DevOps : Conteneurisation et Intégration continue

INTRODUCTION

OBJECTIFS DE LA FORMATION

- Expliquer la démarche DevOps
- Comprendre les conteneurs
- Création d'une infrastructure composée de plusieurs conteneurs
- Mise en place d’une CI/CD
- Gestion des déploiements avec Kubernetes

ORGANISATION DE LA FORMATION

Jour 1

matin : Cours - DevOps & Les conteneurs avec Docker

après midi : TP Docker

Jour 2

matin : Cours - Orchestration de conteneurs avec Docker compose & Architecture Cloud

après midi : TP Docker compose

ORGANISATION DE LA FORMATION

Jour 3

matin : Cours - CI/CD

après midi : TP GITLAB (Installation de GITLAB)

Jour 4

matin : Cours - Orchestration de conteneurs avec Kubernetes

après midi : TP GITLAB (Création d'une Pipeline CI/CD)

PRÉ-REQUIS DE LA FORMATION

- Avoir des notions en développement
- Disposer d’un ordinateur
- Connaître un minimum git.

LES ORIGINES DU DEVOPS – CULTURE DEVOPS

OBSERVATION - PROBLÉMATIQUE

Observations :

- La collaboration entre les équipes de développement (DEV) et d'exploitation (OPS) n'est pas optimale.
- Les entreprises sont souvent structurées en silos, créant des barrières entre ces deux équipes.
- Il semble parfois que ces deux équipes ne se comprennent pas parfaitement.

OBSERVATION - PROBLÉMATIQUE

Perceptions des OPS :

- Souvent perçus comme des obstacles à la rapidité de la production.
- Ils mettent l'accent sur des aspects tels que la sécurité, la sauvegarde et la haute disponibilité.
- Leur préoccupation principale concerne les méthodes de mise à jour, de déploiement, de surveillance, de supervision et d'évolutivité.

OBSERVATION - PROBLÉMATIQUE

Perceptions des DEV :

- Souvent impatients et demandent des environnements de test immédiats.
- Ils ont besoin des versions spécifiques de bibliothèques pour leurs piles technologiques.
- Ils s'inquiètent de la consommation élevée de CPU ou de RAM, se demandant si c'est normal.
- Ils sont parfois perplexes face aux journaux de données par rapport aux données d'application.
- Ils se questionnent sur la raison pour laquelle le mode de débogage est activé en production.

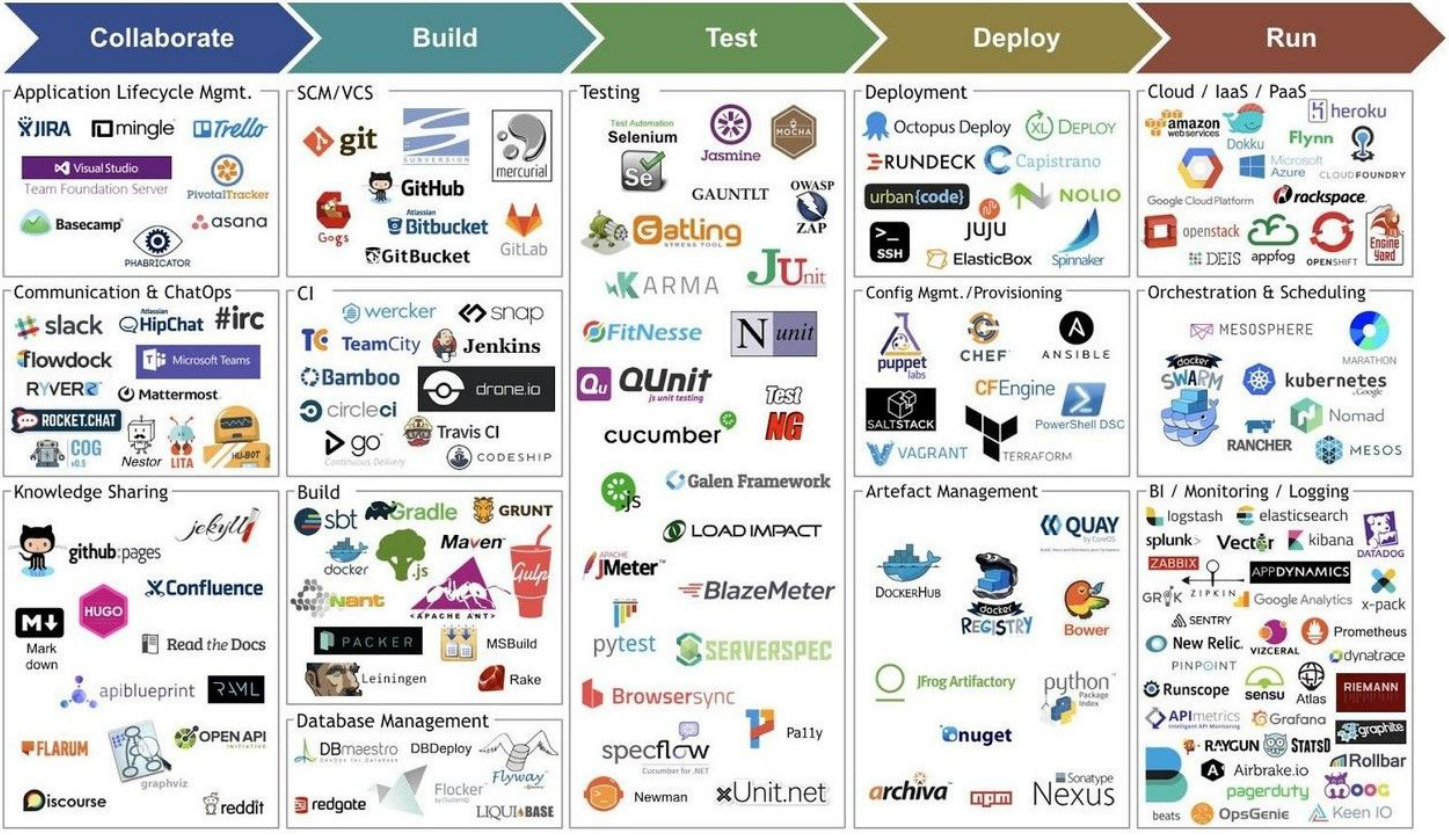
LA CULTURE DEVOPS

- La culture DevOps met l'accent sur la collaboration, l'affinité, les outils et le scaling.
- Elle encourage la communication entre équipes et la coopération au-delà des silos.
- L'objectif est de partager des objectifs communs pour améliorer l'efficacité.
- Les outils DevOps automatisent, structurent et surveillent les processus.
- Ils favorisent une gestion agile du système d'information.
- Le scaling assure que les principes DevOps s'adaptent à la croissance de l'entreprise.
- En résumé, DevOps vise à unir les équipes, à aligner les objectifs et à automatiser pour une gestion plus efficace du SI.

PRINCIPES CLÉS DU DEVOPS

- Améliorer la communication dans l'entreprise, en particulier au sein des services informatiques.
- Transformer les erreurs en opportunités d'apprentissage et d'amélioration.
- Adopter des changements plus petits mais plus fréquents pour favoriser l'agilité.
- Automatiser les tâches chronophages pour gagner en efficacité.
- Mesurer tous les indicateurs pertinents pour évaluer les performances et guider les améliorations continues.

LE CYCLE DEVOPS ET SES OUTILS



L’ECOSYSTEME DEVOPS

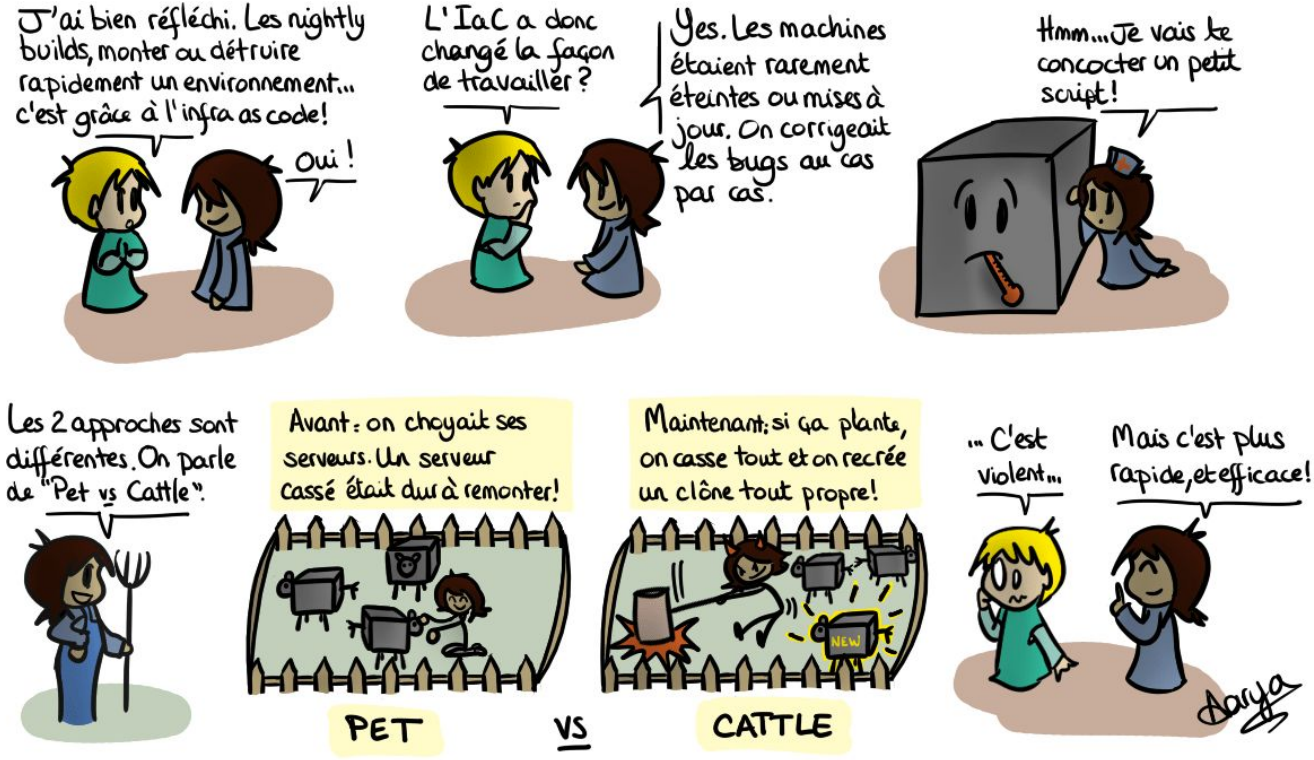
INFRASTRUCTURE AS CODE

- La base de l'IaC est l'automatisation des couches d'infrastructure.
- Elle emprunte les meilleures pratiques du développement logiciel.
- Elle inclut la gestion de configuration (CM).

Objectifs :

- Assurer la cohérence des environnements.
- Permettre une reproductibilité automatisée et sans intervention humaine.
- Favoriser la réutilisation des modèles de manière systématique.
- Offrir une scalabilité native.
- Évolue vers un modèle de gestion inspiré par "Cattle and pets."

INFRASTRUCTURE AS CODE



TYPES D'OUTILS D'IAC

Définition de l'IaC :

- Écrire et exécuter du code pour définir, déployer et mettre à jour l'infrastructure.
- Possibilité de gérer toutes les ressources via du code (serveurs, bases de données, logs, réseau, applications, documentation, tests automatiques, déploiement automatique, etc.).

4 Types d'Outils IaC :

- Scripts ad-hoc : Bash, Python, PowerShell, Perl, etc.
- Outils de gestion de configuration : Puppet, Chef, Ansible, etc.
- Génération automatique de templates de serveurs : Packer.
- Orchestrateurs : Terraform, Heat, Cloud Formation, Templates Azure, etc.

POURQUOI LES CONTENEURS ?

