Virtualisation et Cloud

Chapitre 2 : La Conteneurisation

Octobre 2022

Enseignante : Maymouna BEN SAID

Agenda

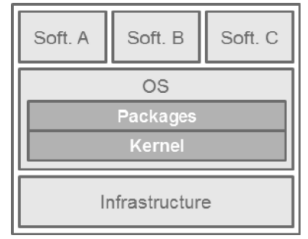
Agenda

1. Virtualisation versus conteneurisation (Cont.)

Plusieurs logiciels sur le même hôte : Problème [1]?

- Interaction entre les différents logiciels est possible :
  - accès au même chemin sur le système de fichiers
  - accès aux mêmes ressources matérielles (ports, cpu, etc)
- La mise à jour du système d'exploitation / (d'un logiciel) va nécessairement impacter tous les logiciels qui tournent dessus
- Pour des logiciels fournis par des éditeurs différents, la cohabitation est très souvent problématique

➡**Solution : la virtualisation**



Installation native de trois logiciels sur un même hôte [1]

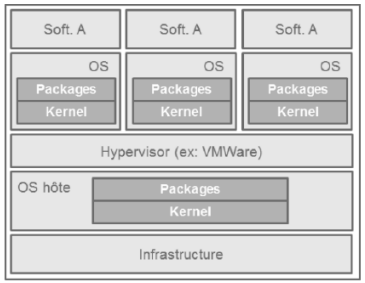
[1] : Docker—Pratique des architectures à base de conteneurs, Dunod

1. Virtualisation versus conteneurisation (Cont.)

Virtualisation basée Hyperviseur

- Émule un ordinateur complet : possibilité d'émuler d'autres types de hardware, architectures CPU et OS, etc.
- Isolation des applications via les VMs

- Niveau d'isolation élevé : problème résolu
- Nouveaux problèmes :
  - Un nouveau OS est nécessaire
  - Stockage VM important même avec OS minimal
  - Trop d'éléments dans une VM



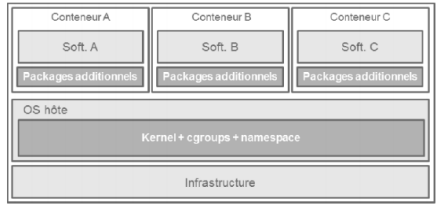
Virtualisation matérielle : trois VMs sur le même hôte [1]

[1] : Docker—Pratique des architectures à base de conteneurs, Dunod

1. Virtualisation versus conteneurisation (Cont.)

Virtualisation basée Conteneur

- Couche de virtualisation sans hyperviseur
- Permet le fonctionnement de plusieurs applications dans un environnement isolé ou limité
- Les applications conteneurisées communiquent directement avec le système hôte et sont capables ainsi de s'exécuter sur le système d'exploitation hôte et sur l'architecture du CPU



Architecture à conteneurs : trois conteneurs sur le même hôte [1]

- Démarrage facile au bout de quelques secondes : ne nécessite pas une émulation du hardware et le démarrage d'un système d'exploitation
- Taille des applications conteneurisées (images) très petite vis à vis celle d'une image d'une VM
- La portabilité des conteneurs est une approche intéressante dans le développement des applications
- Gestion facile des projets dans une machine : pas de conflit de dépendance

[1] : Docker—Pratique des architectures à base de conteneurs, Dunod

1. Virtualisation versus conteneurisation

Virtualisation basée Conteneur

- Solution de compromis

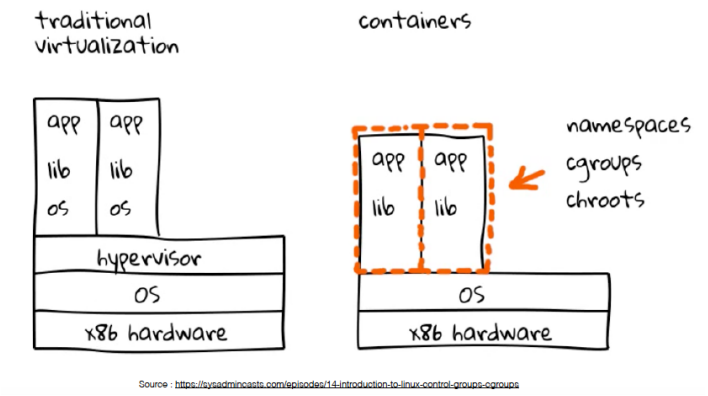
- Une architecture qui permet au développeur d'embarquer l'ensemble des dépendances logicielles et de les isoler
- Stockage très léger : le conteneur s'appuie sur le noyau du système hôte
- Le nombre de conteneurs qu'un système hôte peut encapsuler est nettement très supérieur que son équivalent en VMs
- Coté Cloud : offre des performances meilleures
- Coté entreprise : remplacer les solutions couteuses (hyperviseurs) par des conteneurs

[1] : Docker—Pratique des architectures à base de conteneurs, Dunod

Agenda

Définition

- La conteneurisation utilise les propriétés existantes (fonctionnalités) du noyau du système d'exploitation pour créer un environnement isolé pour les processus.

