Formation Docker : Concepts et architectures

El Hadji Gaye

Auteur	El Hadji Gaye
Pour	Formations
Date	06/11/2024
Objet	Formation Docker : Concepts et architectures.

I) Vocabulaire

Conteneurisation: En informatique, un conteneur est une structure de données, une classe, ou un type de données abstrait, dont les instances représentent des collections d'autres objets. Autrement dit, les conteneurs sont utilisés pour stocker des objets sous une forme organisée qui suit des règles d'accès spécifiques. On peut implémenter un conteneur de différentes façons, qui conduisent à des complexités en temps et en espace différentes. On choisira donc l'implémentation selon les besoins.

Un conteneur est une enveloppe virtuelle qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, environnement d'exécution, librairies, outils et fichiers. Ils sont assemblés en un ensemble cohérent et prêt à être déployé sur un serveur et son système d'exploitation (OS). Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas de noyau, il s'appuie directement sur le noyau de l'ordinateur sur lequel il est déployé.

Virtualisation : La virtualisation consiste, en informatique, à exécuter sur une machine hôte, dans un environnement isolé, des systèmes d'exploitation — on parle alors de virtualisation système — ou des applications— on parle alors de virtualisation applicative. Ces ordinateurs virtuels sont appelés serveur privé virtuel (Virtual Private Server ou VPS) ou encore environnement virtuel (Virtual Environment ou VE).

II) Introduction

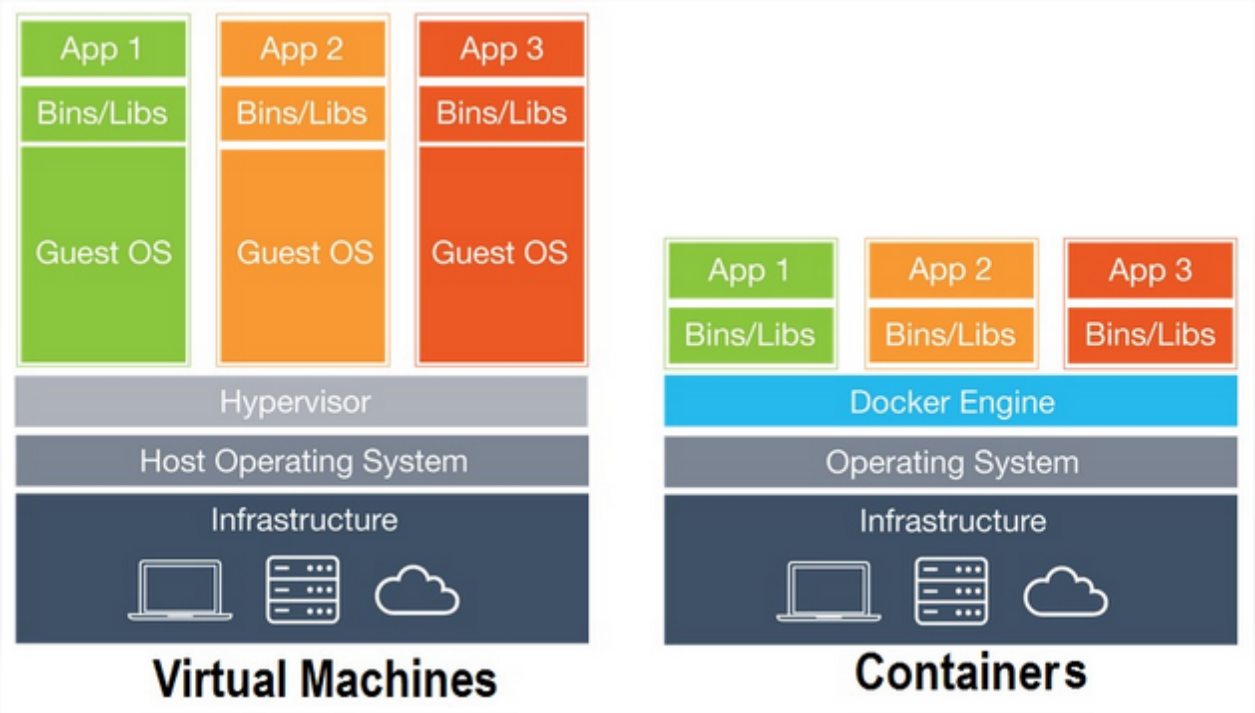
Docker est un logiciel libre permettant de lancer des applications dans des conteneurs logiciels.

Selon la firme de recherche sur l'industrie **451 Research**, « Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de **virtualisation** (voir la section vocabulaire), mais de **conteneurisation** (voir la section vocabulaire), une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc..

Techniquement, Docker étend le format de conteneur Linux standard, LXC, avec une API de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée. Pour arriver à ses fins, Docker utilise entre autres LXC, cgroups et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d'exploitation fournies par la machine hôte.

La technologie de conteneur de Docker peut être utilisée pour étendre des systèmes distribués de façon qu'ils s'exécutent de manière autonome depuis une seule machine physique ou une seule instance par nœud. Cela permet aux nœuds d'être déployés au fur et à mesure que les ressources sont disponibles, offrant un déploiement transparent et similaire aux PaaS pour des systèmes comme Apache Cassandra, Riak ou d'autres systèmes distribués.

