programme Docker, ou son interface graphique *kitematic*, on pourra alors créer des conteneurs.
À droite, le schéma est lègèrement simplifié car l'hôte est un serveur Linux. On peut alors y exécuter directement le moteur Docker.



Exemple : Hello-world

- A titre d'exemple quand on execute une image, telque
- $# Docker run hello-world
- docker execute les operations suivantes en arriere plan:

  1- le Docker client contacte le daemon docker

  2- Le daemon docker charge l'image 'hello-world' du Hub Docker (Docker Hub pour amd 64)

  3- le daemon Docker cree un nouveau container de l'image qui tourne l'executable et qui produit le resultat

  4- le daemon Docker envoi la sortie en streaming au client Docker, qui l'envoi sur le terminal.

Orchestration des conteneurs docker

- Les outils d'orchestration des conteneurs docker:
  - Docker-compose
  - Swarm

Orchestration des dockers

- L'orchestration des dockers s'avere une tache ardue, alors docker-compose est un outil qui permet de décrire dans un fichier yml plusieurs contenaires comme un ensemble de services.
- Docker-compose est un outil qui permet un déploiement facile de l'ensemble des contenaires sur l'environnement de productions.

Docker-Compose

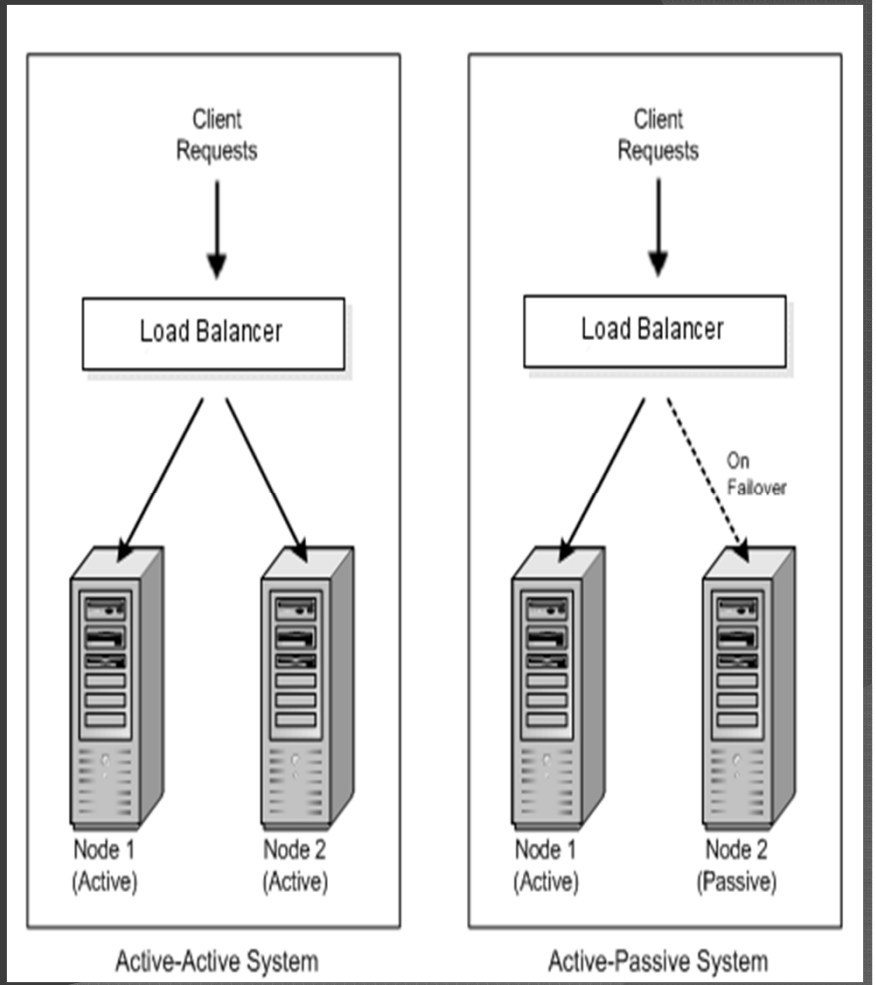
- Docker-Compose est un **outil écrit en Python** qui permet de décrire, dans un **fichier YAML**, plusieurs conteneurs comme un **ensemble de services**.
- Docker Compose vous permet de définir des **applications multi-conteneurs**, appelées piles (*Stacks*), et de les exécuter soit sur un seul nœud Docker, soit dans un cluster.
- L'outil fournit des programmes en ligne de commande que vous pouvez utiliser pour gérer le cycle de vie complet de vos applications.
- Docker définit les piles comme des groupes de services interconnectés qui partagent des dépendances logicielles et qui sont orchestrées et mises à l'échelle ensemble.
- Une pile de dockers vous permet de stocker les différentes fonctionnalités d'une application dans un fichier central : le docker-compose.yml.
- Démarrez-le à partir de là, exécutez-le dans un environnement d'exécution isolé et gérez-le de manière centralisée.