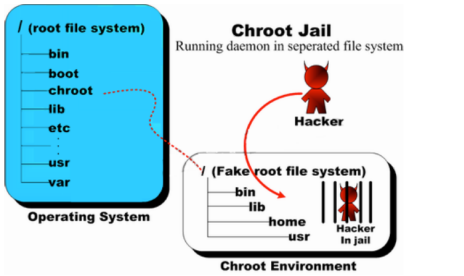


Source : https://sysadmincasts.com/episodes/14-introduction-to-linux-control-groups-cgroups

La commande "jail" : chroot

Définition : chroot (change root)(1979)— Commande qui permet le changement du répertoire racine d'un processus et de ses processus fils (sous-processus)

- chroot fournit un espace disque isolé pour chaque processus
- chroot peut prévenir les logiciels malveillants de l'accès à des fichiers à l'extérieur de l'espace crée (faux répertoire /)
- chroot n'offre aucun autre mécanisme d'isolation processus à part le changement du répertoire racine (/) du processus

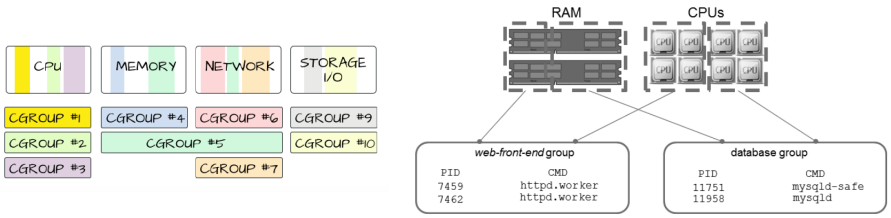


Source : https://sysopsio.wordpress.com/2016/09/09/jails-in-linux/

Contrôle de ressources : cgroups (Control Groups)

Définition : une fonctionnalité du noyau Linux pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.)—wikipedia.fr

- cgroups (2006) fournit un mécanisme pour rassembler des processus dans un groupe et limiter leur accès aux ressources système.
- L'objectif est de contrôler la consommation des ressources par processus.
- Les ressources sont : Temps CPU, Mémoire système Disque dur (stockage), Bande passante (réseau).

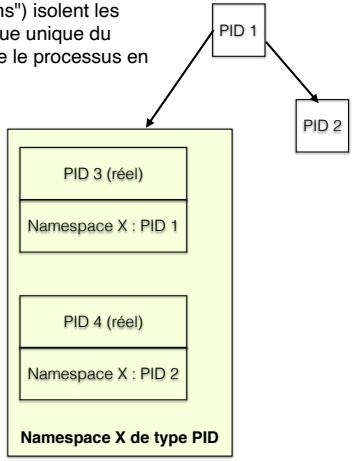


Répartition des ressources

Les espaces de nom : Namespaces

Définition : Namespace — Les namespaces (ou "espaces de noms") isolent les ressources partagées. Ils donnent à chaque processus sa propre vue unique du système, limitant ainsi leur accès aux ressources système sans que le processus en cours ne soit au courant des limitations.— https://devopssec.fr

- Permettent de disposer d'un environnement distinct pour : PID, réseau, points de montage, utilisateurs et groupes, etc.
  - Par défaut, chaque système Linux possède un unique espace de nom (Namespace)
  - Tous les ressources système (système de fichiers, processus PID, IDs des utilisateurs, interfaces réseau, etc.) appartiennent à cet espace de nom.
  - On peut créer des espaces de nom additionnels pour organiser les ressources à l'intérieur.
- **Une utilisation importante des espaces de nom, "namespaces", est l'implémentation d'une virtualisation légère (conteneurisation)**



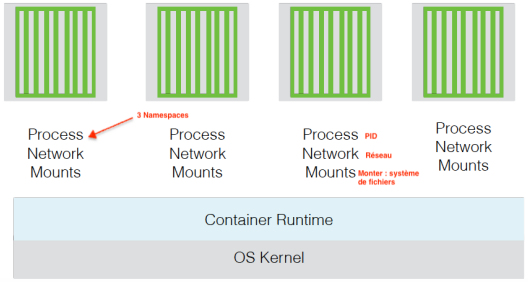
2. Les Fondations Linux

Qu'est ce qu'un Conteneur ?

- Un conteneur est un système de fichiers (RootFS) sur le quel s'exécutent des processus en respectant les règles suivantes :
  - L'isolation : les conteneurs ne se voient pas les uns des autre
  - La limite en terme de ressources

Container Runtime : environnement d'exécution des conteneurs

- Les différentes initialisations au système d'exploitation qui permettent d'isoler un environnement seront prises en charge et abstraites par l'environnement d'exécution des conteneurs (container runtime), qui devra être installé sur la machine hôte : **c'est le seul pré-requis à l'exécution de conteneurs.**



Agenda

3. Les Les outils de conteneurisation (Cont.)

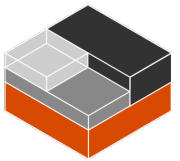
Historique

- 1979 : Unix V7 chroot
- 2000 : FreeBSD Jails
- 2001 : LinuxVServer
- 2004 : Solaris Containers
- 2005 : OpenVZ
- 2006 : cgroups
- 2008 : namespaces
- 2008 : LXC
- 2011 : Warden (Cloudfoundry)
- 2013 : Lmctfy (google) : https://github.com/google/lmctfy
- 2013 : Docker