EVOLUTION DE L'INDUSTRIE LOGICIELLE

Avant :

Applications monolitiques Cycles de dev long

Un seul environnement

Après :

Ensemble de services

Améliorations rapides, cycles itératifs De nombreux environnements

Des serveurs de prod (dans le cloud?) Des laptops de dev

Des serveurs de test (gitlab?) etc

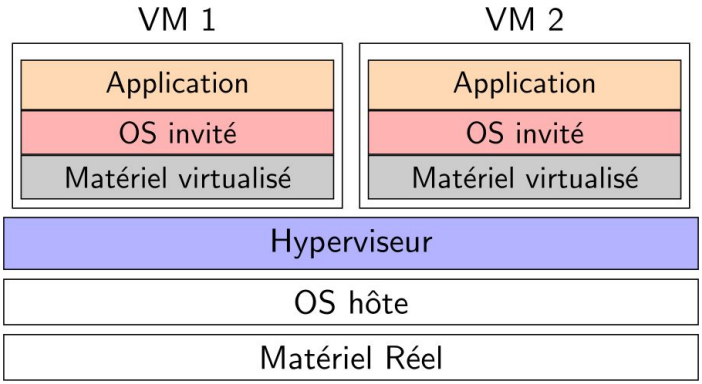
LES CONTENEURS

DEFINITION

Les conteneurs fournissent un environnement isolé sur un système hôte, semblable à un chroot sous Linux, mais en proposant plus de fonctionnalités en matière d'isolation et de configuration.

Ces fonctionnalités sont dépendantes du système hôte et notamment du kernel.

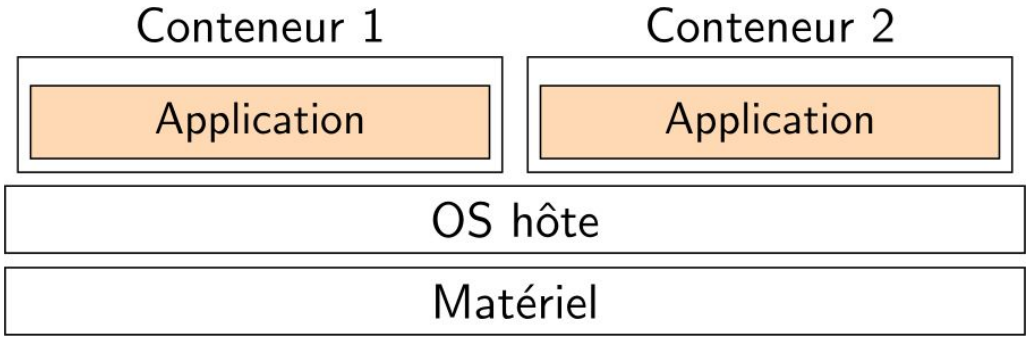
ARCHITECTURE VM



Une image de machine virtuelle contient un OS complet

La VM redémarre dans l'état qui a été sauvegardé

ARCHITECTURE CONTENEUR



Une image de conteneur ne contient que les bibliothèques nécessaires au composant conteneurisé

Les conteneurs partagent le même noyau et une grande partie des services de l'OS hôte

Pas de notion d'état

DÉCOUPLAGE DE LA LOGIQUE APPLICATIVE

Utiliser des noms de services dans votre code (bd, api, etc)

Utiliser une composition pour démarrer votre application (ou un orchestrateur si sur plusieurs serveurs)

Résolution de nom faite de manière automatique

On peut redimensionner, faire de l'équilibrage de charge, répliquer sans changer le code

PASSAGE DU DEV À LA PRODUCTION (DEVOPS)

Créer une image de conteneur et une composition

L'image peut être directement déployé en production

Même environnement pour le dev, le test, et la production

Déploiement simplifié et fluidifié

Les devs peuvent se charger du déploiement

INFRASTRUCTURE AS CODE

L'infrastructure est décrite dans des fichiers textes

Les Dockerfile (= du code)

Sert aussi de documentation de l'infrastructure

Cette description peut être visionnée (dépot git)

Suivi des modifications

Possibilité de revenir en arrière
Mise en place automatique de l'infrastructure

Déterministe Reproductible

CONTENEUR LINUX

CONTENEUR LINUX

Processus isolé par des mécanismes noyaux.

3 éléments fondamentaux

Namespaces
CGroups
Copy-On-Write

NET NAMESPACE

Le processus ne voit que la pile réseau du Namespace dont il fait partie:

Ses interfaces (eth0, l0, différentes de l'hôte)
Table de routage séparée.
règles iptables socket (ss, netstat)

NAMESPACE UTS

Identification propre au Namespace:

{get,set} hostname

NAMESPACE IPC

Permettent à un groupe de processus d'un même namespace d'avoir leurs propres:

- ipc semaphore
- ipc message queues ipc shared memory

Sans risque de conflit avec d'autres groupes d'autres namespaces
Le namespace IPC (Inter-Process Communication) isole les mécanismes de communication interprocessus (sémaphores, mémoire partagée et queues de messages).

NAMESPACE USER

mappe uid/gid vers différents utilisateurs de l'hôte
uid 0 ⇒ 9999 du C1 correspond à uid 10000⇒ 11999 sur l'hôte
uid 0 ⇒ 9999 du C2 correspond à uid 12000⇒ 13999 sur l'hôte

NAMESPACE MOUNT

Un namespace dispose de son propre rootfs (conceptuellement proche d’un chroot)

peut masquer /proc, /sys
peut aussi avoir ses mounts "privés"
/tmp (par utilisateur, par service)

CGROUPS LINUX

Fonctionnalité du noyau, apparue en 2008 (noyau 2.6.24)
Contrôle les ressources d'un processus:
allocation monitoring limite
type:
cpu, memory, network, block io, device

COPY-ON-WRITE

S’il est une chose souvent louée concernant les conteneurs c’est bien sa légèreté. L’utilisateur peut en effet s’étonner du peu de mémoire que consomme un container et de la rapidité du lancement de celui-ci.
Ces performances sont dues à un procédé très astucieux: le copy-on-write (que nous appellerons CoW).

Le copy-on-write (COW) est une technique de gestion des ressources en informatique qui est également utilisée dans les conteneurs.
En termes simples, le copy-on-write permet à plusieurs conteneurs d'utiliser la même image de base, tout en conservant une copie séparée de leurs modifications respectives

INTRODUCTION À DOCKER

DOCKER

Créé en 2011

A popularisé l'utilisation de conteneurs

D'autres technologies de conteneurs : Singularity Podman etc.

DOCKER: LES BRIQUES PRINCIPALES

Docker engine

Un environnement d'exécution et un ensemble de services pour manipuler des conteneurs docker sur une machine

Une application client-serveur

Le serveur -- Un daemon (processus persistant) qui gère les conteneurs sur une machine