Rappel: cluster

- Un cluster designe un groupe de serveurs vus de l'exterieur comme un seul serveur logique.
- Le cluster repond a un double besoin :
  - Demande de traitement des applications constantes auquel un seul serveur peut difficilement repondre
  - Demande forte de haute disponibilte des applications ce qui amene à la redondance des serveurs pour garantir les services mais aussi pour se premunir des pannes

Rappel: Types de Clustering

- Cluster Actif/passif :
  - le principe est de doubler un serveur avec un serveur similaire ainsi les deux serveurs fonctionnent mais un seul qui traite les requêtes.
  - Le cluster Actif/passif réponds a un besoin de disponibilité mais pas de montée en charge
- Cluster Actif/Actif
  - Le principe est de redonder le principe actif avec d'autres serveurs actif similaires ce qui fournit un meilleur disponibilité puisque le cluster repose sur plusieurs serveurs et non plus un seul
  - Dans ce type de cluster tous les serveurs sont actifs donc la charge de travail est repartie entre ses serveurs actifs et si un serveur tombe se sont les autres serveurs qui prennent le relai et supporter une montée en charge pour compenser la défaillance du serveur indisponible
  - Cependant l'acces est concurrentiel dans un cluster actif/actif comme dans le cas par exemple des base de donnees ou l'acces par sessions à un serveur web ce qui necessite une gestion de charge load balancing.



Rappel: Cluster et Load Balancing

- La répartition des charges est de 2 types différents :
- Répartition statique:
  - répartition de charges se fait en amont le démarrage des serveurs.
  - la répartition se fait selon plusieurs criteres : la localisation, la fonction, ou meme par addresses IP dans ce cas les clients se connectent via des adresses URL.
  - Cependant, la charge n'est pas équilibré dans le cas ou plusieurs utilisateurs lancent une requete pour un serveur, et ne couvre pas la disponibilité car si un serveur tombe en panne les clients vont être bloqués.
  - Certes on peut remedier à ce probleme en installant un serveur passif mais cela va augmenter le cout