3. Les Les outils de conteneurisation (Cont.)

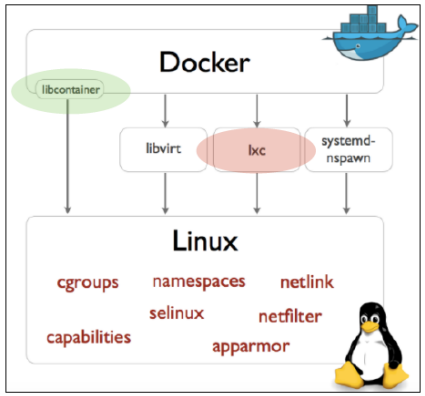
LXC

- LXC est un projet en open source soutenu financièrement par Canonical
- Site web : http://linuxcontainers.org/
- Plus de 60 contributeurs
- Mainteneurs : Serge Hallyn(Ubuntu), Stéphane Graber(Ubuntu)
- Intégré au noyau 2.6.24 : création des instances isolées baptisées "Conteneur Linux" ou LXC (LinuX Containers)
- LXC utilise la notion de contrôle de ressource "cgroups" et propose également une isolation au niveau des espaces de nom (Namespaces) et les modules 'Application Armor' (AppArmor) et Security-Enhanced Linux (SELinux).
- LXC est une technologie de conteneurisation qui offre des conteneurs Linux légers



Docker

- Docker est un projet open source écrit en GO et hébergé sur GitHub (https://github.com/docker).
- Le projet a été initialement développé par la startup DotCloud en 2013
- Docker a utilisé LXC à ses débuts et qui a été remplacé par la suite par sa propre bibliothèque, libcontainer



Différences entre Docker & LXC ?

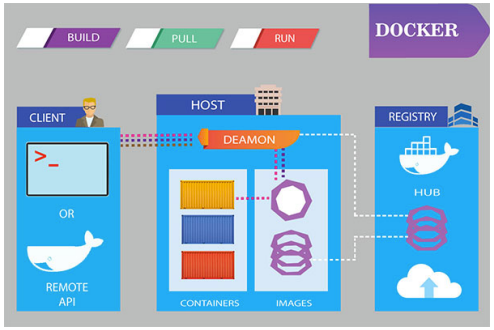
- Docker restreint l'utilisation du conteneur à une seul service ("one service per container" philosophy) : il faut construire X conteneurs pour une application qui consiste de X services.
  - **On peut utiliser un seul conteneur Docker pour lancer plusieurs services via un script mais il est généralement recommandé d'utiliser un service par conteneur.**
- LXC peut contenir plusieurs processus à l'intérieur d'un conteneur

- Docker construit sur LXC pour y ajouter des services de gestion et de déploiement d'images.
  - **Portabilité** des conteneurs entre deux machines
  - **Automatisation** : création des conteneurs à partir des fichiers Dockerfile
  - Docker est un outil de création et distribution d'application portable via le concept d'image

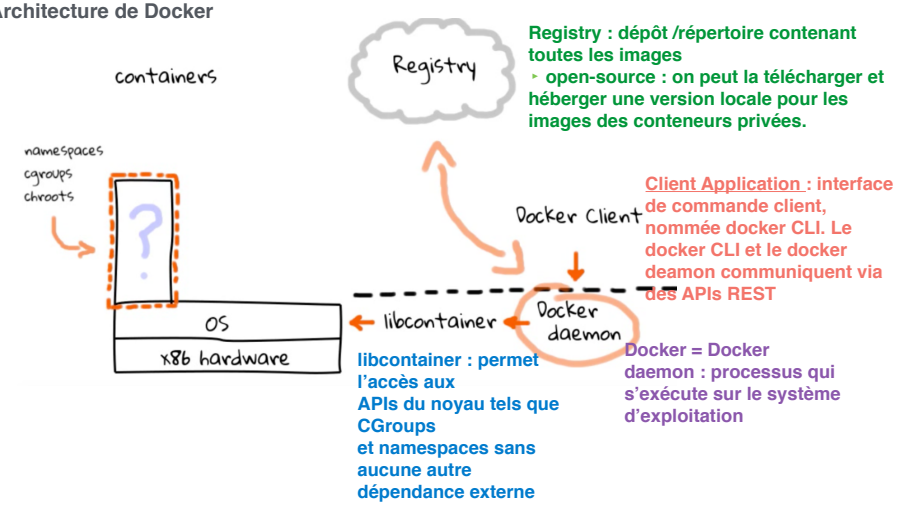
- **l'objectif principal de Docker est de faire le paquetage, la conteneurisation, la distribution et l'exécution des applications autant de fois, n'importe où et à tout moment**

Architecture de Docker

- Docker utilise une architecture client-serveur :
  - Docker Client demande à Docker daemon de créer un conteneur
  - Docker Daemon fait la construction, l'exécution, et la distribution du conteneur et ce en communiquant avec le noyau via libcontainer
- Docker Daemon et libcontainer existent sur la même machine ou système
- Docker client peut fonctionner sur une machine distante



Architecture de Docker



Registry : dépôt /répertoire contenant toutes les images
- **open-source : on peut la télécharger et héberger une version locale pour les images des conteneurs privées.**