Le client -- Une interface en ligne de commande

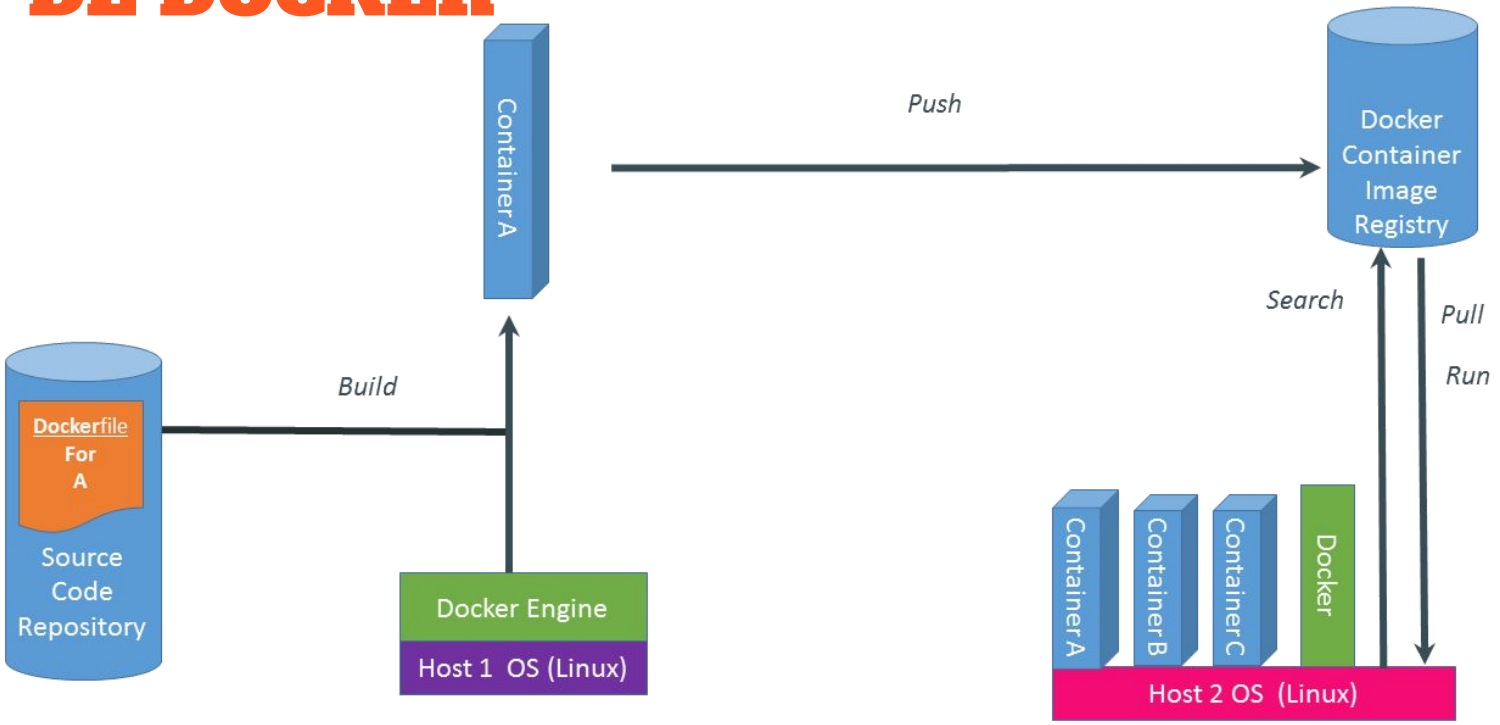
Un/des registres d'images docker

Bibliothèque d'images disponibles Docker Hub

REGISTRE DOCKER

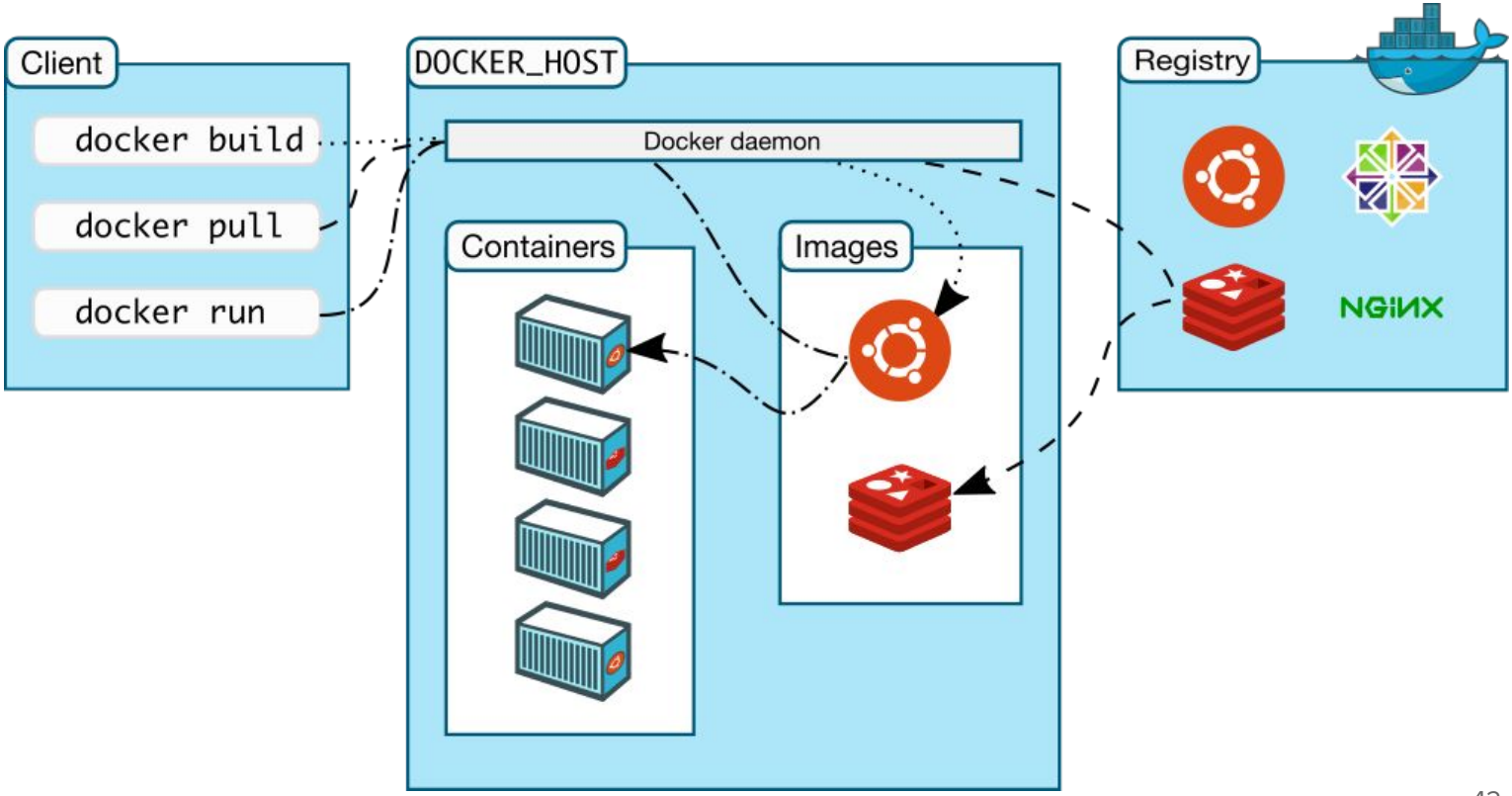
- Un serveur stockant des images docker (Exemple : Harbor)
- Possibilité de récupérer des images depuis ce serveur (pull)
  Possibilité de publier de nouvelles images (push)
- Docker Hub : Dépôt publique d'images Docker

PRINCIPES DE FONCTIONNEMENT DE DOCKER



DOCKER, VUE D'ENSEMBLE

ARCHITECTURE DE DOCKER



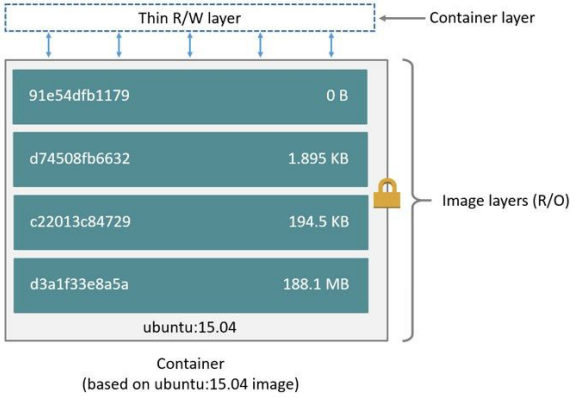
LES COUCHES

Toute image est créée à partir d'une image de base.

Exemples: ubuntu:latest, opensuse:latest, alpine:latest alpine

Image de base minimaliste (5MB) -- intéressant pour les performances

Une image n'inclut que les bibliothèques et services du système d'exploitation mentionné Le conteneur utilise le noyau du système hôte



LES COUCHES

- Les couches correspondent aux différentes modifications qui sont faites pour construire l'image à partir de l'image de base.
- Chaque couche capture les écritures faites par une commande exécutée lors de la création de l'image
- Faible espace de stockage utilisé

- Dans un conteneur en cours d'exécution, il existe une couche supplémentaire accessible en écriture
- Toutes les écritures vers le système de fichier faites à l'exécution du conteneur sont stockées dans cette couche.
- Les autres couches, définies au sein de l'image utilisée pour instancier le conteneur, ne sont accessibles qu'en lecture.
- Cette couche est supprimée à la suppression du conteneur

UTILISATION DE DOCKER

CRÉATION D'IMAGE

- Ecrire un Dockerfile
- Créer l'image à partir du Dockerfile

INSTRUCTIONS

Un Dockerfile est composé d'instructions, une par ligne.

- FROM: image de base à utiliser pour notre future image

  Un seul FROM par Dockerfile

- RUN: commande shell à exécuter

  seront exécutées durant le processus de build utilisable à volonté

  non-interactive: aucun input possible durant le build

CMD

Avec l’instruction CMD, on peut définir une commande à exécuter par défaut lorsque l’on lance un conteneur.

FROM debian

RUN apt-get update

RUN apt-get install -y nginx

CMD nginx -v

Cette commande peut être remplacée à l'exécution en passant des arguments supplémentaires à la commande docker run. CMD est souvent utilisé pour définir l'application qui doit être exécutée dans le conteneur. Si plusieurs instructions CMD sont spécifiées dans un fichier Dockerfile, seule la dernière instruction sera exécutée.

ENTRYPOINT

Définit une commande de base à exécuter par le conteneur,

Les paramètres de la ligne commande sont ajoutés à ces paramètres.

FROM debian

RUN apt-get update

RUN apt-get install -y nginx

ENTRYPOINT ["nginx","-g"]

ENTRYPOINT est également une instruction dans Dockerfile qui spécifie la commande à exécuter lorsqu'un conteneur est démarré. Cependant, contrairement à CMD, la commande spécifiée dans ENTRYPOINT ne peut pas être remplacée à l'exécution en passant des arguments supplémentaires à la commande

ENTRYPOINT

Définit une commande de base à exécuter par le conteneur,

Les paramètres de la ligne commande sont ajoutés à ces paramètres.

FROM debian

RUN apt-get update

RUN apt-get install -y nginx

ENTRYPOINT ["nginx","-g"]

ENTRYPOINT est également une instruction dans Dockerfile qui spécifie la commande à exécuter lorsqu'un conteneur est démarré. Cependant, contrairement à CMD, la commande spécifiée dans ENTRYPOINT ne peut pas être remplacée à l'exécution en passant des arguments supplémentaires à la commande

EXPOSE

Instruction Dockerfile qui indique à Docker quel(s) port(s) publier pour notre image.

Ces ports seront automatiquement exposés avec l'option -P au lancement d'un conteneur.

COPY

L’instruction COPY permet de copier des fichiers et dossiers depuis le contexte de génération, dans le conteneur.

ADD

’instruction ADD fonctionne comme COPY, avec des fonctionnalités en plus:
Peut récupérer des fichiers en ligne (URL http://)

Peut automatiquement décompresser des zip locaux

VOLUME

Les volumes peuvent être partagés:

entre conteneurs

entre hôte et un conteneur

Les accès au système de fichiers via un volume outrepassent le CoW:

Meilleures performances

Ne sont pas enregistrés dans une couche pour ne pas être enregistrés par un docker commit

VOLUME

Les volumes existent indépendamment des conteneurs

Si un conteneur est stoppé, ses volumes sont encore disponibles

Vous êtes responsable de la gestion, de la sauvegarde des volumes

BUILD AND RUN

Depuis le dossier contenant le Dockerfile

L’image obtenue permet de démarrer un conteneur, de manière similaire à celle créée manuellement:

TAG

Les images peuvent avoir des tags.

Un tag définira une version, une variante différente d'une image

par défaut le tag est latest:

docker run ubuntu == docker run ubuntu:latest

Un tag est juste un alias, un surnom pour un identifiant d'image

OPTIMISER LES IMAGES

Nombre de couches

Chaque commande du Dockerfile crée une nouvelle couche: Limiter le nombre de couches peut améliorer les performances

Docker Build multi-stages

Permet de réduire la taille des images.

Permet par exemple de sélectionner seulement certaines des modifications générées lors d'une étape de construction pour les intégrer dans l'image finale

OPTIMISER LES IMAGES

Plusieurs instructions FROM peuvent être utilisées dans un Dockerfile

Chacune définit le début d'une nouvelle stage

Chacune peut utiliser une image de base différente

Seule la dernière stage est conservée dans l'image finale

On peut COPY des fichiers d'une étape précédente dans la nouvelle étape

OPTIMISER LES IMAGES

```
FROM ubuntu AS compiler
RUN apt-get update
RUN apt-get install -y build-essential
COPY hello.c /
RUN make hello
FROM ubuntu
COPY --from=compiler /hello /hello
CMD /hello
```

MANIPULER LES CONTENEURS

Quelques commandes

ls: Montre les conteneurs en cours d'exécution

start: Démarre un conteneur arrêté

stop/restart: Arrête/redémarre un conteneur en cours d'exécution

rm: Supprime un conteneur arrêté

prune: Supprime tous les conteneurs arrêtés

logs: Récupère les logs d'un conteneur

stats: Obtenir les informations sur la consommation de ressources d'un conteneur

top: affiche la liste des processus du conteneur

DEBBUGER LES CONTENEURS

Comment observer ce qu'il se passe dans un conteneur: Se connecter à un conteneur en cours d'exécution avec docker exec
Permet d'exécuter une commande ou d'ouvrir un shell

```
$ docker exec -it f136fa721110 bash
root@f136fa721110:/#
```

AFFICHAGE DE L'ENSEMBLE DES COUCHES D'UNE IMAGE

Avantages des Mécanismes de Couches :

- Stockage efficace : Évite la duplication de données.
- Économie de bande passante : Ne télécharge que les couches nécessaires.
- Optimisations à l'exécution : Évite la recopie de fichiers.
- Démarrage rapide : Création rapide de conteneurs.
- Utilisation minimale de l'espace disque : Partage de couches en lecture seule.
- Commande pour Afficher l'Historique des Couches :

$ docker history image_name

LES NAMESPACE D'IMAGES

3 namespaces

les images officielles

ex: ubuntu, busybox

Les images d'utilisateurs/organisations

ex: tropars/myapp

Les images hébergées (en dehors de docker hub)

ex: registry.example.com:5000/my-private/image

LES NAMESPACE D'IMAGES

3 namespaces

les images officielles

ex: ubuntu, busybox

Les images d'utilisateurs/organisations

ex: tropars/myapp

Les images hébergées (en dehors de docker hub)

ex: registry.example.com:5000/my-private/image

LISTE D'IMAGES STOCKÉES LOCALEMENT

```
$ docker images
REPOSITORY                TAG             IMAGE ID       CREATED         SIZE
jekyll-github-page        latest          65736152d458   4 days ago      735MB
maven                     3.8.1-jdk-11    4e8fadf07a8c   7 months ago    658MB
openjdk                   11-jdk          a9c7e4ad6720   8 months ago    647MB
ubuntu                    20.04           7e0aa2d69a15   8 months ago    72.7MB
```

TÉLÉCHARGER DES IMAGES

docker pull: télécharge une image explicitement

docker run: Commande servant à créer un conteneur à partir d'une image

Télécharge l'image si elle n'est pas présente localement

A PROPOS DES TAGS

Pas besoin de tags quand

- On prototype/teste
- On veut la dernière version d'une image

Tags à utiliser quand

- Quand on va en production
- Pour s'assurer que la même version va être utilisée partout
- Pour avoir de la reproductibilité

PREMIERS PAS AVEC DOCKER

L'outil à la ligne de commande pour exécuter des commandes Docker: docker run hello-world

docker: Nous voulons exécuter une commande docker

run: Commande pour créer et exécuter un conteneur docker

hello-world: Nom de l'image à partir de laquelle est construit le conteneur

INSPECTER LES CHANGEMENTS:

Inspecter les changements:

```
root@d2-8-gra1:/home/debian# docker diff 247fbef77841
C /run
A /run/docker.sock
A /letsencrypt
```

C: Fichier ou répertoire modifié

A: Fichier ou répertoire ajouté

LA GESTION DU RÉSEAU

LA GESTION DU RESEAU

Les conteneurs sont souvent utilisés pour déployer des applications communiquant sur le réseau.