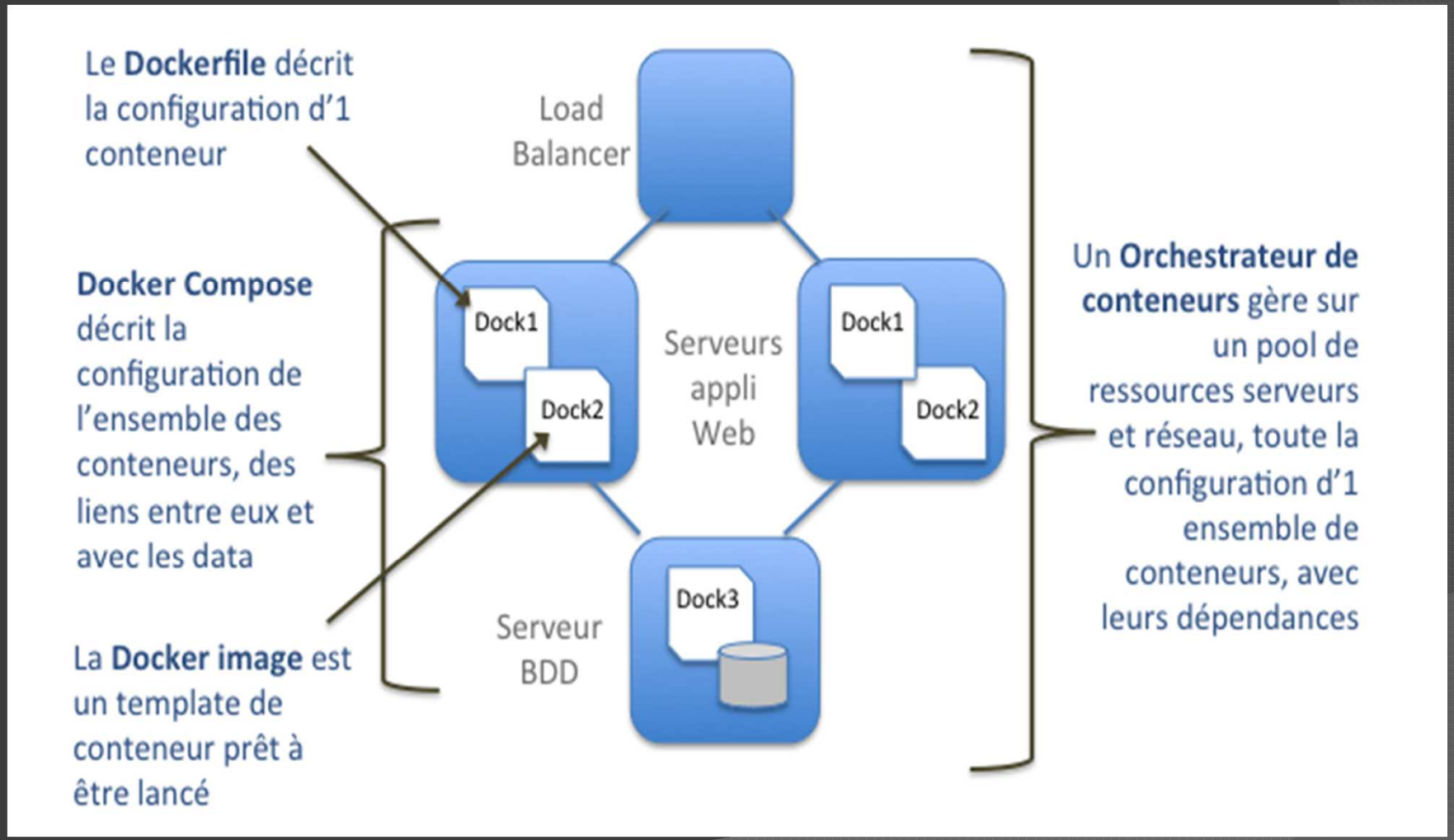
Rappel: Cluster et Load Balancing

- Répartition de charges dynamiques:
  - On utilise un load balancer (repartiteur de charge) qui est un équipement spécifique entre les différents serveurs
  - le répartiteur de charges vérifie la disponibilité des serveurs pour ne pas envoyer de requetes vers des serveurs indisponibles.
  - Le load balancing se base sur des algorithmes de repartition des charges :
    - Round Robin: les requetes sont orientés vers les serveurs à tour de role
    - Round Robin pondéré :pour les serveurs qui n'ont pas la meme puissance de traitement, chaque serveur a un poids.
    - Least connection: l'algorithme assigne les requetes aux serveurs qui execute le moins
  - Les caracteristiques d'un load balancer :
    - la repartition des charges entre les serveurs du cluster se fait de maniere precise
    - Tolerance aux pannes: si un serveur tombe en panne un autres prend le relai
    - La transparence de la repartition vis-à-vis de l'utilisateur puisque les utilisateurs ne sont pas liés au meme serveur