III) Conteneur Docker vs Machine virtuelle : Avantages et Inconvénients



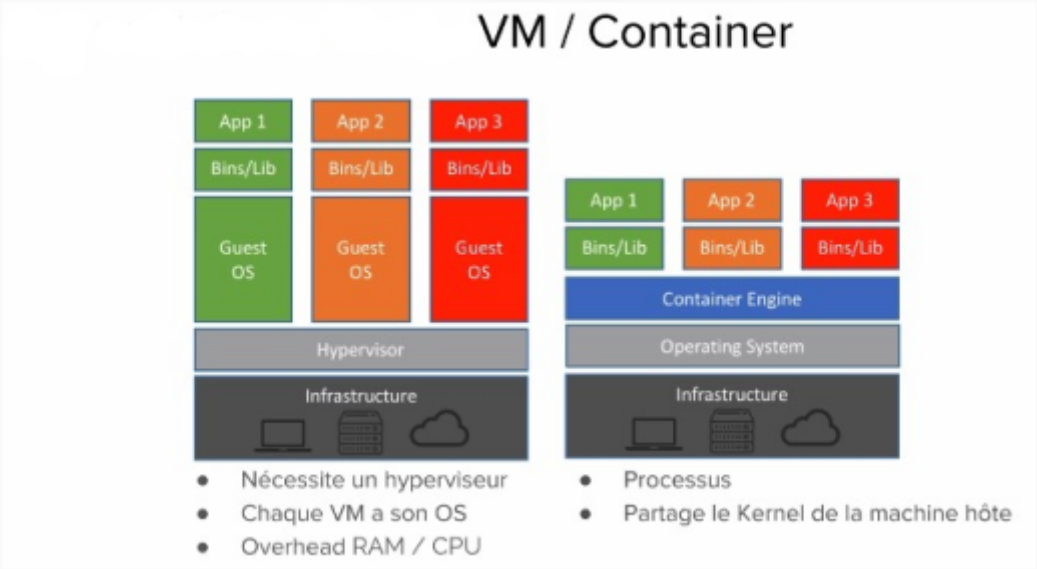
La plateforme Docker présente de nombreux avantages. Elle permet de composer, de créer, de déployer et d'échelonner rapidement des containers sur les hôtes Docker. Elle offre aussi un haut degré de portabilité, ce qui permet aux utilisateurs de s'enregistrer et de partager des containers sur une large variété d'hôtes au sein d'environnements publics et privés.

Par rapport aux machines virtuelles, Docker présente également plusieurs avantages. Elle permet de **développer des applications de façon plus efficiente**, en utilisant moins de ressources, et de déployer ces applications plus rapidement.

Cependant, elle présente aussi plusieurs inconvénients. Il peut être **difficile de gérer de façon efficiente un grand nombre de containers simultanément**. De plus, la sécurité être un problème. Les containers sont isolés, mais partagent le même système d'exploitation. De fait, une attaque ou une faille de sécurité sur l'OS peut compromettre tous les containers. Pour minimiser ce risque, certaines entreprises exécutent leurs containers au sein d'une machine virtuelle.

MAJ : Docker a rencontré une faille de sécurité qui a touché près de 5 % des utilisateurs. Près de 190 000 d'entre eux ont vu les données de leurs containers exposées après un accès non autorisé à base de données Hub. L'organisation a demandé aux entreprises et aux personnes concernées de changer leur mot de passe.

En résumé :


VM / Container
App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Container Engine
Operating System
Infrastructure
Nécessite un hyperviseur
Chaque VM a son OS
Overhead RAM / CPU
Processus
Partage le Kernel de la machine hôte

IV) Fonctionnement de Docker

Docker est une plate-forme ouverte pour le développement, l'expédition et l'exécution d'applications. Docker vous permet de séparer vos applications de votre infrastructure afin de pouvoir livrer rapidement des logiciels. Avec Docker, vous pouvez gérer votre infrastructure de la même manière que vous gérez vos applications. En tirant parti des méthodologies de Docker pour expédier, tester et déployer rapidement le code, vous pouvez réduire considérablement le délai entre l'écriture du code et son exécution en production.

1. *La plateforme Docker*

Docker offre la possibilité de conditionner et d'exécuter une application dans un environnement vaguement isolé appelé conteneur. L'isolement et la sécurité vous permettent d'exécuter plusieurs conteneurs simultanément sur un hôte donné. Les conteneurs sont légers car ils n'ont pas besoin de la charge supplémentaire d'un hyperviseur, mais s'exécutent directement dans le noyau de la machine hôte. Cela signifie que vous pouvez exécuter plus de conteneurs sur une combinaison matérielle donnée que si vous utilisiez des machines virtuelles. Vous pouvez même exécuter des conteneurs Docker dans des machines hôtes qui sont en fait des machines virtuelles!

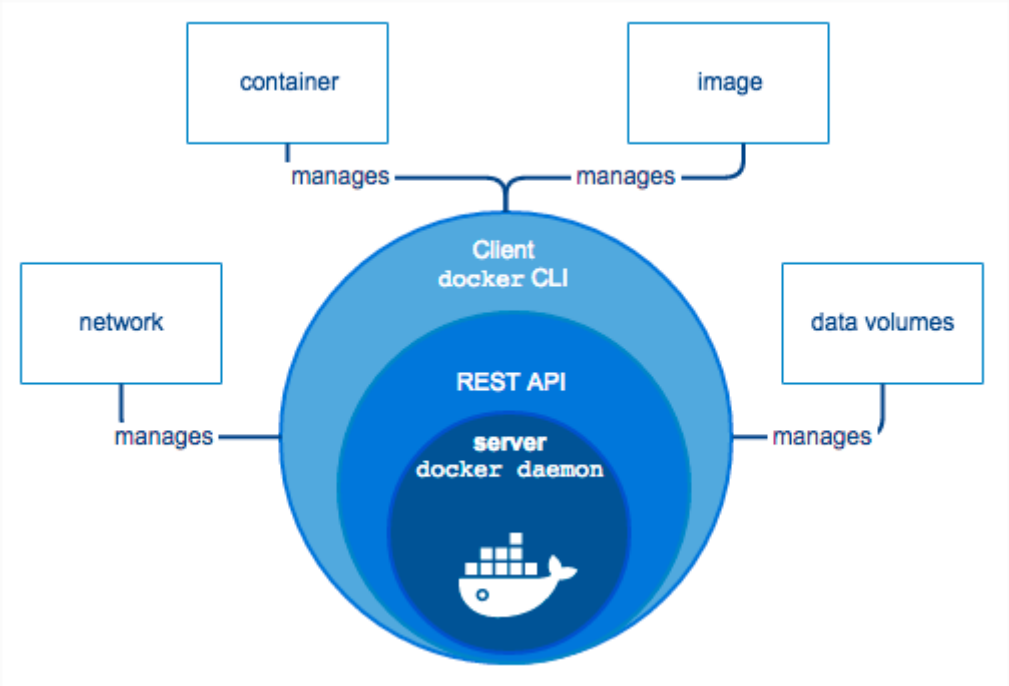
Docker fournit des outils et une plateforme pour gérer le cycle de vie de vos conteneurs:

- Développez votre application et ses composants de support à l'aide de conteneurs.
- Le conteneur devient l'unité de distribution et de test de votre application.
- Lorsque vous êtes prêt, déployez votre application dans votre environnement de production, en tant que conteneur ou service orchestré. Cela fonctionne de la même manière que votre environnement de production soit un centre de données local, un fournisseur de cloud ou un hybride des deux.

2. *Moteur Docker*

Le moteur Docker ou *Docker Engine* est une application client-serveur avec ces composants majeurs:

- Un serveur qui est un type de programme de longue durée appelé processus démon (la **dockerd** commande).
- Une API REST qui spécifie les interfaces que les programmes peuvent utiliser pour parler au démon et lui indiquer quoi faire.
- Un client d'interface de ligne de commande (CLI) (la **docker** commande).



La CLI utilise l'API Docker REST pour contrôler ou interagir avec le démon Docker via des scripts ou des commandes CLI directes. De nombreuses autres applications Docker utilisent l'API et la CLI sous-jacentes.

Le démon crée et gère des *objets* Docker, tels que des images, des conteneurs, des réseaux et des volumes.

Remarque : Docker est concédé sous la licence open source Apache 2.0.

3. *Pourquoi utiliser Docker ?*

Livraison rapide et cohérente de vos applications

Docker rationalise le cycle de vie du développement en permettant aux développeurs de travailler dans des environnements standardisés à l'aide de conteneurs locaux qui fournissent vos applications et services. Les conteneurs sont parfaits pour l'intégration continue et les flux de travail de livraison continue (CI / CD).

Considérez l'exemple de scénario suivant :

- Vos développeurs écrivent du code localement et partagent leur travail avec leurs collègues à l'aide de conteneurs Docker.
- Ils utilisent Docker pour pousser leurs applications dans un environnement de test et exécuter des tests automatisés et manuels.
- Lorsque les développeurs trouvent des bogues, ils peuvent les corriger dans l'environnement de développement et les redéployer dans l'environnement de test à des fins de test et de validation.
- Une fois les tests terminés, fournir le correctif au client est aussi simple que de pousser l'image mise à jour vers l'environnement de production.

Déploiement et évolutivité réactifs

La plate-forme basée sur des conteneurs de Docker permet des charges de travail hautement portables. Les conteneurs Docker peuvent s'exécuter sur l'ordinateur portable local d'un développeur, sur des machines physiques ou virtuelles dans un centre de données, sur des fournisseurs de cloud ou dans un mélange d'environnements.

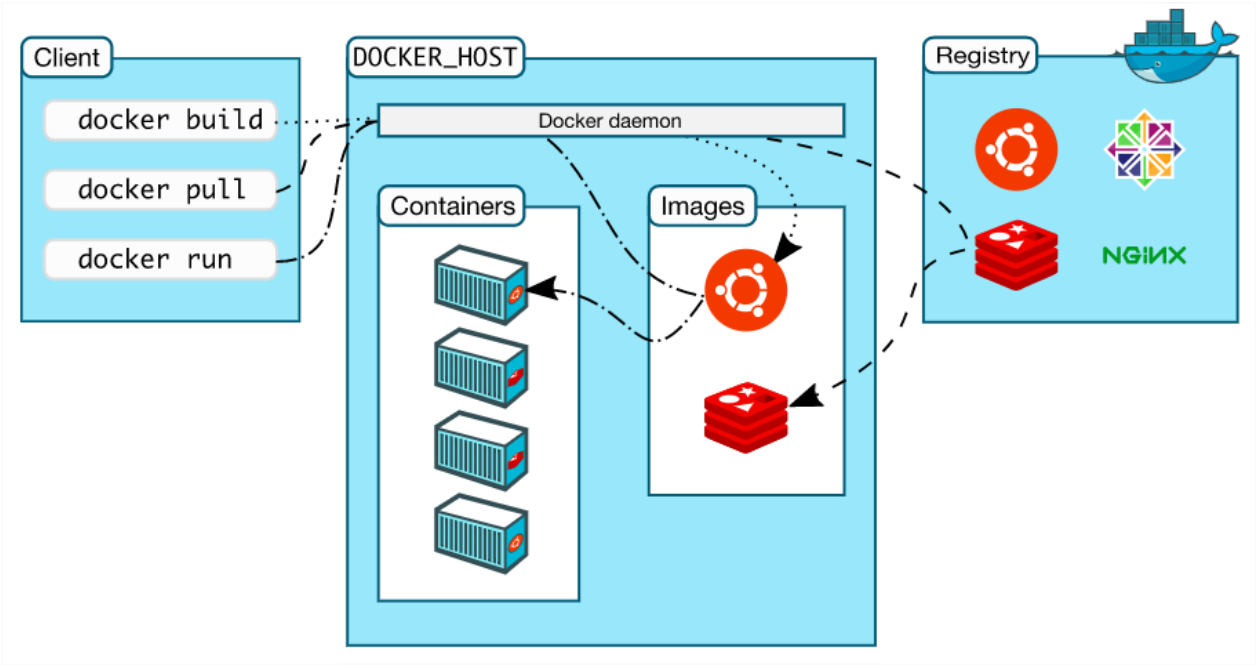
La portabilité et la légèreté de Docker facilitent également la gestion dynamique des charges de travail, en augmentant ou en supprimant les applications et les services selon les besoins de l'entreprise, en temps quasi réel.