Plusieurs questions:

Comment communiquer avec un appli conteneurisé depuis l'extérieur?

Comment communiquer entre conteneurs?

Sur un même noeud?

Sur des noeuds différents?

LA GESTION DU RESEAU

Quand vous installez Docker, 3 réseaux sont créés automatiquement, bridge, none, et host, suivant 3 pilotes bridge, null et host

Commande pour voir les réseaux : docker network ls

Le pilote bridge interconnecte les conteneurs qui se trouve sur un réseau de ce type de pilot

LA GESTION DU RESEAU

none:

type null: aucun réseau pour un conteneur sur un réseau

de ce type

host:

type host: stack réseau identique à l'hôte

Vous n'aurez sûrement jamais à utiliser ces réseaux, et créer

des réseaux de ces types.

LA GESTION DU RESEAU

```
$ docker run -d -P nginx
```

-d lance le conteneur en arrière plan

-P rend accessible tous les ports exposés par le conteneur au niveau de la machine hôte

```
docker ps
CONTAINER ID   IMAGE     COMMAND                  PORTS
f136fa721110   nginx     "/docker-entrypoint.…"   0.0.0.0:49153->80/tcp, :::49153->80/tcp
```

Envoyer une requête sur le port 49153 de la machine permet d'atteindre le serveur web dans le conteneur

BILAN

On sait créer des images

- de manière manuelle
- de manière automatisée

On sait lancer des conteneurs

- partager les données avec des volumes
- les interconnecter sur le réseau

PROBLEME

On veut coordonner des conteneurs

On veut simplifier la gestion multi-conteneurs

On ne veut pas utiliser de scripts shell complexes

On veut une interface standardisée avec l'API Docker

DOCKER COMPOSE

MANIPULER LES CONTENEURS

Un outil complémentaire du Docker Engine

Présenté dans les versions récentes comme un plugin de Docker Engine

Ancienne syntaxe: docker-compose command

Nouvelle syntaxe: docker compose command

Permet de décrire une application construite à base de conteneurs dans un fichier YAML

Déploiement sur une seule machine

Scénario visé Checkout du code sur un dépôt

Executer docker compose up... Et c'est tout!

L'application est configurée et démarrée

SERVICES

Compose introduit une notion de service:

Concrétement, un conteneur exécutant un processus

Chaque conteneur exécute un service inter-dépendant

Le service peut-être évolutif en lançant plus ou moins

d’instances du conteneur avec Compose.

Exemple Wordpress:

Service db

Service wordpress

EXEMPLE DOCKER-COMPOSE.YML

```
mariadb:
    image: mysql
    environment:
     - MYSQL_ROOT_PASSWORD=password
     - MYSQL_DATABASE=wordpress
    volumes:
     - ./database:/var/lib/mysql
```

EXEMPLE DOCKER-COMPOSE.YML

```
wordpress:
    image: wordpress
    links:
     - mariadb:mysql
    environment:
     - WORDPRESS_DB_PASSWORD=password
     - WORDPRESS_DB_USER=root
    ports:
     - 80:80
    volumes:
     - ./html:/var/www/html
```

RESUME DOCKER COMPOSE

Compose est un outil pour définir, lancer et gérer des services qui sont définis comme une ou plusieurs instances d'un conteneur,

Compose utilise un fichier de configuration YAML comme définition de l'environnement,

Avec docker-compose on peut générer des images, lanceret gérer des services, ...

Certaines commandes de docker-compose sont équivalentes à l'outil docker, mais s'appliquent seulement aux conteneurs de la configuration de compose.

POURQUOI MIGRER DANS LE CLOUD?

LE CONTEXTE

Les entreprises déjà très performantes migrent, d'autres migrent parce qu'elles n'ont pas le choix.

Une triple opportunité pour aller dans le cloud : commerciale, sécuritaire et de réduction de l'obsolescence.

POURQUOI LE CLOUD ? CÔTÉ ÉCONOMIQUE

- Appréhender les ressources IT comme des services “fournisseur”,
- Faire glisser le budget “investissement” (Capex) vers le budget “fonctionnement” (Opex),
- Réduire les coûts en mutualisant les ressources, et éventuellement avec des économies d'échelle,
- Réduire les délais,
- Aligner les coûts sur la consommation réelle des ressources.

POURQUOI LE CLOUD ? CÔTÉ TECHNIQUE

- Abstraire les couches basses (serveur, réseau, OS, stockage)
- S'affranchir de l'administration technique des ressources et services (BDD, pare-feux, load-balancing, etc.)
- Concevoir des infrastructures scalables à la volée
- Agir sur les ressources via des lignes de code et gérer les infrastructures "comme du code"

LES ENJEUX

- Répondre aux nouveaux enjeux business et à la pression des directions métiers;
- Gagner en agilité, réactivité, flexibilité et time-to-market, déploiement à l'échelle;
- Bénéficier des nouvelles technologies, services packagés et notamment pour les données;
- Obtenir une couverture mondiale et une réplication géographique;
- Harmoniser et centraliser des systèmes permettant des économies d'échelle;
- Sécuriser ses systèmes d'information;
- Réduire la dette technique par l'utilisation des services managés et d'outils d'automatisation proposés par les fournisseurs cloud;
- Rationaliser ses datacenters et ses infrastructures;
- Bâtir des business plateformes et ouvrir des potentialités d'innovation.

LES FREINS

- Compréhension des dirigeants des enjeux du cloud;
- Exigences réglementaires (OIV);
- Exposition aux lois extraterritoriales lors du choix d'un fournisseur international;
- Confidentialité des données de l'entreprise;
- Protection des secrets technologiques et industriels;
- Performance économique attendue;
- Maturité interne de l'organisation: ressources humaines, compétences

LE CLOUD, VUE D'ENSEMBLE

CARACTERISTIQUES

- Fournir un (des) service(s)... Self service
- À travers le réseau
- Mutualisation des ressources
- Élasticité rapide
- Mesurabilité

SELF SERVICE

- L'utilisateur accède *directement* au service
- Pas d'intermédiaire humain
- Réponses immédiates
- Catalogue de services permettant leur découverte

À TRAVERS LE RÉSEAU

- L'utilisateur accède au service à travers le réseau
- Le *fournisseur* du service est distant du *consommateur*
- Réseau = internet ou pas
- Utilisation de protocoles réseaux standards (typiquement : HTTP)

MUTUALISATION DES RESSOURCES

- Un cloud propose ses services à de multiples utilisateurs/organisations (*multi-tenant*)
- *Tenant* ou *projet* : isolation logique des ressources Les ressources sont disponibles en grandes quantités (considérées illimitées)
- Le taux d'occupation du cloud n'est pas visible
- La localisation précise des ressources n'est pas visible

ÉLASTICITÉ RAPIDE

- Provisionning et suppression des ressources quasi instantané
- Permet le *scaling* (passage à l'échelle)
- Possibilité d'automatiser ces actions de *scaling*

Virtuellement pas de limite à cette élasticité

MESURABILITÉ

- L'utilisation des ressources cloud est monitorée par le fournisseur
- Le fournisseur peut gérer son *capacity planning* et sa facturation à partir de ces informations
- L'utilisateur est ainsi facturé en fonction de son usage précis des ressources
- L'utilisateur peut tirer parti de ces informations

MODÈLES

On distingue :

- modèles de service : IaaS, PaaS, SaaS
- modèles de déploiement : public, privé, hybride

IAAS

Infrastructure as a Service

- Compute (calcul)
- Storage (stockage)
- Network (réseau)
- Utilisateurs cibles : administrateurs (système, stockage, réseau)

PAAS

Platform as a Service

Désigne deux concepts :

- Environnement permettant de développer/déployer une application (spécifique à un langage/framework - exemple : Python/Django)
- Ressources plus haut niveau que l'infrastructure, exemple : BDD
- Utilisateurs cibles : développeurs d'application

SAAS

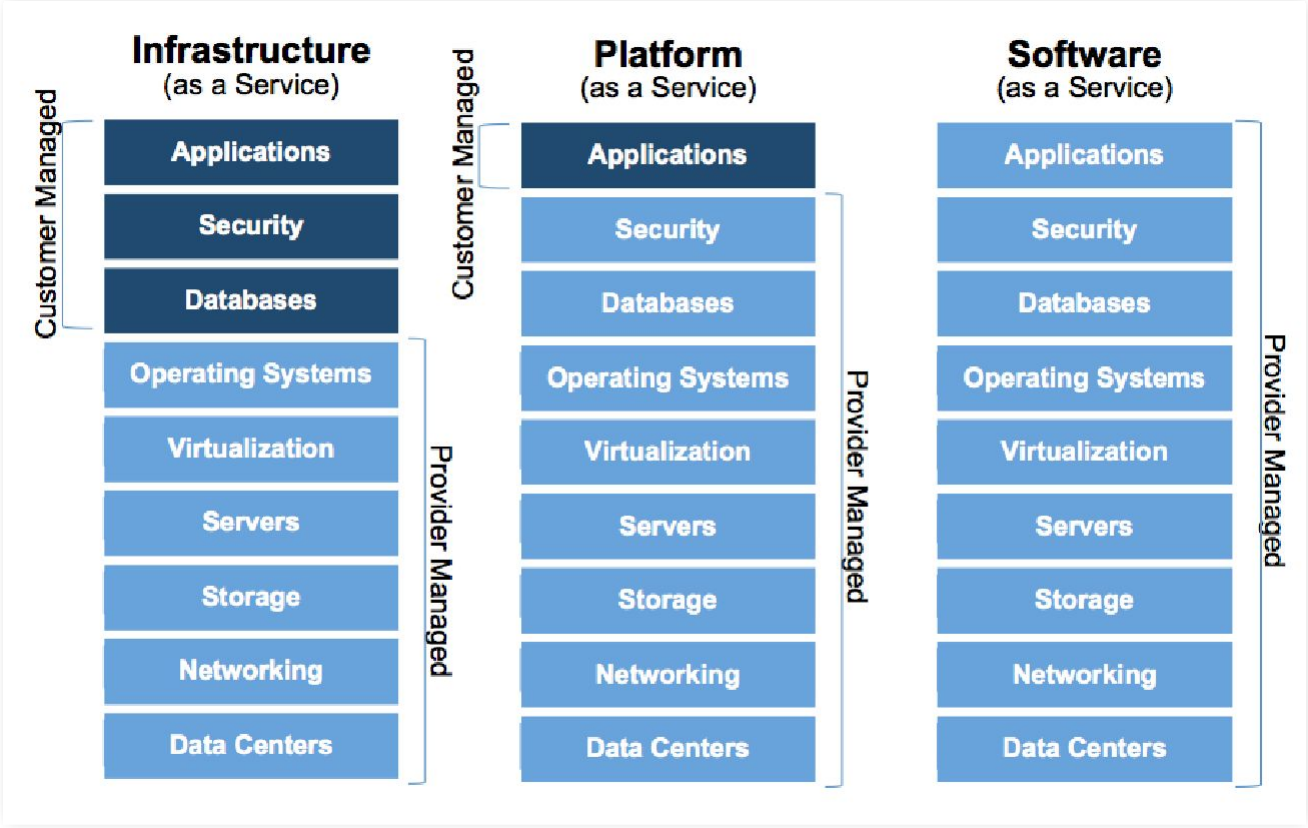
Software as a Service

- Utilisateurs cibles : utilisateurs finaux
- Ne pas confondre avec la définition *économique* du SaaS

QUELQUECHOSE AS A SERVICE ?

- Load balancing as a Service (Infra)
- Database as a Service (Platform)
- MonApplication as a Service (Software)
- etc.