Docker-Compose

- Compose : C'est un outil qui permet de définir et exécuter des applications multi-conteneurs. Les conteneurs sont idéaux pour des applications simple micro services.
- C'est pour cela qu'il est idéal de faire une interconnexion entre plusieurs conteneurs.
- L'exemple le plus parlant et celui d'un serveur web contenant les service **php, mysql et nginx** dans des containers différents (voir le TP).
- Docker-Compose va vous permettre d'orchestrer vos conteneurs, et ainsi de simplifier vos déploiements sur de multiples environnements.

Architecture d'une orchestration par Docker Compose



CLI de Docker Compose

- La CLI de Docker est très proche de celle de CLI de Docker compose.
- Pour utiliser le CLI (Command Line Interface) de Docker Compose, nous avons besoin d'un fichier docker-compose.yml
- Pour récupérer l'ensemble des images décrites dans votre fichier docker-compose.yml et les télécharger depuis le Docker Hub, on execute la commande : docker-compose pull
- Cette commande est execute dans le dossier ou se trouve le fichier docker-compose.yml

Démarrer une stack Docker Compose

- Un ***stack*** est un ensemble de conteneurs Docker lancés via un seul et unique fichier.
- Pour lancer la création de l'ensemble des conteneurs, vous devez lancer la commande : **docker-compose up** (pour rappel, vous faites un **docker run** pour lancer un seul conteneur).
  NB: Vous pouvez ajouter l'argument -d pour faire **tourner les conteneurs en tâche de fond**.

le statut d'une stack Docker Compose

- Après avoir démarré une stack Docker Compose, vous aurez certainement besoin de voir si l'ensemble des conteneurs sont bien dans un état fonctionnel, et prêts à rendre un service.
- la commande **docker-compose ps**
- On ajoute aussi l'option -a pour lister les conteneurs actifs

Les logs d'une stack Docker Compose

- Pour visualiser les **logs** de vos conteneurs.
- Pour cela, vous devez utiliser la commande
- **docker-compose logs -f --tail 5**
- Cette commande permet de voir l'ensemble des logs sur les différents conteneurs de façon continue, tout en limitant l'affichage aux 5 premières lignes.

Valider un stack Docker Compose

- Lors de l'écriture d'un fichier docker-compose, nous ne sommes pas à l'abri d'une **erreur**.
- Pour éviter au maximum cela, vous devez utiliser la commande :
- **docker-compose config**
- qui vous permettra de **valider la syntaxe de votre fichier**, ainsi d'être certain de son bon fonctionnement.

Arreter un stack Docker Compose

- Pour arrêter une **stack Docker Compose**, vous devez utiliser la commande :
- **docker-compose stop**
- Cependant, celle-ci ne supprimera pas les différentes ressources créées par votre stack.
- Ainsi, si vous lancez à nouveau un **docker-compose up -d**, l'ensemble de votre stack sera tout de suite à nouveau fonctionnelle.
- Pour **supprimer** l'ensemble de la stack Docker Compose, vous devez utiliser la commande : **docker-compose down** qui **détruira l'ensemble des ressources** crées.

Récapitulatif des commandes principales pour utiliser une stack Docker Compose

- **docker-compose up -d** vous permettra de **démarrer** l'ensemble des conteneurs en arrière-plan ;
- **docker-compose ps** vous permettra de voir le **status** de l'ensemble de votre stack ;
- **docker-compose logs -f --tail 5** vous permettra d'afficher les logs de votre stack ;
- **docker-compose stop** vous permettra d'**arrêter** l'ensemble des services d'une stack ;
- **docker-compose down** vous permettra de **détruire** l'ensemble des ressources d'une stack ;
- **docker-compose config** vous permettra de **valider** la syntaxe de votre fichier docker-compose.yml.