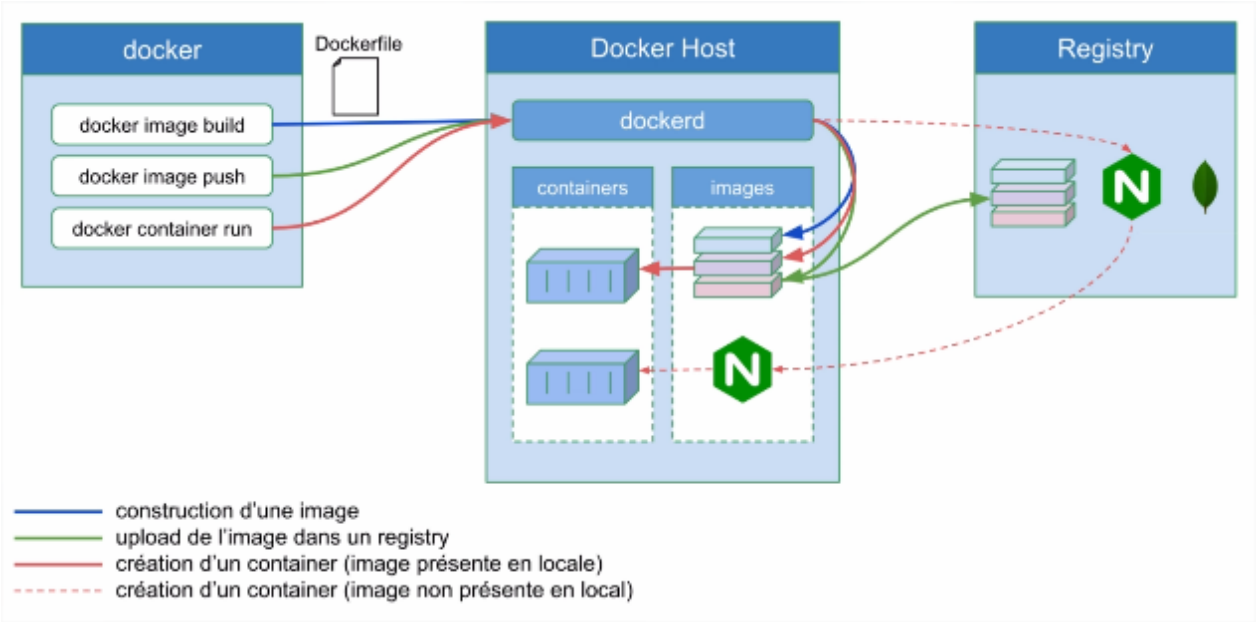
Exécuter plus de charges de travail sur le même matériel

Docker est léger et rapide. Il offre une alternative viable et rentable aux machines virtuelles basées sur un hyperviseur, afin que vous puissiez utiliser davantage votre capacité de calcul pour atteindre vos objectifs commerciaux. Docker est parfait pour les environnements à haute densité et pour les petits et moyens déploiements où vous devez faire plus avec moins de ressources.

4. *L'Architecture de Docker*

Docker utilise une architecture client-serveur. Le client Docker communique avec le démon Docker, qui s'occupe du gros du travail de construction, d'exécution et de distribution de vos conteneurs Docker. Le client et le démon Docker peuvent s'exécuter sur le même système, ou vous pouvez connecter un client Docker à un démon Docker distant. Le client et le démon Docker communiquent à l'aide d'une API REST, via des sockets UNIX ou une interface réseau.





Le démon Docker (deamon)

Le démon Docker (dockerd) écoute les requêtes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux et les volumes. Un démon peut également communiquer avec d'autres démons pour gérer les services Docker.

Le client Docker

Le client Docker (docker) est le principal moyen par lequel de nombreux utilisateurs de Docker interagissent avec Docker. Lorsque vous utilisez des commandes telles que docker run, le client envoie ces commandes à **dockerd**, qui les exécute. Le docker commande utilise l'API Docker. Le client Docker peut communiquer avec plus d'un démon.
Le client communique avec le démon local via le socket Unix via **/var/run/docker.sock**. Ils pourraient communiquer via socket HTTP avec une configuration particulière.

Registres Docker

Un registre Docker stocke les images Docker. Docker Hub est un registre public que tout le monde peut utiliser, et Docker est configuré pour rechercher des images sur Docker Hub par défaut. Vous pouvez même exécuter votre propre registre privé.
Lorsque vous utilisez les commandes docker pullou docker run, les images requises sont extraites de votre registre configuré. Lorsque vous utilisez la docker pushcommande, votre image est poussée vers votre registre configuré.

Objets Docker

Lorsque vous utilisez Docker, vous créez et utilisez des images, des conteneurs, des réseaux, des volumes, des plugins et d'autres objets. Cette section est un bref aperçu de certains de ces objets.