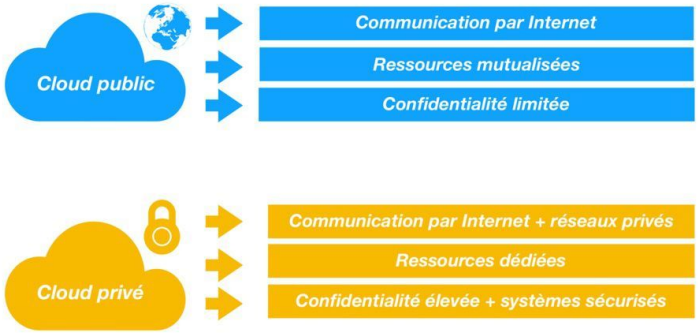
LES MODÈLES DE SERVICE EN UN SCHÉMA



CLOUD PUBLIC OU PRIVÉ ?

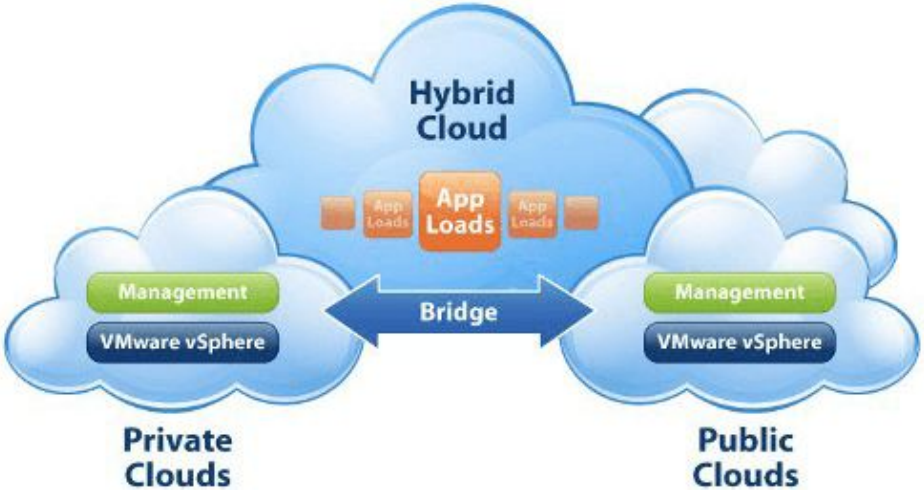
À qui s'adresse le cloud ?

- Public : tout le monde, disponible sur internet
- Privé : à une organisation, disponible sur son réseau



CLOUD HYBRIDE

Utilisation mixte de multiples clouds privés et/ou publics
Concept séduisant mais mise en œuvre a priori difficile
Certains cas d'usages s'y prêtent très bien
Intégration continue (CI)
Éviter le *lock-in*



L'INSTANT VIRTUALISATION

La virtualisation est une technologie permettant d'implémenter la fonction *compute*

Un cloud fournissant du compute *peut* utiliser la virtualisation

Mais peut également utiliser :

Du bare-metal

Des containers (système)

LES APIS, LA CLÉ DU CLOUD

Rappel : API pour *Application Programming Interface*
Au sens logiciel : Interface permettant à un logiciel d'utiliser une bibliothèque
Au sens cloud : Interface permettant à un logiciel d'utiliser un service (XaaS)
Interface de programmation (via le réseau, souvent HTTP)
Frontière explicite entre le fournisseur (provider) et l'utilisateur (user)
Définit la manière dont l'utilisateur communique avec le cloud pour gérer ses ressources
Gérer : CRUD (Create, Read, Update, Delete)

API REST

Une ressource == une URI (*Uniform Resource Identifier*)
Utilisation des verbes HTTP pour caractériser les opérations (CRUD)
GET
POST
PUT
DELETE
Utilisation des codes de retour HTTP
Représentation des ressources dans le corps des réponses HTTP

REST - EXEMPLES

GET http://endpoint/volumes/
GET http://endpoint/volumes/?size=10
POST http://endpoint/volumes/
DELETE http://endpoint/volumes/xyz

LE MARCHE DU CLOUD

AMAZON WEB SERVICES (AWS), LE LEADER

Lancement en 2006
À l'origine : services web "e-commerce" pour développeurs
Puis : d'autres services pour développeurs
Et enfin : services d'infrastructure Récemment, SaaS

ALTERNATIVES IAAS PUBLICS À AWS

Google Cloud Platform
Microsoft Azure
DigitalOcean
En France :
Cloudwatt (Orange Business Services)
Numergy (SFR)
OVH
Ikoula
Scaleway
Outscale

FAIRE DU IAAS PRIVÉ

OpenStack
CloudStack
Eucalyptus
OpenNebula

OPENSTACK EN QUELQUES MOTS

Naissance en 2010
Fondation OpenStack depuis 2012
Écrit en Python et distribué sous licence Apache 2.0
Soutien très large de l'industrie et contributions variées

EXEMPLES DE PAAS PUBLIC

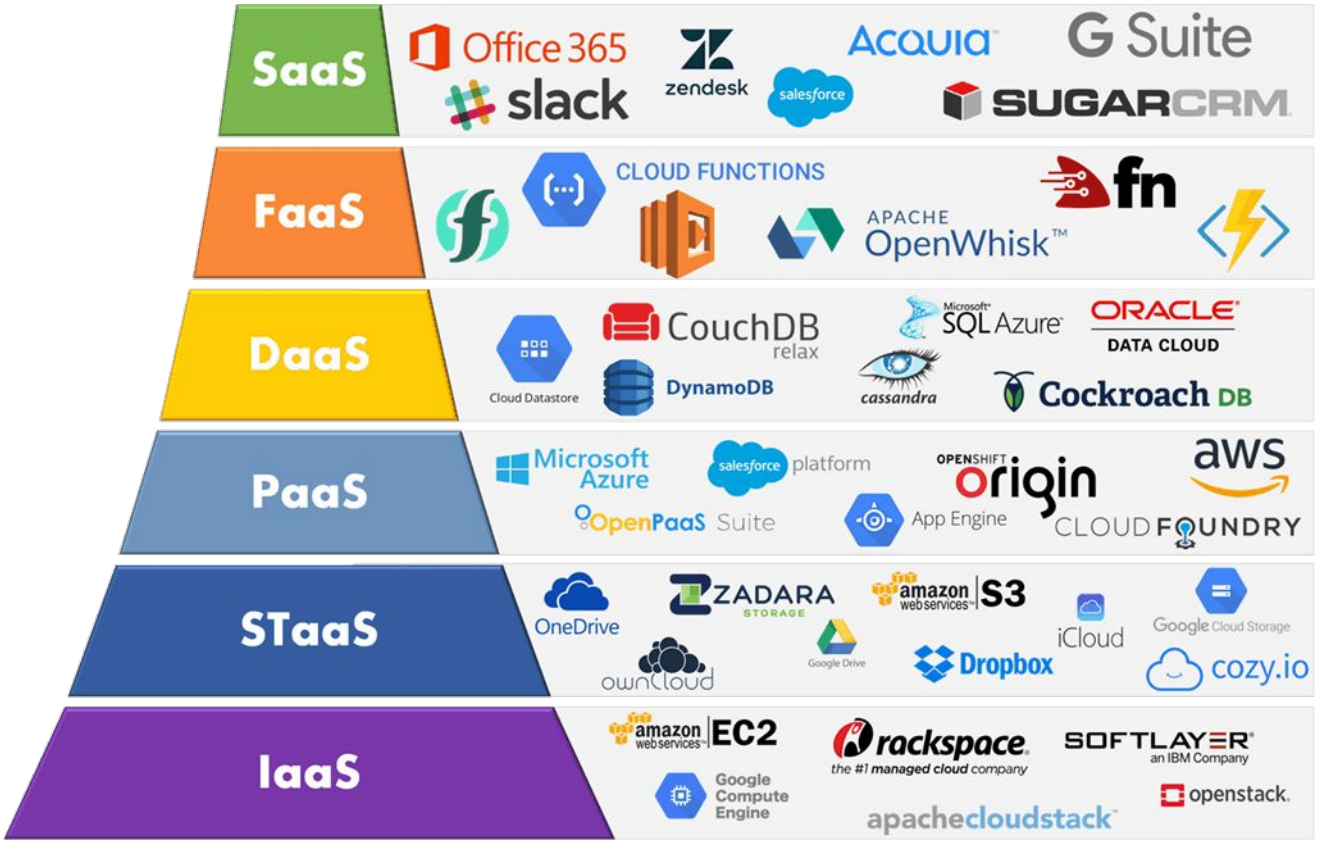
Amazon Elastic Beanstalk (https://aws.amazon.com/fr/elasticbeanstalk)

Google App Engine (https://cloud.google.com/appengine)
Heroku (https://www.heroku.com)

SOLUTIONS DE PAAS PRIVÉ

Cloud Foundry, Fondation (https://www.cloudfoundry.org)
OpenShift, Red Hat (https://www.openshift.org)
Solum, OpenStack (https://wiki.openstack.org/wiki/Solum)

LES MODÈLES DE SERVICE EN UN SCHÉMA



LES CONCEPTS INFRASTRUCTURE AS A SERVICE

LA BASE

Infrastructure :

- Compute
- Storage
- Network

RESSOURCES *COMPUTE*

- Instance
- Image
- Flavor (gabarit)
- Paire de clé (SSH)

INSTANCE

- Dédiée au compute
- Durée de vie typiquement courte, à considérer comme éphémère
- Ne doit pas stocker de données persistantes
- Disque racine non persistant
- Basée sur une image

IMAGE CLOUD

- Image disque contenant un OS déjà installé
- Instanciable à l'infini
- Sachant parler à l'API de metadata

API ... DE METADATA

http://169.254.169.254

Accessible depuis l'instance

Fournit des informations relatives à l'instance

Expose les *userdata*

L'outil cloud-init permet d'exploiter cette API

FLAVOR (GABARIT)

Instance type chez AWS
Définit un modèle d'instance en termes de CPU, RAM, disque (racine), disque éphémère

Le disque éphémère a, comme le disque racine, l'avantage d'être souvent local donc rapide

PAIRE DE CLÉ

- Clé publique + clé privée SSH
- Le cloud manipule et stocke la clé publique
- Cette clé publique est utilisée pour donner un accès SSH aux instances

RESSOURCES RÉSEAU 1/2

- Réseau L2
- Port réseau
- Réseau L3
- Routeur
- IP flottante
- Groupe de sécurité

RESSOURCES RÉSEAU 2/2

- Load Balancing as a Service
- VPN as a Service
- Firewall as a Service

RESSOURCES STOCKAGE

Le cloud fournit deux types de stockage

- Block Storage
- Object Storage

STOCKAGE BLOCK

- Volumes attachables à une instance
  Accès à des raw devices type
  /dev/vdb
- Possibilité d'utiliser n'importe quel système de fichiers
  Possibilité d'utiliser du LVM, du chiffrement, etc.
- Compatible avec toutes les applications existantes
- Nécessite de *provisionner* l'espace en définissant la taille du volume

STOCKAGE BLOCK PARTAGE ?

- Le stockage block n'est pas une solution de stockage partagé comme NFS
- NFS se situe à une couche plus haute : système de fichiers
- Un volume est *a priori* connecté à une seule machine

"BOOT FROM VOLUME"

- Démarrer une instance avec un disque racine sur un volume
- Persistance des données du disque racine
- Se rapproche du serveur classique

STOCKAGE OBJET

- API : faire du CRUD sur les données
- Pousser et retirer des objets dans un container/bucket
- Pas de hiérarchie, pas de répertoires, pas de système de fichiers
- Accès lecture/écriture uniquement par les APIs
  Pas de *provisioning* nécessaire
- L'application doit être conçue pour tirer parti du stockage objet

ORCHESTRATION

- Orchestrer la création et la gestion des ressources dans le cloud
- Définition de l'architecture dans un template
- Les ressources créées à partir du template forment la stack
- Il existe également des *outils* d'orchestration (plutôt que des *services*)

BONNES PRATIQUES D'UTILISATION

POURQUOI DES BONNES PRATIQUES ?