Cluster de docker

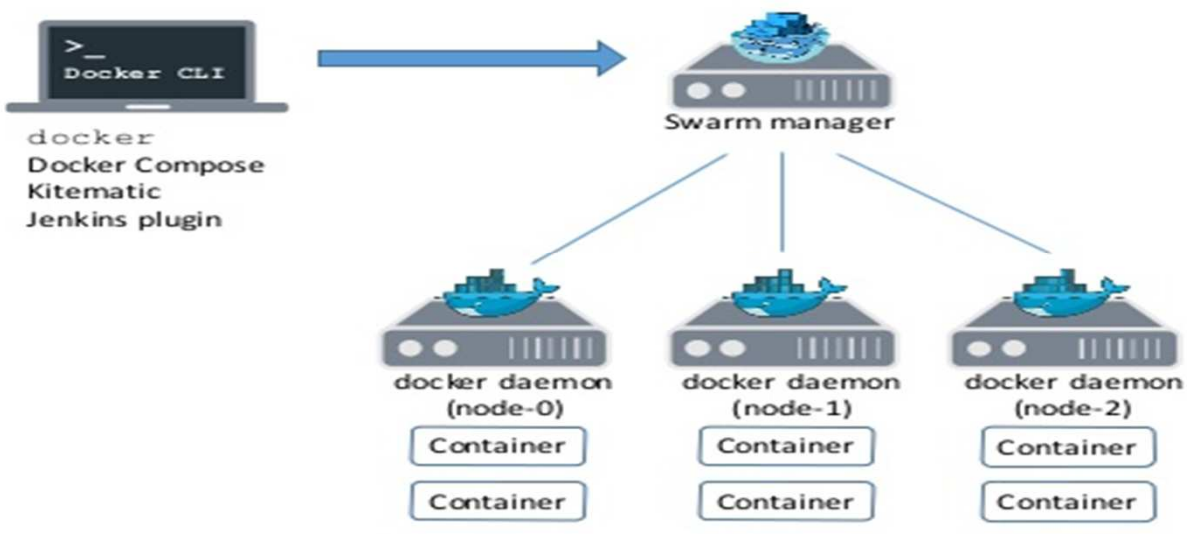
- Tandis que le docker engine est fait pour lancer des applications dans des containers, il lui manque de sérieuses fonctionnalités pour aller en production.
- La principale est son exécution sur une seule machine, pas de mode distribué, le container est attaché à un seul nœud !
- La seconde est sa fragilité puisqu'il ne peut pas compter sur une haute disponibilité de son infrastructure. En résumé, si on perd le nœud exécutant les containers, on perd tout !

ORCHESTRATION DES CONTENEURS PAR DOCKER-SWARM

Orchestration des containers

Pour gerer un cluster de Docker (Docker Clusters) on considere deux parties majeurs :

- Les Noeuds gerants (Manager Nodes): gerent les clusters en maintenant l'etat de clusters, preparation des services, et mode endpoints de swarm (HTTP API) qui represente la clé caracteristique pour le stokage des donnees critiques sur le cluster swarm.
- les noeuds travailleurs (Worker Nodes): qui executent les containers, et ne rentrent pas dans les decisions du planning.
- Le node travailleur doit avoir au moins un node manager.
- Il est possible de transformer un noeud travailleur en manager quand ce dernier est en maintenance



Orchestration des containers

- Docker Swarm est basé sur une **architecture maître-esclave**.
- Le Docker Swarm est un cluster de machines hébergeant docker Engine, connecté par un réseau Overlay qui va gérer le service de découverte.
- Chaque cluster Docker (l'essaim) se compose d'au moins un nœud de gestion (manager) et d'un nombre quelconque de nœuds de type « Workers ».
- le gestionnaire (Swarm Manager) est responsable de la gestion du cluster et de la répartition des tâches.
- les « Swarm Workers » prennent en charge l'exécution des unités de travail « Tasks ».
- Les applications de conteneurs sont distribuées en un certain nombre de nœuds Docker en tant que « Services ».