Images

Une image est un modèle en lecture seule avec des instructions pour créer un conteneur Docker. Souvent, une image est basée sur une autre image, avec quelques personnalisations supplémentaires. Par exemple, vous pouvez créer une image basée sur l'Ubuntu image, mais qui installe le serveur Web Apache et votre application, ainsi que les détails de configuration nécessaires à l'exécution de votre application.
Vous pouvez créer vos propres images ou n'utiliser que celles créées par d'autres et publiées dans un registre. Pour créer votre propre image, vous créez un Dockerfile avec une syntaxe simple pour définir les étapes nécessaires pour créer l'image et l'exécuter. Chaque instruction d'un Dockerfile crée un calque dans l'image. Lorsque vous modifiez le Dockerfile et reconstruisez l'image, seuls les calques qui ont été modifiés sont reconstruits. Cela fait partie de ce qui rend les images si légères, petites et rapides par rapport aux autres technologies de virtualisation.

Conteneurs

Un conteneur est une instance exécutable d'une image. Vous pouvez créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de l'API ou de la CLI Docker. Vous pouvez connecter un conteneur à un ou plusieurs réseaux, y attacher du stockage ou même créer une nouvelle image en fonction de son état actuel.
Par défaut, un conteneur est relativement bien isolé des autres conteneurs et de sa machine hôte. Vous pouvez contrôler le degré d'isolement du réseau, du stockage ou des autres sous-systèmes sous-jacents d'un conteneur par rapport aux autres conteneurs ou à la machine hôte.
Un conteneur est défini par son image ainsi que par les options de configuration que vous lui fournissez lorsque vous le créez ou le démarrez. Lorsqu'un conteneur est supprimé, toute modification de son état qui n'est pas stockée dans le stockage persistant disparaît.

Exemple de docker *run* commande

La commande suivante exécute un ubuntuconteneur, s'attache de manière interactive à votre session de ligne de commande locale et s'exécute /bin/bash.

```
$ docker run -i -t ubuntu /bin/bash
```

Lorsque vous exécutez cette commande, ce qui suit se produit (en supposant que vous utilisez la configuration de registre par défaut):

1. Si vous ne disposez pas de l' ubuntu image localement, Docker la extrait de votre registre configuré, comme si vous l'aviez exécutée docker pull ubuntu manuellement.
2. Docker crée un nouveau conteneur, comme si vous aviez exécuté une docker container create commande manuellement.
3. Docker alloue un système de fichiers en lecture-écriture au conteneur, comme couche finale. Cela permet à un conteneur en cours d'exécution de créer ou de modifier des fichiers et des répertoires dans son système de fichiers local.
4. Docker crée une interface réseau pour connecter le conteneur au réseau par défaut, car vous n'avez spécifié aucune option de mise en réseau. Cela inclut l'attribution d'une adresse IP au conteneur. Par défaut, les conteneurs peuvent se connecter à des réseaux externes à l'aide de la connexion réseau de la machine hôte.
5. Docker démarre le conteneur et s'exécute /bin/bash. Étant donné que le conteneur fonctionne de manière interactive et est attaché à votre terminal (en raison des indicateurs -iet -t), vous pouvez fournir une entrée à l'aide de votre clavier pendant que la sortie est enregistrée sur votre terminal.
6. Lorsque vous tapez exit pour mettre fin à la /bin/bash commande, le conteneur s'arrête mais n'est pas supprimé. Vous pouvez le redémarrer ou le supprimer.

Prestation de service

Les services vous permettent de mettre à l'échelle des conteneurs sur plusieurs démons Docker, qui fonctionnent tous ensemble comme un essaim avec plusieurs gestionnaires et travailleurs. Chaque membre d'un essaim est un démon Docker, et tous les démons communiquent à l'aide de l'API Docker. Un service vous permet de définir l'état souhaité, tel que le nombre de répliques du service qui doivent être disponibles à un moment donné. Par défaut, le service est équilibré en charge sur tous les nœuds de travail. Pour le consommateur, le service Docker semble être une seule application. Docker Engine prend en charge le mode Swarm dans Docker 1.12 et supérieur.

5. *La technologie sous-jacente de Docker*

Docker est écrit en Go et tire parti de plusieurs fonctionnalités du noyau Linux pour fournir ses fonctionnalités.

Espaces de noms

Docker utilise une technologie appelée namespaces pour fournir l'espace de travail isolé appelé le *conteneur*. Lorsque vous exécutez un conteneur, Docker crée un ensemble d'espaces de *noms* pour ce conteneur.

Ces espaces de noms fournissent une couche d'isolation. Chaque aspect d'un conteneur s'exécute dans un espace de noms distinct et son accès est limité à cet espace de noms.

Docker Engine utilise des espaces de noms tels que les suivants sous Linux:

- **pid : Isolation de** processus (PID: ID de processus).
- **net :** Gestion des interfaces réseau (NET: Networking).
- **ipc :** Gérer l'accès aux ressources IPC (IPC: InterProcess Communication).
- **mnt :** Gestion des points de montage du système de fichiers (MNT: Mount).
- **uts :** isoler les identificateurs de noyau et de version. (UTS: système de partage de temps Unix).

Groupes de contrôle

Docker Engine sur Linux repose également sur une autre technologie appelée *groupes de contrôle* (cgroups). Un groupe de contrôle limite une application à un ensemble spécifique de ressources. Les groupes de contrôle permettent à Docker Engine de partager les ressources matérielles disponibles avec des conteneurs et d'appliquer éventuellement des limites et des contraintes. Par exemple, vous pouvez limiter la mémoire disponible à un conteneur spécifique.

Union des systèmes de fichiers

Les systèmes de fichiers Union, ou UnionFS, sont des systèmes de fichiers qui fonctionnent en créant des couches, ce qui les rend très légers et rapides. Docker Engine utilise UnionFS pour fournir les blocs de construction des conteneurs. Docker Engine peut utiliser plusieurs variantes d'UnionFS, notamment AUFS, btrfs, vfs et DeviceMapper.