Client Application : interface de commande client, nommée docker CLI. Le docker CLI et le docker deamon communiquent via des APIs REST

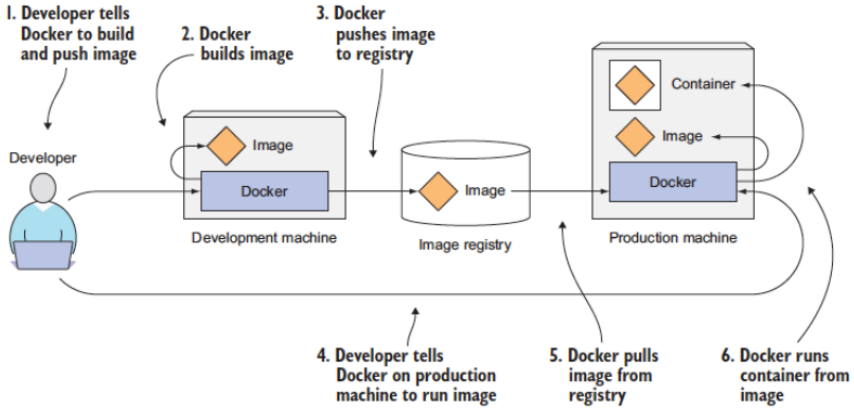
libcontainer : permet l'accès aux APIs du noyau tels que CGroups et namespaces sans aucune autre dépendance externe

Docker = Docker daemon : processus qui s'exécute sur le système d'exploitation

Workflow Docker

1. **Docker Client** demande à **Docker daemon** de créer un conteneur en utilisant une image spécifique
2. **Docker daemon** télécharge l'image depuis le dépôt **Registry**
3. **Docker daemon** communique avec le noyau Linux via **libcontainer** pour créer le conteneur
   - Pour construire un espace isolé pour l'application, le conteneur est crée avec les fonctionnalités noyau : namespaces, cgroups

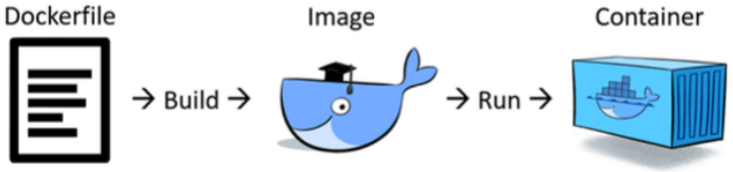
Workflow Docker



Source : Marko Luksa. Kubernetes in Action. Manning Publications Co. 2018

Les apports de Docker—Notion d'Image

- **Image** : c'est le template / modèle, les données de l'application prêtes à l'emploi
  - Archive qui peut être **échangée** entre plusieurs hôtes et **réutilisée** tant de fois
- **Conteneur** : instance qui fonctionne / peut être démarrée



https://www.saagie.com/blog/your-first-steps-into-docker/

Les apports de Docker—Hub et Images

- Les images sont disponibles dans des dépôts : https://hub.docker.com https://store.docker.com
- Ce dépôt contient les images déposées par les autres
- Choix multiple : possibilité de télécharger des versions
- Le dépôt git d'une image contient les informations sur les options possibles et les explications
- Contraintes de sécurité liées à l'utilisation des images publiques ?

Gestion d'une image— docker pull / docker push

1. docker pull : télécharger une image

```
docker pull [OPTIONS] ImageName:TAG
```

- Télécharger / extraire une image depuis le dépôt (registry) Docker Hub
- TAG est la version (par defaut latest)
- Exemple : docker pull debian:8.5

2. docker push : envoyer (uploader) l'image sur un serveur

```
docker push ImageName:TAG
```


Gestion d'une image— Construction d'une image

1. docker commit Construction d'une image à partir d'un conteneur

```
docker commit [OPTIONS] CONTAINER [REPOSITORY/ImageName:TAG]
```

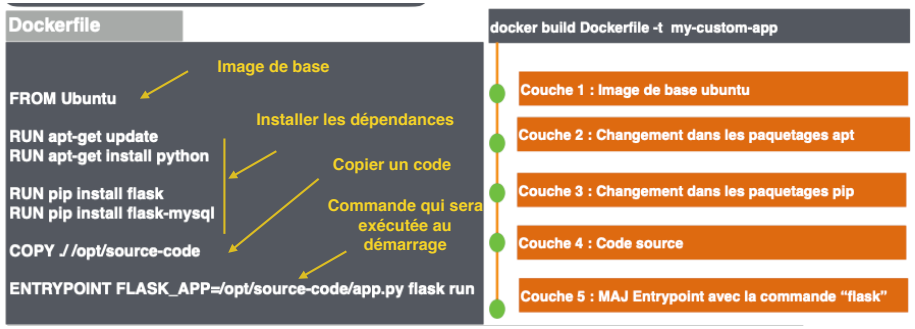
2. Construction d'une image à partir du fichier Dockerfile

- Une image est construite à partir d'un fichier de description appelé Dockerfile
- Le fichier Dockerfile contient les commandes nécessaires pour :
  - télécharger une image de base
  - appliquer une série de commandes spécifiques
  - décrire l'environnement d'exécution