Docker Swarm

- Depuis la version 1.13, la façon de créer un cluster avec Docker Swarm a été complètement révolutionnée et simplifiée.
- Swarm (essaim en français) est un logiciel développé par Docker et permet la gestion des conteneurs **en mode centralisé du cluster** appelé l'orchestration des conteneurs.
- le gestionnaire de containers en mode cluster (appelé Orchestrateur) est directement embarqué avec le moteur, donc n'a pas besoin d'installation.
- Un essaim est un nombre quelconque de moteurs Docker en mode Swarm. Chaque moteur Docker fonctionne sur un nœud séparé et l'intègre dans le cluster.
- Jusqu'à la version 1.11 de Docker, Swarm devait être implémenté comme un outil séparé. Les nouvelles versions de la plateforme de conteneur prennent en charge un mode **Swarm** natif.
- Le cluster manager, est ainsi à la disposition de chaque utilisateur Docker avec l'installation du moteur Docker.

Le Docker swarm

- Le Docker Swarm est un cluster de machines hébergeant docker Engine, connecté par un réseau Overlay qui va gérer le service de découverte.
- Un cluster peut avoir un ou plusieurs managers mais aussi un ou plusieurs workers (quoique un cluster de 1 nœud n'a pas vraiment de sens).
- Attention, les nœuds workers ne peuvent pas visualiser ou modifier la configuration du cluster, seuls les managers peuvent le faire.
- Lorsque les nœuds Docker ne fonctionnent pas dans un cluster Swarm, on dit qu'ils sont dans un "single-engine mode".
- Dès qu'ils sont intégrés dans un cluster ils fonctionnent en "swarm mode".

Domaines d'application de Docker Swarm

- Un domaine d'application central de Docker Swarm est la répartition de la charge.
- Le load balancing (Equilibrage de charge) est une technique qui permet de distribuer une charge entres plusieurs serveurs appartenant au même groupe ou **cluster** (appelé aussi ferme de serveur).
- Il s'agit d'une technologie efficace pour optimiser la qualité de services.
- En mode Swarm, Docker dispose de **fonctions de répartition de charge (load balancing) intégrées**.
- Par exemple, si vous exécutez un serveur Web NGINX avec 4 instances, Docker distribue intelligemment les requêtes entrantes aux instances du serveur Web disponibles.

Les Services dans Docker Swarm

- Le terme « service » se réfère à une structure abstraite avec laquelle vous définissez les tâches qui doivent être exécutées dans un cluster.
- Chaque service consiste en un ensemble de tâches individuelles, dont chacune est traitée dans son propre conteneur sur l'un des nœuds du cluster.
- Docker Swarm supporte deux modes dans lesquelles les services Swarm sont définis :
  - **Services répliqués :** un service répliqué est une tâche qui s'exécute dans un nombre de répliques défini par l'utilisateur. **Chaque réplique est une instance du conteneur Docker défini dans le service**. Les services répliqués peuvent être mis à l'échelle en créant des répliques supplémentaires. Par exemple, un serveur Web comme NGINX peut être mis à l'échelle à 2,4 ou 100 instances avec une seule ligne de commande.
  - **Services globaux :** lorsqu'un service s'exécute en mode global, chaque nœud disponible dans le cluster démarre une tâche pour le service correspondant. Lorsqu'un nouveau nœud est ajouté au cluster, le gestionnaire Swarm lui assigne immédiatement une tâche pour le service global. Les services globaux sont adaptés, par exemple, à la surveillance d'applications ou de programmes anti-virus.