Format du conteneur

Docker Engine combine les espaces de noms, les groupes de contrôle et UnionFS dans un wrapper appelé format de conteneur. Le format de conteneur par défaut est lib container. À l'avenir, Docker pourra prendre en charge d'autres formats de conteneurs en s'intégrant à des technologies telles que BSD Jails ou Solaris Zones.

V) Les versions de Docker

Il existe deux version de docker :

- Docker Community Edition (CE)
    - Open source
    - Release stable tous les 6 mois
    - Realease test avant chaque release stable

- Docker Enterprise Edition (EE)
    - 3 tiers (basic, standard, avancé)
    - Une release tous les trimestres avec A an de support
    - Plateforme Caas (Container as a Service)
    - Technologie certifiée : infrastructure, Plugins, Contaners, …

VI) Docker Swarm

Les versions actuelles de Docker incluent le mode Swarm pour gérer nativement un cluster de moteurs Docker appelé Swarm. Utilisez la CLI Docker pour créer un essaim, déployer des services d'application sur un essaim et gérer le comportement de Swarm.

Les fonctionnalités de Docker Swarm

- **Gestion de cluster intégrée à Docker Engine:** utilisez l'interface de ligne de commande Docker Engine pour créer un essaim de moteurs Docker où vous pouvez déployer des services d'application. Vous n'avez pas besoin de logiciel d'orchestration supplémentaire pour créer ou gérer un essaim.
- **Conception décentralisée: au** lieu de gérer la différenciation entre les rôles de nœud au moment du déploiement, le moteur Docker gère toute spécialisation au moment de l'exécution. Vous pouvez déployer les deux types de nœuds, de gestionnaires et de nœuds de calcul, à l'aide du moteur Docker. Cela signifie que vous pouvez créer un essaim entier à partir d'une seule image disque.
- **Modèle de service déclaratif:** Docker Engine utilise une approche déclarative pour vous permettre de définir l'état souhaité des différents services de votre pile d'applications. Par exemple, vous pouvez décrire une application composée d'un service Web frontal avec des services de mise en file d'attente de messages et un backend de base de données.
- **Mise à l'échelle:** pour chaque service, vous pouvez déclarer le nombre de tâches que vous souhaitez exécuter. Lorsque vous augmentez ou diminuez l'échelle, le gestionnaire d'essaim s'adapte automatiquement en ajoutant ou en supprimant des tâches pour maintenir l'état souhaité.
- **Réconciliation de l'état souhaité:** le nœud du gestionnaire de swarm surveille en permanence l'état du cluster et réconcilie les différences entre l'état réel et l'état souhaité exprimé. Par exemple, si vous configurez un service pour exécuter 10 répliques d'un conteneur et qu'une machine de travail hébergeant deux de ces répliques se bloque, le gestionnaire crée deux nouvelles répliques pour remplacer les répliques qui se sont écrasées. Le gestionnaire d'essaim affecte les nouveaux réplicas aux nœuds de calcul en cours d'exécution et disponibles.
- Réseau **multi-hôte:** vous pouvez spécifier un réseau de superposition pour vos services. Le gestionnaire d'essaim attribue automatiquement des adresses aux conteneurs sur le réseau de superposition lorsqu'il initialise ou met à jour l'application.
- **Découverte de service: les** nœuds du gestionnaire Swarm attribuent à chaque service de l'essaim un nom DNS unique et équilibre la charge des conteneurs en cours

d'exécution. Vous pouvez interroger chaque conteneur en cours d'exécution dans l'essaim via un serveur DNS intégré dans l'essaim.

- **Équilibrage de charge:** vous pouvez exposer les ports des services à un équilibreur de charge externe. En interne, l'essaim vous permet de spécifier comment distribuer les conteneurs de services entre les nœuds.
- **Sécurisé par défaut:** chaque nœud de l'essaim applique l'authentification mutuelle et le cryptage TLS pour sécuriser les communications entre lui-même et tous les autres nœuds. Vous avez la possibilité d'utiliser des certificats racine auto-signés ou des certificats d'une autorité de certification racine personnalisée.
- **Mises à jour** progressives **:** au moment du déploiement, vous pouvez appliquer des mises à jour de service aux nœuds de manière incrémentielle. Le gestionnaire de swarm vous permet de contrôler le délai entre le déploiement du service sur différents ensembles de nœuds. En cas de problème, vous pouvez revenir à une version précédente du service.

VII) Présentation des offres concurrentes de Docker

Bien que les versions actuelles de Docker incluent le mode Swarm pour gérer nativement un cluster, certains architectes trouvent beaucoup de limites à Docker swarm donc préfére se tourner vers des solutions comme **Kubernetes** et **Apache Mesos**.

1. Kubernetes

Kubernetes (communément appelé « **K8s2** ») est un système open source qui vise à fournir une « plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs. Il fonctionne avec toute une série de technologies de conteneurisation, et est souvent utilisé avec Docker. Il a été conçu à l'origine par Google, puis offert à la Cloud Native Computing Foundation.