Gestion d'une image— Construction d'une image

- docker build : créer une image à partir d'un Dockerfile

```
docker build [OPTIONS] PATH
```



Gestion d'un conteneur— Création d'un conteneur

- docker create : créer un conteneur à partir d'une image

```
docker create  -- name ContainerName [OPTIONS] ImageName:TAG [COMMAND]
```

- ContainerName — Nom du conteneur à créer
- ImageName:TAG — Nom de l'image à exécuter et version
- Command —Commande à exécuter dans le docker
- Exemples d'options :
  - **-v HostVolume : ContainerVolume** — Crée un volume partagé entre l'hôte et le conteneur
  - **- - network** — Choisir le réseau du Docker
  - **p HostPort : DockerPort** — mapping de port entre l'hôte et le conteneur

Gestion d'un conteneur— Commandes

- Lancer un conteneur à partir d'une image

```
docker run - -name ContainerName ImageName:Tag command
```

- Lancer un conteneur (existant)

```
docker start ContainerName
```

  - docker run = docker create + docker start

- Arrêter un conteneur

```
docker stop ContainerName
```

- Lister les conteneurs exécutés

```
docker ps
```

- Lister les conteneurs existants

```
docker ps -a
```

- Exécuter une commande dans un docker existant

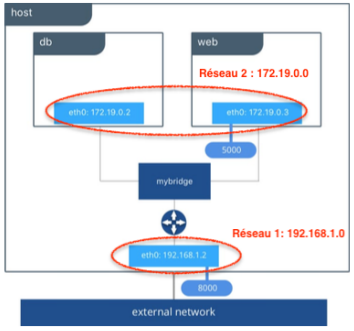
```
docker exec -it ContainerName Command
```


Gestion d'un conteneur— Réseau Docker

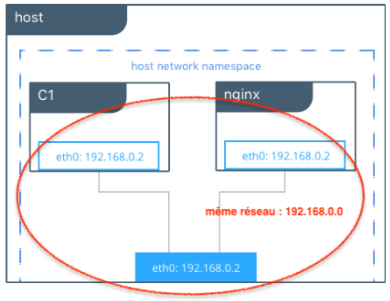
- Lors de la création d'un conteneur, trois réseaux sont automatiquement définis :
  - **Bridge** : le réseau interne par défaut attaché au conteneur
  - **Host :** réseau de la machine hôte
  - **None** : réseau isolé, le conteneurs est complètement et n'a pas d'accès à aucun réseau et n'a pas d'accès au monde externe
- Tous les conteneurs sont associés par défaut au réseau Bridge et ont des adresses à l'intérieur
- Pour permettre l'accès aux conteneurs depuis l'extérieur, il faut faire le mapping de port entre l'hôte et le conteneur ou associer le conteneur au réseau "host"
- Deux conteneurs qui sont crées sur le même réseau peuvent communiquer directement sans aucune configuration soit via leurs adresses IP ou leurs hostnames

Gestion d'un conteneur— Réseau Docker

Driver : Bridge



Driver : host



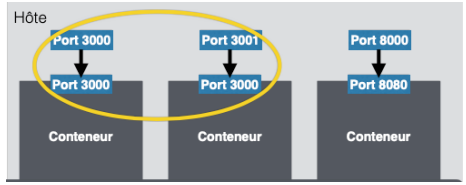
Source : https://devopssec.fr/

Gestion d'un conteneur— Port Publishing

- Par défaut et lors de la création, un conteneur ne partage aucun port avec l'extérieur
  - La publication de ports réseau est un moyen de permettre au conteneur de communiquer avec d'autres conteneurs et le monde externe.
- Publier un port (- - publish / -p)
  - Publie un port conteneur à un port host (mapping)
  - Exemple d'utilisation : Transfert de tout le trafic de données entrant sur le port hôte 8000 au port conteneur 8080

```
docker run --publish 8000:8080 image
```

  - 8000 : port usuel de l'application conteneurisée
  - 8080 : port hôte (accès conteneur)



Gestion d'un conteneur— Port EXPOSE

- Exposer un port (EXPOSE, utilisé dans le DOCKERFILE)
  - Indique le port sur le quel le conteneur écoute / accepte tous le trafic entrant.
  - Correspond généralement au port usuel de l'application
  - Exemple d'utilisation : EXPOSE 80 (Apache web server) / EXPOSE 3306 (MySQL server)
- Rôle de EXPOSE

(1) Fournir de la documentation

- Indique le port sur le quel le conteneur écoute / accepte tous le trafic entrant.