- Adapter ses pratiques au cloud pour en tirer parti pleinement
- Risque de ne pas répondre aux attentes
- Gagner en qualité
- Gagner en sécurité
- Se contenter d'un cas d'usage *test & dev*

HAUTE DISPONIBILITÉ (HA)

Le control plane (les APIs) du cloud est HA
Les ressources provisionnées ne le sont pas forcément

INFRASTRUCTURE AS CODE

Avec du code :

- Provisionner les ressources d'infrastructure
- Configurer les dites ressources, notamment les instances

Le métier évolue : Infrastructure Developer (DevOps)

SCALING, PASSAGE À L'ÉCHELLE

Scale out plutôt que Scale up

- Scale out : passage à l'échelle horizontal
- Scale up : passage à l'échelle vertical Auto-scaling
- Géré par le cloud
- Géré par un composant extérieur

SECURITE DU CLOUD

LA PERTE DE MAÎTRISE DE GOUVERNANCE

Le client, contractant un service Cloud Computing, donne une partie de la gouvernance infrastructure IT au fournisseur. Dans ce cas, l'accord de niveau de service (SLA) doit jouer un rôle crucial pour défendre l’intérêt du client

ISOLEMENT DES ENVIRONNEMENTS ET DES DONNÉES

Le partage des ressources est l'une des caractéristiques les plus importantes du Cloud Computing.
Plusieurs clients peuvent, par exemple, partager le même serveur physique. Si la séparation des environnements n'est pas assez efficace, «invasions» entre les clients pourraient se produire.

Dans le cas d'un environnement partagé entre plusieurs clients locataires, deux sortes d'attaques sont possibles, la première de type "Guest-hopping" et la deuxième contre les hyperviseurs directement. (Failles Spectre et Meltdown)

RISQUES DE CONFORMITÉ

En externalisant certains services et processus de base, la conformité aux lois sur la protection des données et les normes réglementaires telles que PCI DSS et ISO 27001 peut devenir très compliqué.
Les fournisseurs de services peuvent imposer des restrictions sur la conduite d'un audit de leurs infrastructures

LA PUBLICATION DES INTERFACES DE GESTION

Les interfaces de gestion des services contractés (par exemple dans un modèle SaaS) sont publiées directement sur le net. Cela augmente considérablement les risques par rapport aux systèmes traditionnels dans lesquels des interfaces de gestion sont accessibles uniquement à partir des réseaux internes

La protection des données

Pour le client de services Cloud Computing, la protection des données est difficile.
Il est très difficile de sécuriser les données qui se trouvent réparties dans plusieurs emplacements.
S'assurer que les données sont traitées correctement est également compliqué parce que le contrôle sur les transferts de données est hors de la portée de son propriétaire.

La suppression dangereuse ou incomplète des données

Historiquement, la suppression sécurisée des données a été une question très complexe qui consistait à développer une série de processus pour s'assurer qu'il n'y a pas de copie des données dans n'importe quel endroit. La réutilisation des ressources matérielles est très commune dans le Cloud Computing. Un nouveau client peut, par exemple, être affecté d'une section de stockage dans lequel, jusqu'à récemment, les données d'un autre client ont eu lieu. Cela peut entraîner à un risque de perte de confidentialité, si les données précédentes n'ont pas été supprimées complètement et en toute sécurité

Les utilisateurs malveillants

Cloud Computing a besoin des profils utilisateurs de haut niveau pour son administration.

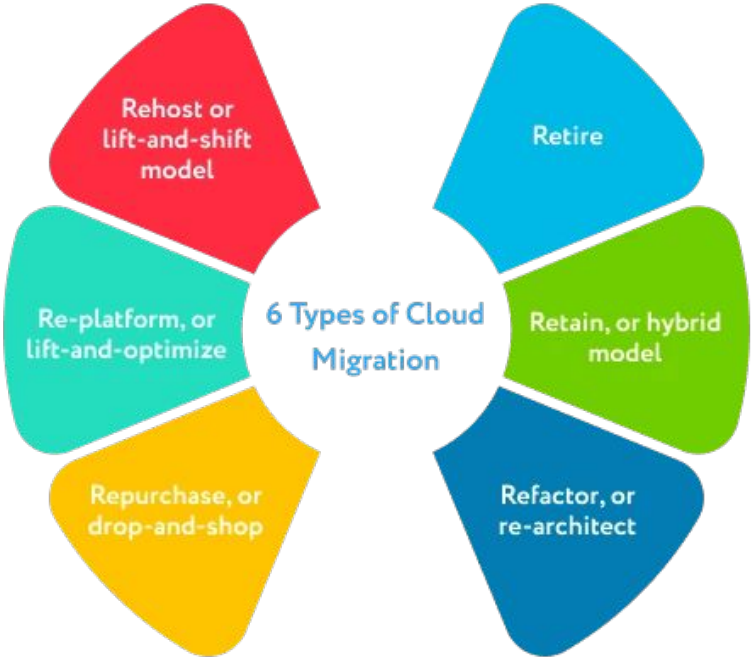
Un administrateur système aura des privilèges complets sur différentes ressources de différents clients. Un utilisateur malveillant qui compromet la sécurité du système avec succès et saisie une session admnistrateur obtiendra l'accès à de nombreuses informations clientes.

Les risques classiques : application sur le Cloud

Les risques liés à la sécurité des données dans le Cloud ne sortent pas du périmètre des risques informatiques globalement, mais ils présentent des vulnérabilités et doivent être prises en considération lors de la prise en compte de la sécurité des données.
Ces risques comprennent le hameçonnage, les accès privilèges dans le Cloud, et la source ou l'origine des données elles-mêmes.

LES 6 STRATÉGIES DE MIGRATION D'APPLICATION : « LES 6 R »

Les 6 stratégies de migration d'applications



Rehosting

Le réhébergement, autrement connu sous le nom de "lift-and-shift" est l'une des stratégies de migration vers le cloud la plus rapide et la plus simple qui déplace les données sans modification au niveau du code.

Refactoring/Re-architecting

Cette approche est motivée par un fort désir d'améliorer votre produit et représente le contraire de la migration lift-and-shift. Il est supposé qu'un objectif commercial spécifique sera défini dès le départ, par exemple en termes de disponibilité ou de fiabilité des performances de l'application. Parfois, cela signifie que vous devez repenser complètement la logique de votre application et développer la version cloud native à partir de zéro pour la rendre native au cloud.

Repurchasing (Rachat)

Le rachat, également connu sous le nom de stratégie "drop and shop", remplace l'application sur site par un logiciel fourni par le fournisseur natif du cloud. Cela signifie, que dans cette stratégie, vous modifiez l'application propriétaire utilisée pour la nouvelle plate-forme ou le nouveau service basé sur le cloud.

Utilisez la stratégie rachat si vous remplacez un logiciel pour des fonctions standard comme la finance, la comptabilité, le CRM, la GRH, l'ERP, la messagerie électronique, le CMS, etc.

Retire (retirer)

Dans cette stratégie, vous vous débarrassez des applications qui ne sont plus nécessaires ou productives pour votre portefeuille informatique. En effet, pour de nombreuses applications et environnements complexes, certains composants de l'infrastructure peuvent être facilement désactivés sans aucune diminution de productivité ou perte de valeur pour les consommateurs finaux.

Utilisez la stratégie retire si vous souhaitez archiver les applications qui contiennent des données utiles. Supprimez les applications avec des capacités de duplication pour réduire les coûts.

Retain (Retenir)

Dans cette stratégie également appelée "re-visite", consiste à revisiter certaines applications/parties critiques de vos actifs numériques qui nécessitent une refactorisation importante avant de les migrer vers le cloud.

Utilisez la stratégie retain si vous adoptez un modèle de cloud hybride lors de la migration, une application héritée peut ne pas être compatible avec le cloud et fonctionne bien sur site.

DERRIÈRE LE CLOUD

COMMENT IMPLÉMENTER UN SERVICE DE COMPUTE

- Virtualisation
- Containers (système)
- Bare metal

IMPLÉMENTATION DU STOCKAGE

- Pas de RAID matériel
- Le logiciel est responsable de garantir les données
- Les pannes matérielles sont prises en compte et gérées
- Le projet Ceph et le composant OpenStack Swift implémentent du SDS (Stockage défini par logiciel, abstraction entre le logiciel et le matériel)

GITLAB CI/CD

Intégration continue

- Assurer la construction quotidienne d'une version opérationnelle du système.
- Exécuter des tests quotidiennement pour garantir la qualité. Effectuer des commits de ses modifications dans le dépôt chaque jour.
- Mettre en place un système de surveillance des changements dans le dépôt, capable de :

- Récupérer une copie du logiciel depuis le dépôt.
- Compiler et exécuter les tests.
- En cas de succès des tests, envisager la création d'une nouvelle version du logiciel.
- En cas d'échec des tests, notifier le développeur concerné.

Intégration continue

Pour mettre en place l'intégration continue, nous devons disposer de :

- Un dépôt pour stocker le code source.
- Un processus automatisé de construction du logiciel.
- Une plateforme pour l'exécution des tests.

De plus, il est essentiel d'avoir :

- La volonté de travailler de manière incrémentale.
- Une procédure commune pour soumettre les modifications.

Intégration continue

Chaque développeur doit suivre la procédure suivante de manière systématique :

- Commencer à partir de la version la plus récente du système.
- Écrire des procédures de test et effectuer les modifications nécessaires.
- Exécuter tous les tests et s'assurer qu'ils réussissent.
- Récupérer les dernières modifications depuis le dépôt, relancer les tests, et vérifier qu'ils passent avec succès.
- Envoyer les contributions vers le dépôt.

À ce stade, les tests d'intégration continue seront automatiquement déclenchés et exécutés.

Intégration continue

Il existe divers outils que nous pouvons utiliser pour mettre en place l'intégration continue :

Dépôt pour le code source :

- Systèmes de contrôle de version tels que SVN, Git, etc.
- Plateformes d'hébergement de code comme Github.

Processus de construction automatique du logiciel :

- Outils de construction tels que Make, Ant, Maven, etc.

Plateforme pour exécuter des tests :

- Frameworks de tests unitaires tels que xUnit, JUnit, etc.
- Outils d'intégration continue comme Jenkins, Gitlab CI/CD, Github Actions, etc.
- Conteneurisation avec Docker pour des environnements de tests isolés.

Déploiement continu

Définition : L'optimisation des procédures de déploiement de logiciels vise à garantir que le logiciel peut être déployé en production à tout moment. Cela implique l'automatisation des étapes nécessaires pour déployer le logiciel de manière fiable et efficace.

Motivation : Le déploiement en production d'un logiciel est une phase critique et complexe du processus de développement. En le réalisant aussi fréquemment que possible, on réduit les risques et on acquiert une meilleure maîtrise de cette procédure. Pour atteindre cet objectif, il est essentiel de maximiser l'automatisation des étapes de déploiement, ce qui permet de garantir un déploiement rapide, cohérent et sans erreur du logiciel.

Déploiement continu

Les étapes principales d'une procédure de livraison continue sont les suivantes :

Commit d'une modification.

Exécution des tests unitaires (Intégration continue).

Tests de validation fonctionnelle (tests d'acceptation) :

- Vérifient si le logiciel répond aux besoins du client.
- Peuvent également être automatisés.

Tests de performances :

- Évaluent la capacité du logiciel à répondre à la charge.
- Mesurent les performances et la scalabilité.

Tests exploratoires :

- Tests non automatisés réalisés par des experts.
- Exemple : tests d'utilisabilité.

Le déploiement en production.

Infrastructure as Code

Dans l'approche de la "livraison continue", la gestion de l'infrastructure est également automatisée.