a. Historique

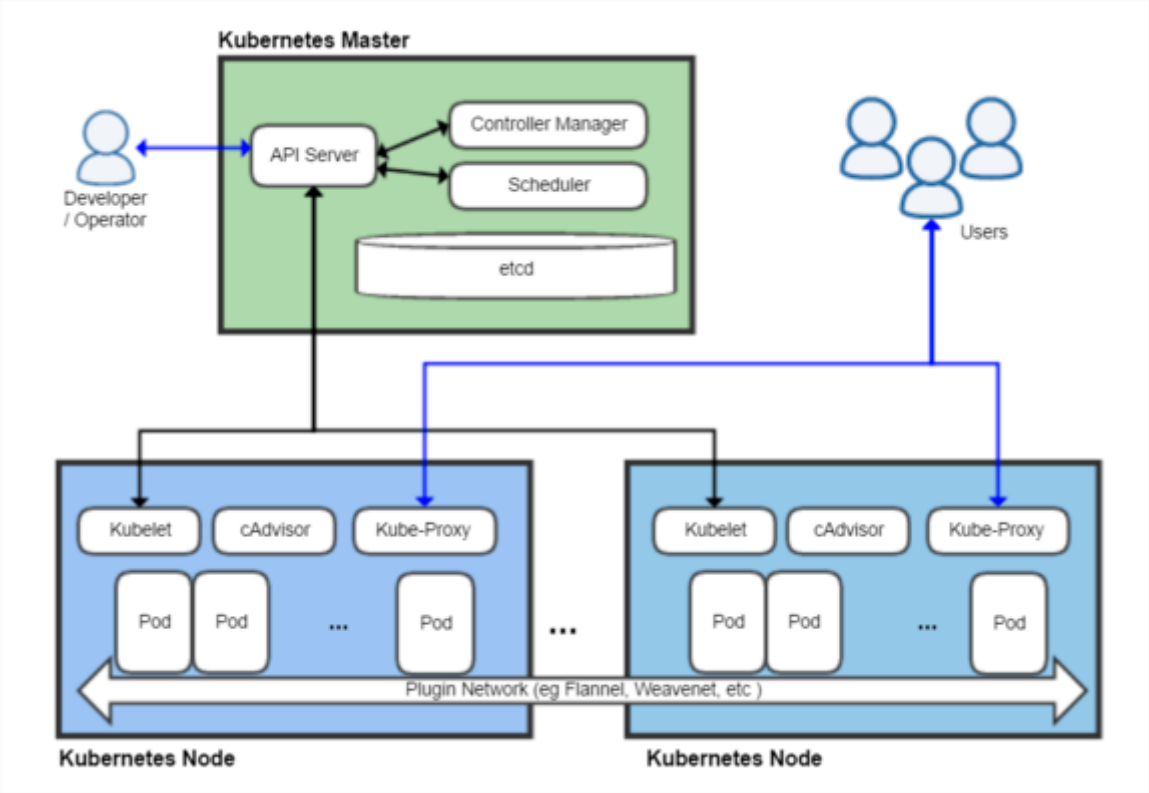
Kubernetes (grec pour « timonier » ou « pilote ») créé par Joe Beda, Brendan Burns et Craig McLuckie, rapidement rejoints par d'autres ingénieurs de Google comme Brian Grant et Tim Hockin, est annoncé pour la première fois par Google à la mi-20145. Son développement et son architecture ont été fortement influencés par le système Borg (en) de Google. D'ailleurs, la plupart des contributeurs principaux sont issus du projet Borg. Le nom original de Kubernetes en interne fut Project Seven, en référence au personnage de Star Trek qui est un Borg devenu amical. Les sept rayons de la barre du logo de Kubernetes sont un clin d'œil au nom original.

Kubernetes version 1.0 est sorti le 21 juillet 2015. Avec la sortie de la première version de Kubernetes, Google fit un partenariat avec la Fondation Linux pour créer la Cloud Native Computing Foundation (CNCF) et offrit Kubernetes comme technologie de départ.

Kubernetes est également utilisé par RedHat pour son produit OpenShift, par CoreOS dans son produit Tectonic, et par Rancher Labs pour sa plateforme de gestion de conteneurs Rancher.

b. Architecture

Mon texte Kubernetes suit l'architecture maitre/esclave. Les composants de Kubernetes peuvent être divisés en ceux qui gèrent un nœud individuel et ceux qui font partie du plan de contrôle.



Plan de contrôle Kubernetes

Le maître Kubernetes est l'unité de contrôle principale qui gère la charge de travail et dirige les communications dans le système. Le plan de contrôle de Kubernetes consiste en plusieurs composants, chacun ayant son propre processus, qui peuvent s'exécuter sur un seul node maître ou sur plusieurs maîtres permettant de créer des clusters haute disponibilité. Les différents composants du plan de contrôle de Kubernetes sont décrits ci-dessous:

etcd

etcd est une unité de stockage distribuée persistante et légère de données clé-valeur développée par CoreOS, qui permet de stocker de manière fiable les données de configuration du cluster, représentant l'état du cluster à n'importe quel instant. D'autres composants scrutent les changements dans ce stockage pour aller eux-mêmes vers l'état désiré.

Serveur d'API

Le serveur d'API est un élément clé et sert l'API Kubernetes grâce à JSON via HTTP. Il fournit l'interface interne et externe de Kubernetes. Le serveur d'API gère et valide des requêtes REST et met à jour l'état des objets de l'API dans etcd, permettant ainsi aux clients de configurer la charge de travail et les containers sur les nœuds de travail.

L'ordonnanceur

L'ordonnanceur est un composant additionnel permettant de sélectionner quel node devrait faire tourner un pod non ordonnancé en se basant sur la disponibilité des ressources. L'ordonnanceur gère l'utilisation des ressources sur chaque node afin de s'assurer que la charge de travail n'est pas en excès par rapport aux ressources disponibles. Pour accomplir cet objectif, l'ordonnanceur doit connaître les ressources disponibles et celles actuellement assignées sur les serveurs.

Controller Manager

Le gestionnaire de contrôle (controller manager) est le processus dans lequel s'exécutent les contrôleurs principaux de Kubernetes tels que DaemonSet Controller et le Replication Controller. Les contrôleurs communiquent avec le serveur d'API pour créer, mettre à jour et effacer les ressources qu'ils gèrent (pods, service endpoints, etc.).