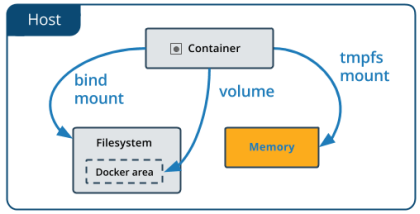
> **" The EXPOSE instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published."—** docs.docker.com

(2) Informe Docker Deamon sur le port de l'application si le flag '-P' est utilisé

- Le flag '-P' (—publish-all) indique à Docker de publier tous les ports de l'application : Docker a besoin de connaitre ces ports (see exposed ports)

Gestion d'un conteneur— Gestion des données Docker

- Les données dans un conteneur sont éphémères : les fichiers créés à l'intérieur d'un conteneur sont stockés sur un espace de travail : couche de conteneur inscriptible (writable).
  - Cela signifie que : Les données ne sont pas conservées lorsque ce conteneur est supprimé
- Trois méthodes de stockages :
  - **tmpfs mount** : (éphémère) permet de stocker les données dans la mémoire du système hôte en plus de la couche inscriptible
  - **volume** : configurer un volume **Docker** commun entre hôte et conteneur au moment du lancement
  - **bind mount** : monter un répertoire du système de fichier hôte au conteneur lors de son lancement



docs.docker.com

Autour de Docker—Ecosystème Docker

- Les L'écosystème Docker regroupe tous les utilitaires utilisés avec Docker :
  - **Docker Engine** : c'est l'application client-serveur open source dont les composants centraux sont : démon et client Docker.
  - **Docker machine** : outil de gestion et provisioning des machines virtuelles exécutant Docker. C'est un script utilisé pour la création des machines virtuelles utilisant docker.
  - **Docker Swarm** : Un swarm est un groupe de machines exécutant Docker et appartenant à un Cluster
  - **Docker Compose** : outil permettant de définir le comportement de plusieurs conteneurs qui dépendent les uns des autres et qui permet leur lancement

- **Docker Compose**

```
version: '3.7'
services:
  db:
    image: mysql:5.7
    container_name: mysql_c
    restart: always
    volumes:
      - db-volume:/var/lib/mysql
      - ./articles.sql:/docker-entrypoint-initdb.d/articles.sql
    environment:
      MYSQL_ROOT_PASSWORD: test
      MYSQL_DATABASE: test
      MYSQL_USER: test
      MYSQL_PASSWORD: test
  app:
    image: myapp
    container_name: myapp_c
    restart: always
    volumes:
      - ./app:/var/www/html
    ports:
      - 8080:80
    depends_on:
      - db
volumes:
  db-volume:
```

Services : conteneurs

Conteneur1 :
Nom:db
image: mysql:5.7

Conteneur2 :
Nom:app
image: myapp

Commande	Description
docker-compose -f docker-compose.yml up	**Lancement des conteneurs**
docker-compose -f docker-compose.yml down	**Destruction des conteneurs**

Agenda

1. Virtualisation versus conteneurisation
2. Les Fondations Linux
   2.1. chroot
   2.2. cgroups
   2.3. namespace
3. Les outils de conteneurisation
   3.1. LXC
   3.2. Docker
4. Kubernetes

4. KUBERNETES

Motivations : Pourquoi?

- **Gestion des conteneurs en Cluster**
  - Besoin de gérer et configurer plusieurs conteneurs pour une application
  - Répartition des conteneur sur un seul serveur : risque d'arrêt application si échec serveur
  - Solution : déploiement des conteneurs en Cluster :
    - ➡ Plusieurs serveurs en communication entre eux
    - ➡ Ensemble de machines configurées pour travailler ensemble et vues comme un système unique

4. KUBERNETES

Motivations : Pourquoi?

- **Orchestration des conteneurs**
  - Besoin d'appliquer des mises à jour sans interruption de l'application
  - Besoin de faire un monitoring continu et vérifier si les conteneurs sont fonctionnels
  - Relancer un conteneur si problème ou échec
  - Besoin de faire le déploiement rapide de nouveaux conteneurs en cas de besoin

L'orchestration de conteneurs est l'automatisation d'une grande partie de l'effort opérationnel requis pour exécuter des services conteneurisés : déploiement, le passage à l'échelle, la mise en réseau, équilibrage de charge entre VMs, etc.

4. KUBERNETES

Introduction à Kubernetes

- Kubernetes est une plateforme pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées
- Kubernetes a été crée et utilisé par Google
  - Annoncé comme un outil open source à 2014
  - Date de la première version Kubernetes v1.0 : 21-07-2015
- Parallèlement à sa publication, Google s'est associé à la Linux Foundation pour former la **Cloud Native Computing Foundation (CNCF)**, et a proposé Kubernetes comme technologie de base
- Kubernetes prend en charge Docker et d'autres outils de conteneurisation
- Disponible sur les plateformes Cloud : Google Container Engine, Azure, AWS EKS, etc.

Notes

Le nom Kubernetes tire son origine du Grec ancien, signifiant capitaine ou pilote
K8s est l'abréviation dérivée par le remplacement des 8 lettres "ubernete" par "8"

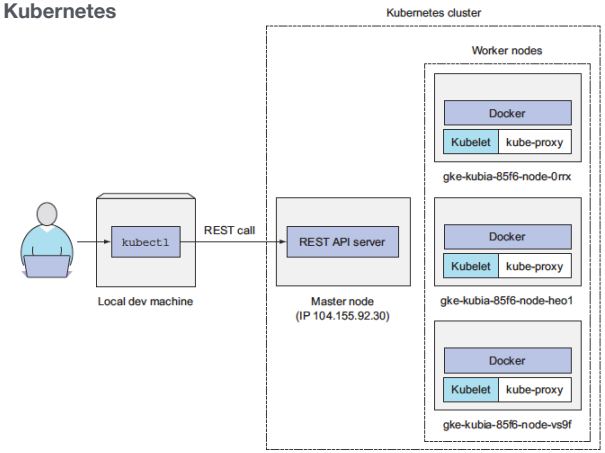
kubernetes

Introduction à Kubernetes

- L'objectif de Kubernetes est d'automatiser le déploiement des applications conteneurisées dans un cluster
- Pour ce faire, Kubernetes utilise des outils d'orchestration : REST API, CLI et une interface web graphique