Gestion automatisée de l'infrastructure :

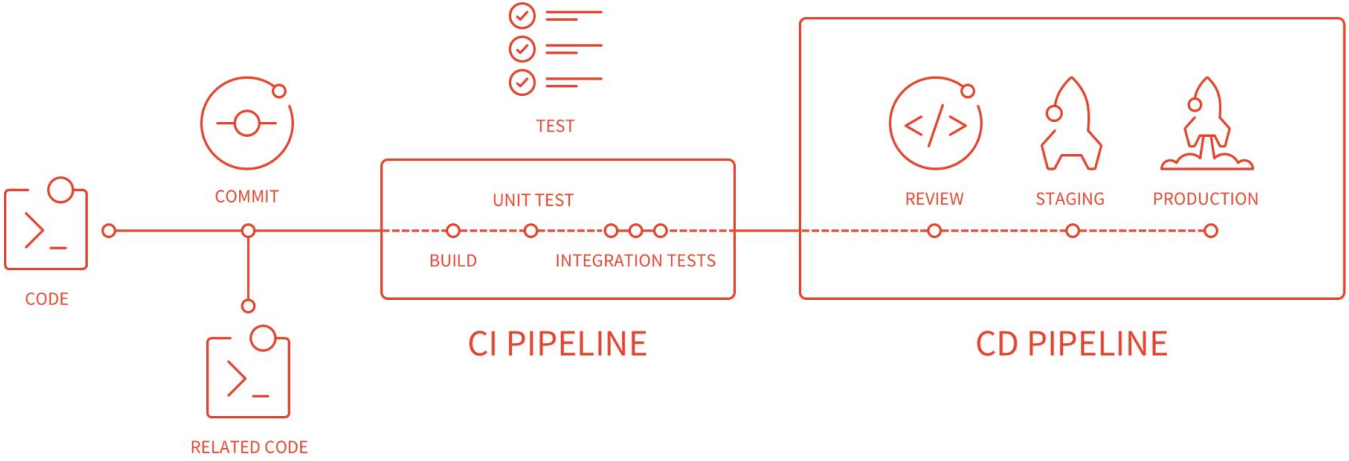
- La configuration de l'infrastructure d'exécution est définie sous forme de code.
  - Gérée par le gestionnaire de versions.
  - Cette configuration décrit les différentes étapes de la mise en place, telles que les logiciels à installer.
- Des environnements d'exécution doivent être créés à différentes étapes du processus de livraison continue.
  - Ces environnements sont nécessaires pour toutes les phases de test.
  - Ils sont également déployés pour la mise en production.

Infrastructure as Code

Avantages de l'automatisation de l'infrastructure en livraison continue :

- Réduction des erreurs.
- Consistance de l'environnement.
- Traçabilité des modifications et résolution d'erreurs facilitée.
- Liaison entre les modifications d'infrastructure et d'application.
- Déploiement rapide d'environnements.

MANIPULER LES CONTENEURS



ORCHESTRATION KUBERNETES

KUBERNETES : COMPOSANTS

Kubernetes est écrit en Go, compilé statiquement. Un ensemble de binaires sans dépendance

Faciles à conteneuriser et à packager

Peut se déployer uniquement avec des conteneurs sans dépendance d'OS

KUBERNETES : COMPOSANTS DU CONTROL PLANE

etcd: Base de données

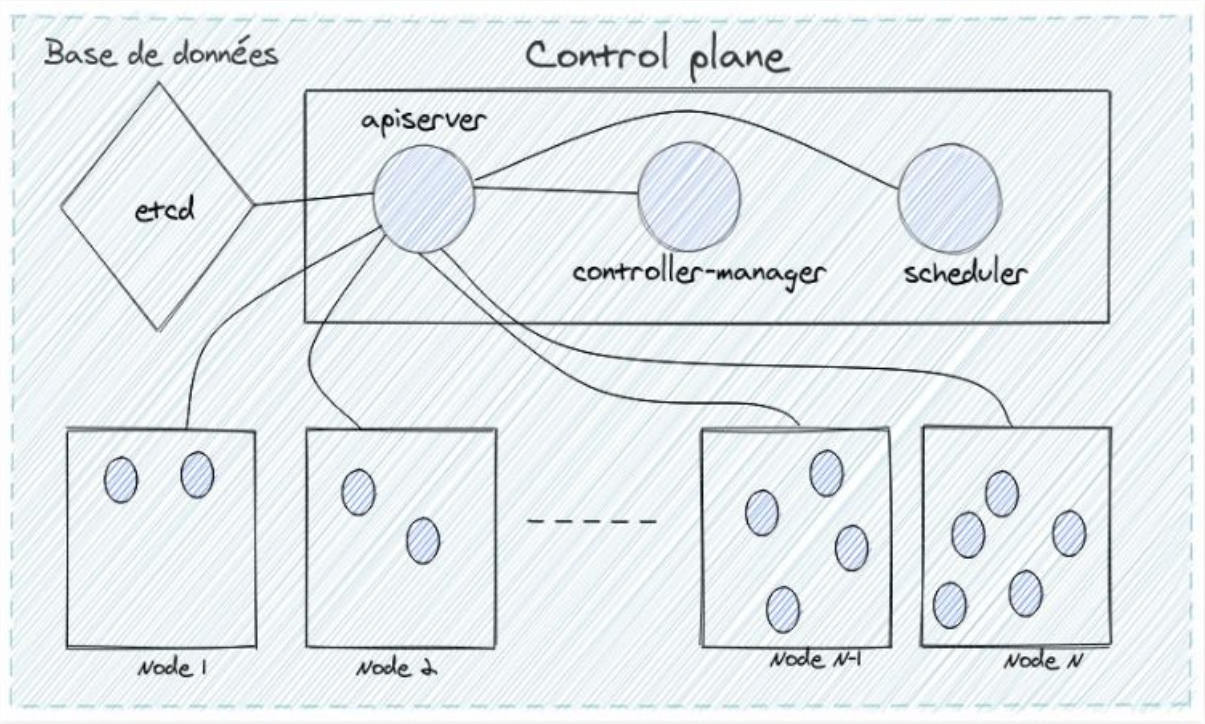
kube-apiserver : API server qui permet la configuration d'objets Kubernetes (Pod, Service, Deployment, etc.)

kube-proxy : Permet le forwarding TCP/UDP et le load balancing entre les services et les backends (Pods)

kube-scheduler : Implémente les fonctionnalités de scheduling

kube-controller-manager : Responsable de l'état du cluster, boucle infinie qui régule l'état du cluster afin d'atteindre un état désiré

KUBERNETES : COMPOSANTS DU CONTROL PLANE



KUBERNETES : ETCD

Base de données de type Clé/Valeur (Key Value Store)

Stocke l'état d'un cluster Kubernetes

Point sensible (stateful) d'un cluster Kubernetes

KUBERNETES : KUBE-API SERVER

Les configurations d'objets (Pods, Service, RC, etc.) se font via l'API server

Un point d'accès à l'état du cluster aux autres composants via une API REST

Tous les composants sont reliés avec l'API server

KUBERNETES : KUBE-SCHEDULER

Planifie les ressources sur le cluster

En fonction de règles implicites (CPU, RAM, stockage disponible, etc.)

En fonction de règles explicites (règles d'affinité et anti-affinité, labels, etc.)

KUBERNETES : KUBE-PROXY

Responsable de la publication des Services

Utilise iptables

Route les paquets à destination des conteneurs et réalise le load balancing TCP/UDP

KUBERNETES : KUBE-CONTROLLER-MANAGER

Boucle infinie qui contrôle l'état d'un cluster

Effectue des opérations pour atteindre un état donné De base dans Kubernetes : replication controller, endpoints controller, namespace controller et service accounts controller

KUBERNETES : AUTRES COMPOSANTS

kubelet : Service "agent" fonctionnant sur tous les nœuds et assure le fonctionnement des autres services

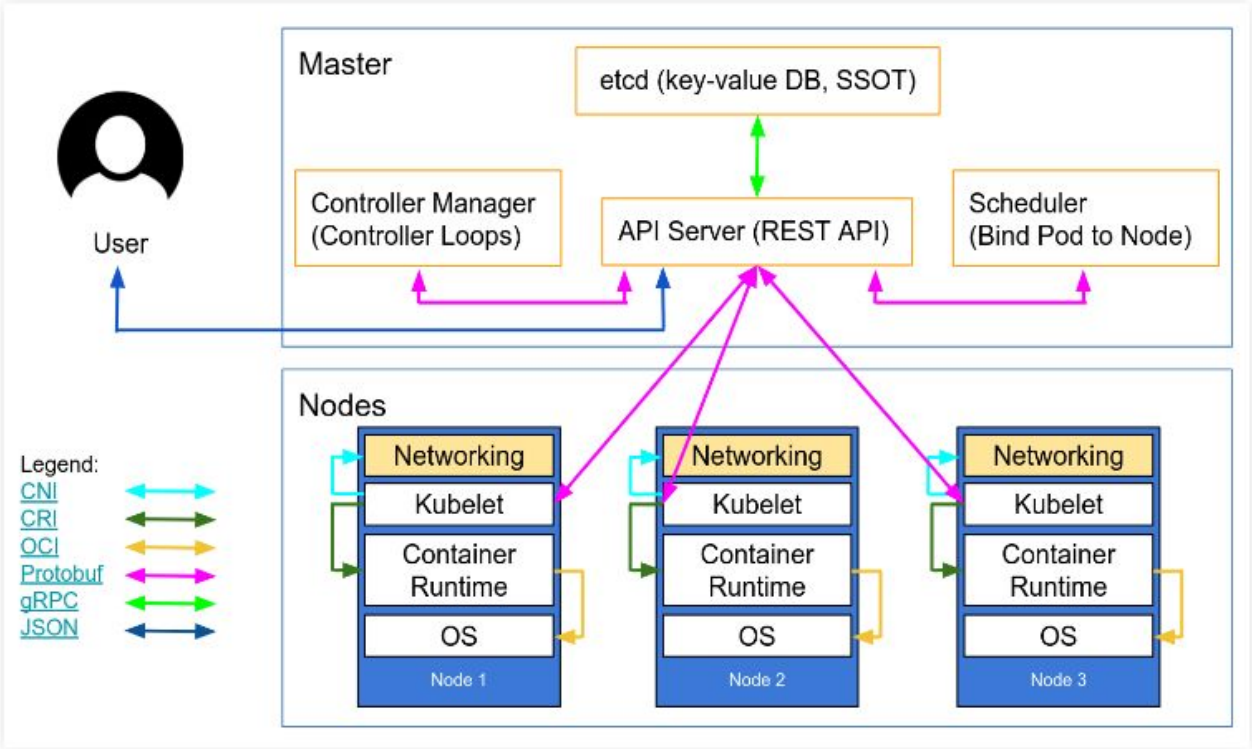
kubectl : Ligne de commande permettant de piloter un cluster Kubernetes

KUBERNETES : AUTRES COMPOSANTS

kubelet : Service "agent" fonctionnant sur tous les nœuds et assure le fonctionnement des autres services

kubectl : Ligne de commande permettant de piloter un cluster Kubernetes

KUBERNETES : COMPOSANTS DU CONTROL PLANE



KUBERNETES : CONCEPTS ET OBJETS

KUBERNETES : API RESOURCES

- Namespaces
- Pods
- Deployments
- DaemonSets
- StatefulSets
- Jobs
- Cronjobs

KUBERNETES : NAMESPACES

- Fournissent une séparation logique des ressources : Par utilisateurs
- Par projet / applications Autres...
- Les objets existent uniquement au sein d'un namespace donné
- Évitent la collision de nom d'objets

KUBERNETES : LABELS

Système de clé/valeur

Organisent les différents objets de Kubernetes (Pods, RC, Services, etc.) d'une manière cohérente qui reflète la structure de l'application

Corrèlent des éléments de Kubernetes : par exemple un service vers des Pods

KUBERNETES : LABELS

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

KUBERNETES : POD

Ensemble logique composé de un ou plusieurs conteneurs

Les conteneurs d'un pod fonctionnent ensemble (instanciation et destruction) et sont orchestrés sur un même hôte

Les conteneurs partagent certaines spécifications du Pod :

La stack IP (network namespace)

Inter-process communication (PID namespace) Volumes

C'est la plus petite et la plus simple unité dans Kubernetes

KUBERNETES : DEPLOYMENT

Permet d'assurer le fonctionnement d'un ensemble de Pods

Version, Update et Rollback

Anciennement appelés Replication Controllers

KUBERNETES : DEPLOYMENT

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

KUBERNETES : DAEMONSET

Assure que tous les noeuds exécutent une copie du pod

Ne connaît pas la notion de replicas.