Node Kubernetes

Le Node aussi appelé Worker ou Minion est une machine unique (ou une machine virtuelle) où des conteneurs (charges de travail) sont déployés. Chaque node du cluster doit exécuter le programme de conteneurisation (par exemple Docker), ainsi que les composants mentionnés ci-dessous, pour communiquer avec le maître afin de configurer la partie réseau de ces conteneurs.

Kubelet

Kubelet est responsable de l'état d'exécution de chaque nœud (c'est-à-dire, d'assurer que tous les conteneurs sur un nœud sont en bonne santé). Il prend en charge le démarrage, l'arrêt, et la maintenance des conteneurs d'applications (organisés en pods) dirigé par le plan de contrôle.

Kubelet surveille l'état d'un pod et s'il n'est pas dans l'état voulu, le pod sera redéployé sur le même node. Le statut du node est relayé à intervalle de quelques secondes via messages d'état vers le maître. Dès que le maître détecte un défaut sur un node, le Replication Controller voit ce changement d'état et lance les pods sur d'autres hôtes en bonne santé.

Kube-Proxy

Le kube-proxy est l'implémentation d'un proxy réseau et d'un répartiteur de charge, il gère le service d'abstraction ainsi que d'autres opérations réseaux. Il est responsable d'effectuer le routage du trafic vers le conteneur approprié en se basant sur l'adresse IP et le numéro de port de la requête entrante.

cAdvisor

cAdvisor est un agent qui surveille et récupère les données de consommation des ressources et des performances comme le processeur, la mémoire, ainsi que l'utilisation disque et réseau des conteneurs de chaque node.

2. *Apache Mesos*

Apache Mesos est un projet open-source gestionnaire de cluster. Il est développé par l'université de Berkeley. Il est construit en utilisant les mêmes principes que le noyau Linux, uniquement à un niveau d'abstraction différent. Le noyau Mesos s'exécute sur chaque machine et fournit des applications (par exemple, Hadoop, Spark, Kafka, Elasticsearch) avec des API pour la gestion des ressources et la planification dans tout le centre de données et les environnements cloud.

Apache Mesos est ce qu'on appelle un système de gestion de cluster. C'est plus précisément une solution pensée pour mettre en place et optimiser des systèmes distribués. Mesos va permettre de gérer et partager de manière fine, souple et dynamique des ressources entre différents clusters, pour diverses applications.

Il s'agit d'un système open source distribué sous licence Apache. Mesos est même devenu un projet de première importance pour la Fondation Apache (c'est un "Top-Level Project") depuis juillet 2013. La solution Mesos est déjà en production dans de nombreuses entreprises de premier plan, comme Hubspot, Vimeo, Twitter, Airbnb ou eBay.

A quoi sert cette solution ?

En permettant d'isoler et partager des ressources, **Apache Mesos** convient particulièrement bien pour exécuter des applications et des systèmes distribués. **Apache Mesos** est donc notamment utilisé pour faire tourner Hadoop, Spark, Storm, Kafka, ou Elastic Search. Autant de systèmes associés au Big Data, et qui ont clairement le vent en poupe aujourd'hui dans l'informatique.

Concrètement, Mesos permet d'exécuter plusieurs systèmes distribués sur le même cluster : par exemple plutôt que de faire tourner un cluster pour Hadoop, et un autre pour Storm, un même cluster pourrait faire tourner les deux grâce au mécanisme de gestion de ressources de Mesos.

Mesos ouvre donc globalement la voie à une meilleure optimisation de la gestion des ressources. Ce qui ne laisse évidemment pas indifférents les responsables informatiques, surtout ceux qui se sont lancés dans les systèmes distribués. Il semble d'ailleurs que Google ait mis au point un système très comparable, d'abord appelé "Borg" puis "Omega", et qu'il l'utilise massivement en interne.

Architecture de Mesos

Mesos est parfois comparé à la virtualisation, mais en plus efficace... Car, un peu comme la virtualisation, Mesos facilite "l'abstraction du CPU, de la mémoire, du stockage et autres ressources de calcul des machines, qu'elles soient physiques ou virtuelles". Il permet ainsi de bâtir "facilement et efficacement des systèmes distribués et élastiques", comme l'explique le site officiel du projet.

Dans ce contexte, Mesos peut permettre de mettre en place des planificateurs ("schedulers") de ressources personnalisées, offrant une gestion fine de la planification multi-ressources (mémoire et CPU). Les containers Docker sont aussi supportés. Une interface web permet de voir l'état du cluster. Testée en production, la capacité de dimensionnement de Mesos pourrait prendre en charge des milliers de nœuds.

Qui utilise cette solution ?

Deux géants de l'informatique ont largement partagé leur expérience d'utilisation de Mesos.

- Airbnb en est par exemple un gros utilisateur, et a largement communiqué sur son expérience en la matière. Une présentation réalisée par l'ingénieur Brenden Matthews (en juillet 2013) expliquait en détail comment Airbnb exécutait notamment Hadoop et Storm sur Mesos.

- Twitter est aussi un utilisateur bien connu de Mesos, et pour cause : le co-créateur de Mesos, Benjamin Hindman, a été salarié par le réseau social, qui l'a tout simplement mis au cœur de son architecture. Twitter ne s'en cache pas, plusieurs de ses services clés tournent sur Mesos, comme ses services analytics, ou publicitaires. "Nous nous en servons pour bâtir tous nos nouveaux services. En utilisant Mesos, ces services peuvent évoluer et tirer parti efficacement d'un ensemble partagé de serveurs. En outre, Mesos a changé la façon dont les développeurs pensent les lancements de nouveaux services. Au lieu de raisonner en machines statiques, les ingénieurs pensent en ressources : CPU, mémoire ou disque. L'utilisation de cette technologie chez nous s'est traduite aussi par une réduction du temps séparant le prototype du lancement", expliquait Dave Lester, "avocat de l'open source" chez Twitter.