- **Fonctionnalités Kubernetes :**
  - Provisionnement et déploiement des conteneurs
  - Equilibrage de charge
  - Passage à l'échelle : évolutivité
  - Augmenter la disponibilité des services
  - Allocation des ressources entre conteneurs
  - Assurer le monitoring

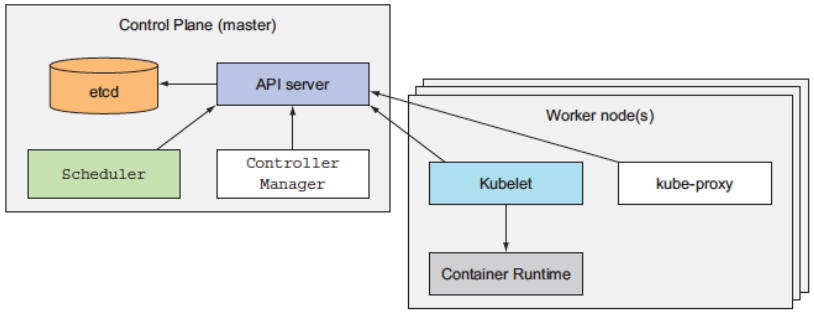
Architecture Kubernetes



Source : Marko Luksa. Kubernetes in Action. Manning Publications Co. 2018

Architecture Kubernetes

- Kubernetes est formé de deux composants essentiels :
  - **Control Plane (Master Node) :** héberge le contrôleur et le gestionnaire du système Kubernetes complet
  - **Worker node(s) :** hébergent les applications déployées



Source : Marko Luksa. Kubernetes in Action. Manning Publications Co. 2018

Architecture Kubernetes—Master Node : Control Plane

A. **Kubernetes Control Plane (Master Node) :** responsable de services qui gèrent les fonctionnalités d'orchestration

- **Les composants du Master**

(1) **API Server**
- Représente le front-end du Master : point d'accès k8s (Entrypoint to k8s cluster)
- Gère toutes les communications internes et externes entre les différents composants

(2) **etcd**
- Stockage permanent utilisé par Kubernetes pour enregistrer la configuration complète du Cluster.
- Conserve l'état actuel du Cluster : données de configuration, status des Worker nodes et de leurs conteneurs (données) : conserve l'état actuel de chaque composant
- Outil de restauration du Cluster complet

(3) **Scheduler :** élément responsable de la planification et de l'organisation des applications.
- Fait l'affectation des charges de travail (conteneurs) selon la disponibilité des ressources dans les noeuds

(4) **The controller manager :** gère et supervise les composants configurés pour un cluster

Architecture Kubernetes — Worker Node

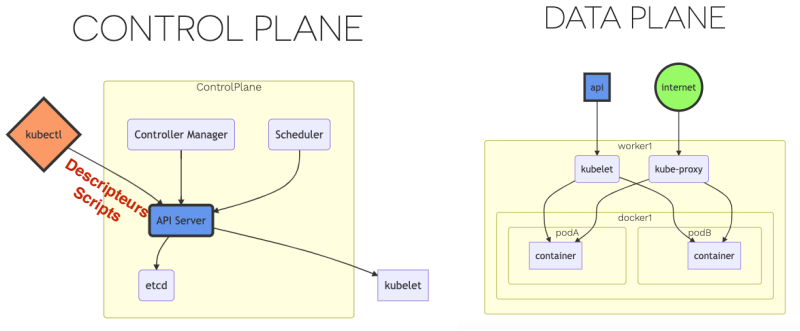
B. **Worker node :** des machines physiques ou virtuelles (VMs) qui hébergent les instances des applications.

- **Les composants du Worker node**
  - (1) **Container Runtime :** gère les opération sur les conteneurs à l'intérieur d'un noeud. Exemple : Docker
  - **(2) Kubelet**
    - Agent qui maintient le noeud et communique avec le Master via l'API Server
    - Supervise le fonctionnement des conteneurs : si un noeud tombe en panne, il sera détecté par Kubelet et l'information sera transférée au Master
  - **(3) Kube-proxy**
    - Responsable de la gestion du réseau des conteneurs telle que affectation des adresses IP, opération de publication des ports, etc.