Processus d’un cluster Docker avec Swarm

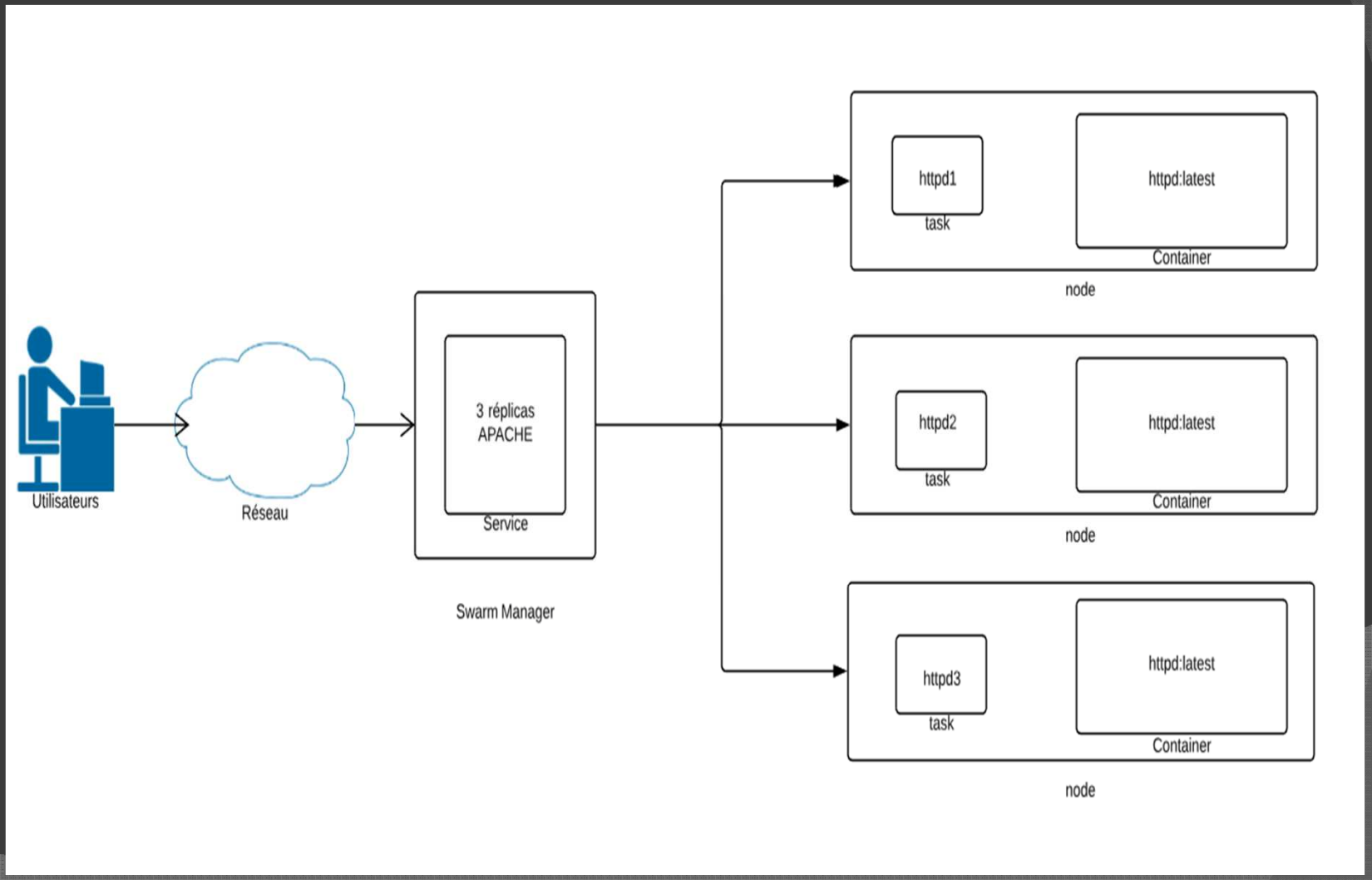
- La création d’un cluster Docker (Docker swarm) comprend trois étapes :

1-Implémenter les hôtes Docker

2- Initialiser Swarm

3- Intégrer les hôtes Docker dans l‘essaim

Exemple de Docker Swarm en mode replique



Docker-Swarm: implémenter les hôtes dockers

- L'outil d'approvisionnement **Docker Machine** est recommandé pour le déploiement des nœuds Docker.
- Cela simplifie l'implémentation des hôtes Docker (aussi les « Dockerized hosts », les hôtes virtuels, y compris le moteur Docker).
- Avec Docker Machine vous implémentez des comptes pour votre essaim sur n'importe quelle infrastructure et pouvez les gérer à distance.
- **Les plugins de pilote pour Docker Machine**, sont fournis par différentes plateformes de Cloud.
- Ceci réduit l'effort de provisionnement des hôtes Docker chez les fournisseurs tels qu'Amazon Web Services (AWS) ou Digital Ocean via une seule ligne de code.
- Utilisez le code suivant pour créer un hôte Docker (ici : *docker-sandbox*) dans l'infrastructure Digital Ocean.
- $ docker-machine create --driver digitalocean --digitalocean-access-token xxxxx docker-sandbox
- AWS (Amazon Web Services) permet de créer un Docker-Host (ici : aws-sandbox) avec la commande suivante :
- $ docker-machine create --driver amazonec2 --amazonec2-access-key AKI******* --amazonec2-secret-key 8T93C******* aws-sandbox

Docker-Swarm: Initialisation de Swarm

- Après avoir implémenté le nombre souhaité d'hôtes virtuels pour votre essaim, vous pouvez les gérer via Docker Machine et les regrouper avec Docker Swarm.
- Accédez d'abord au nœud que vous souhaitez utiliser comme Swarm Manager.
- Docker Machine fournit la commande suivante pour établir une **connexion cryptée SSH** avec l'hôte Docker.
- Si la connexion au nœud désiré est établie, utilisez la commande suivante pour initialiser un essaim.
- $ docker swarm init [OPTIONS]
- La commande définit le nœud actuellement sélectionné comme un gestionnaire d'essaim et crée deux tokens aléatoires : un token Manager et un token Worker.

Docker-Swarm: intégrer les hôtes docker dans l'essaim

- **ajouter un nœud Worker :** si vous souhaitez ajouter un **nœud Worker** (nœud de travail) à votre essaim, accédez au nœud correspondant via *docker-machine* et exécutez la commande suivante :
- docker swarm join [OPTIONS] HOST:PORT
- La partie obligatoire de la commande *docker swarm join* est le drapeau *--token*, qui contient le token d'accès au cluster.
- docker swarm join \ --token SWMTKN-1-511cy9taxx5w47n80vopivx6ii6cjpi71vfncqhcfcawxfcb14-6cng4m8lhlrdfuq9jgzznre1p \ 10.0.2.15:2377
- Dans l'exemple actuel, la commande contient le Token Worker généré précédemment ainsi que l'adresse IP sous laquelle le Swarm Manager est disponible.
- Si vous n'avez pas le Token correspondants à portée de main, déterminez le *docker swarm join-token worker.*

Déploiement avec Docker Swarm

Ce déploiement se décline en Trois étapes :

- Tout d'abord la **création du cluster avec les différentes machines** la composant en se basant sur une architecture Slave Master .
- Ensuite la **création d'un réseau commun** que partageront tous les conteneurs.
- Enfin **le déploiement** en tant que tel en se basant sur le fichier YAML utilisé par Docker Compose et la commande **`docker stack deploy`.**
- C'est la **GROSSE nouveauté** de docker Swarm, le déploiement devient extrêmement rapide et facile et surtout offre une couche d'abstraction entre le déploiement et le type d'architecture.
- En effet le même fichier **YAML** sert pour le déploiement sur un serveur seul ou sur un cluster. Les seules différences pour le déploiement sur un cluster étant les options de déploiement avec le choix du nombre de réplicas
- **Docker Swarm** gère nativement le **load balancing** (répartition de la charge sur les différents noeuds) et n'est pas soumis au **Single Point Of Failure** (si le master tombe il sera automatiquement remplacé par un des noeuds qui deviendra le nouveau master