C. kubectl

- Outil CLI fournit par Kubernetes pour gérer le cluster
- L'utilisateur utilise kubectl pour envoyer des commandes au cluster control plane (Master)
- kubectl utilise un fichier de configuration qui contient les information de configuration

Architecture Kubernetes

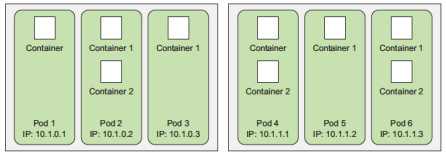


- **Master Node(s) :** dans un environnement de production, au moins deux (02) noeuds Master à l'intérieur du cluster sont déployés (pour le backup en cas de panne)
- **Worker node(s) :** nécessite des ressources de calcul importantes car les charges de travail (workloads) sont élevées

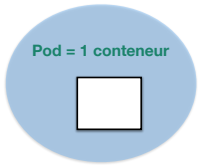
Terminologie Kubernetes — Pod

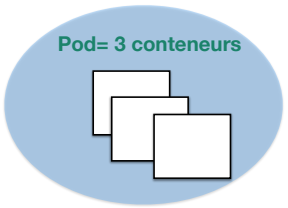
Pod

- Représente la plus petite unité orchestrable de k8s seule : unité de base
- Destiné à exécuter une instance unique d'une application : lors de la mise en échelle (élévation du nombre des applications), on multiplie les Pods (1 pour chaque instance)
- Peut regrouper plusieurs conteneurs qui sont orchestrés sur un même hôte
- Les conteneurs à l'intérieur d'un Pod fonctionnent ensemble (start & stop)
- **Le plus fréquent : un Pod englobe un seul conteneur**



Source : Marko Luksa. Kubernetes in Action. Manning Publications Co. 2018

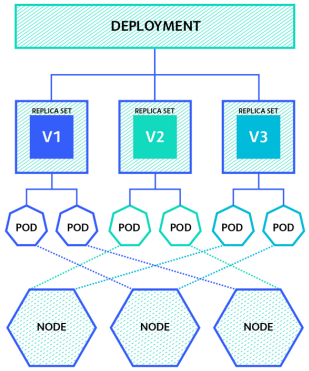




Terminologie Kubernetes — Deployment

Pod & Deployment ?

- Le deployment définit un modèle (template) pour la création du Pod
  - **Définit le nombre de copies /instances d'une application : nombre des Pods**
- Maintient les Pods en fonctionnement : si un Pod s'arrête (dies), les services de ce dernier sont relayés à une deuxième copie



Source : Cloud Native Computing Foundation (CNCF)

Terminologie Kubernetes— Service

- **Pod & Service ?**
  - Les Pods sont éphémères : meurent facilement lors d'un crash conteneur et ils seront remplacés par "Deployment" par des nouveaux
    - Mise à jour de l'adresse IP à chaque fois??
    - Solution : Création d'un service associé à un Pod qui maintient une adresse permanente pour un Pod
  - **Un service** est responsable de la création d'une address IP à un ensemble de Pods : lors de son création, un service est affecté à une seule adresse IP
  - Les cycles de vie Pod et Service ne sont pas connectés

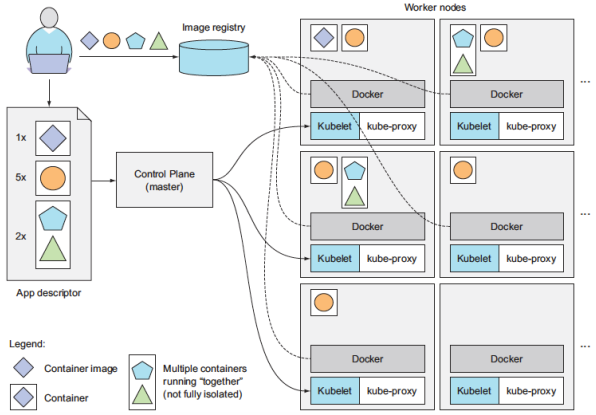
Fonctionnement Kubernetes—Déploiement des applications

- Pour exécuter une application sur Kubernetes, on doit :
  - (a) Préparer les images nécessaires
  - (b) Faire le Push des images à un dépôt images : Registry
  - (c) Poster un descripteur de l'application à l'API Server via Kubect
- **Descripteur application**
  - Contient des informations tels que : images, interactions entre les conteneurs, conteneurs qui doivent s'exécuter sur le même noeud, nombre de copies à exécuter pour chaque image, etc.
- **Fonctionnement :** L'API server procède par le traitement du descripteur :
  - Le Scheduler planifie et répartie les groupes de conteneurs spécifiés sur les Worker Nodes disponibles en se basant sur les ressources de calcul nécessaires pour chaque groupe et celles qui sont disponibles au niveau de chaque Worker Node
  - Le Kubelet de chaque Worker Node informe le gestionnaire de conteneur (Container Runtime, exemple Docker) pour faire le téléchargement des images nécessaires depuis le dépôt et leur exécution sur des conteneurs

Fonctionnement Kubernetes—Déploiement des applications

Exemple :

- Un descripteur d'application liste quatre conteneurs groupés en trois PODs
  - Les deux premiers PODs contiennent chacun un seul conteneur
  - Le troisième POD contient deux conteneurs doivent être exécutés sur le même noeud
- Le descripteur fournit de plus le nombre de copies (duplication) de chaque POD



Source : Marko Luksa. Kubernetes in Action. Manning Publications Co. 2018

1. Docker—Pratique des architectures à base de conteneurs, Dunod
2. Kubernetes in Action. Marko Luksa. Manning Publications Co. 2018
3. Kube-Kubernetes. Prof. Dr. Nabil Abdennadher. TSM-CIComp-EN Cloud Computing. Maste of Science in Engineering Course.
4. TIW - Cloud computing, Docker. Fabien Rico, Jean Patrick Gelas. Cours Université Claude Bernard. Lyon 1.