Utilisé pour des besoins particuliers comme : l'exécution d'agents de collection de logs comme fluentd ou logstash

l'exécution de pilotes pour du matériel comme

nvidia-plugin

l'exécution d'agents de supervision comme NewRelic agent ou Prometheus node exporter

KUBERNETES : DAEMONSET

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
      - name: fluentd
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
```

KUBERNETES : STATEFULSET

Similaire au Deployment

Les pods possèdent des identifiants uniques. Chaque replica de pod est créé par ordre d'index Nécessite un Persistent Volume et un Storage Class.

Supprimer un StatefulSet ne supprime pas le PV associé

KUBERNETES : STATEFULSET

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
```

KUBERNETES : JOB

Crée des pods et s'assurent qu'un certain nombre d'entre eux se terminent avec succès.

Peut exécuter plusieurs pods en parallèle

Si un noeud du cluster est en panne, les pods sont reschedulés vers un autre noeud.

KUBERNETES : JOB

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure
```

KUBERNETES : CRONJOB

Un CronJob permet de lancer des Jobs de manière

planifiée.

la programmation des Jobs se définit au format

Cron le champ jobTemplate contient la définition de l'application à lancer comme Job

KUBERNETES : CRONJOB

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          -  name: pi
             image: perl
             command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
```

KUBERNETES : NETWORK PLUGINS

Kubernetes n'implémente pas de solution de gestion

de réseau par défaut

Le réseau est implémenté par des solutions tierces : Calico

KUBERNETES : SERVICES

Abstraction des Pods sous forme d'une IP virtuelle de

Service

Rendre un ensemble de Pods accessibles depuis l'extérieur ou l'intérieur du cluster

Load Balancing entre les Pods d'un même Service

Sélection des Pods faisant parti d'un Service grâce aux labels

KUBERNETES : SERVICES

Abstraction des Pods sous forme d'une IP virtuelle de

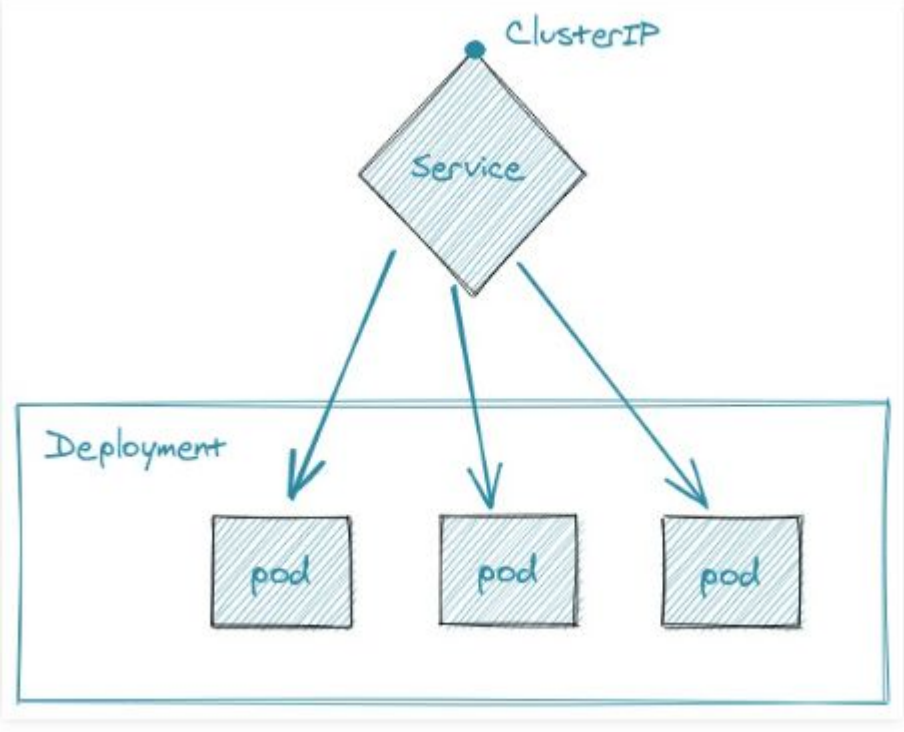
Service

Rendre un ensemble de Pods accessibles depuis l'extérieur ou l'intérieur du cluster

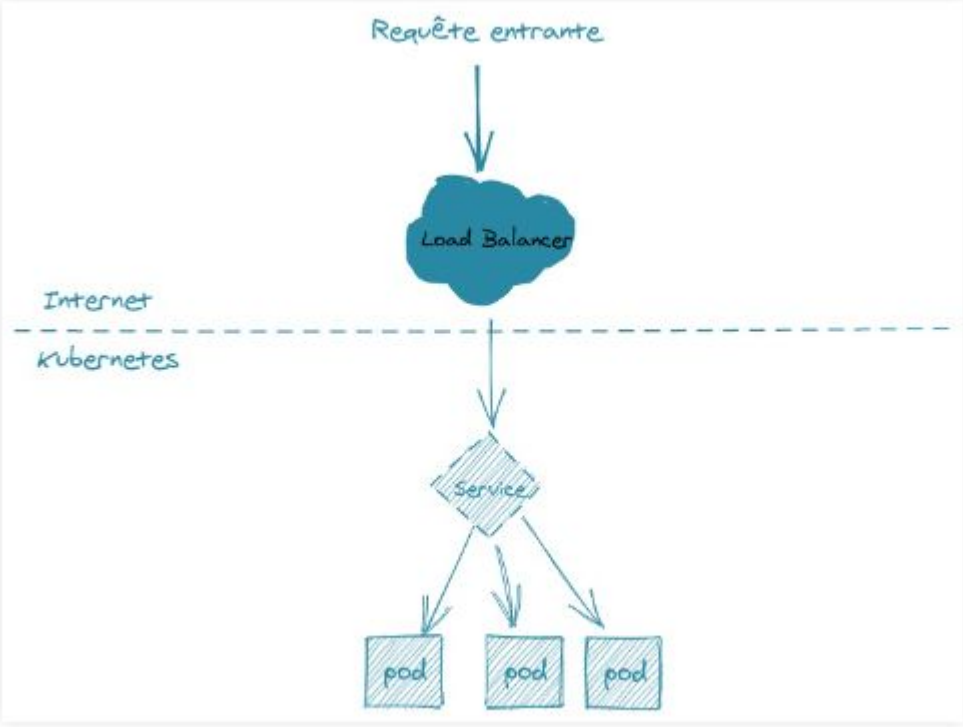
Load Balancing entre les Pods d'un même Service

Sélection des Pods faisant parti d'un Service grâce aux labels

KUBERNETES : SERVICE CLUSTER IP



KUBERNETES : SERVICE LB



KUBERNETES: INGRESS

L'objet Ingress permet d'exposer un Service à l'extérieur d'un cluster Kubernetes

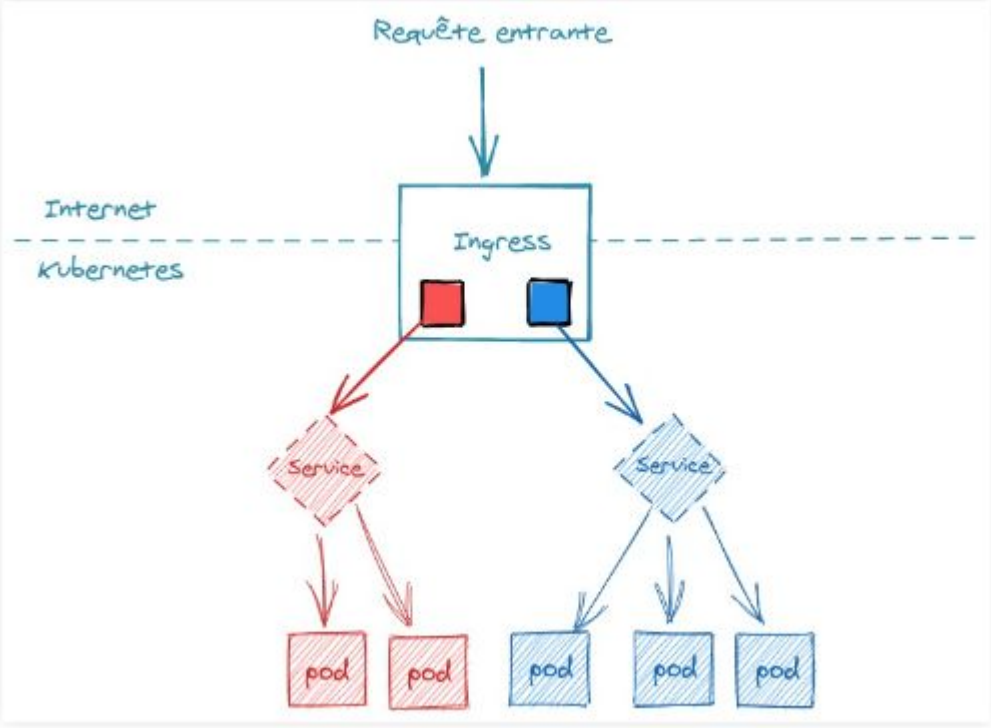
Il permet de fournir une URL visible permettant

d'accéder un Service Kubernetes

Il permet d'avoir des terminations TLS, de faire du

Load Balancing , etc...

KUBERNETES : INGRESS



KUBERNETES : VOLUMES

Fournir du stockage persistant aux pods

Fonctionnent de la même façon que les volumes Docker pour les volumes hôte :

EmptyDir ~= volumes docker

HostPath ~= volumes hôte

Support de multiples backend de stockage :

AWS : EBS

GlusterFS / NFS

Ceph

iSCSI

KUBERNETES : STORAGE CLASS

Permet de définir les différents types de stockage

disponibles

Utilisé par les Persistent Volumes

pour solliciter un espace de stockage au travers des

Persistent Volume Claims

KUBERNETES : CONFIGMAPS

Objet Kubernetes permettant de stocker séparément les fichiers de configuration

Il peut être créé d'un ensemble de valeurs ou d'un

fichier resource Kubernetes (YAML ou JSON)

Un ConfigMap peut sollicité par plusieurs pods

KUBERNETES : SECRETS

Objet Kubernetes de type secret utilisé pour stocker des informations sensibles comme les mots de passe, les tokens , les clés SSH...

Similaire à un ConfigMap , à la seule différence que le contenu des entrées présentes dans le champ data sont encodés en base64. Il est possible de directement créer un Secret spécifique à l'authentification sur une registry Docker

privée.

Il est possible de directement créer un Secret à partir d'un compte utilisateur et d'un mot de passe.

PODS RESOURCES : REQUEST ET LIMITS

Permettent de gérer l'allocation de ressources au sein d'un cluster

Par défaut, un pod/container sans request/limits est en best effort

Request: allocation minimum garantie (réservation)

Limit: allocation maximum (limite)

Se base sur le CPU et la RAM

PODS RESOURCES : REQUEST ET LIMITS

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
```

HORIZONTAL AUTOSCALING

Permet de scaler automatiquement le nombre de

pods d'un deployment

Métriques classiques (CPU/RAM): En fonction d'un % de la request CPU/RAM

Métriques custom (Applicative)

HORIZONTAL AUTOSCALING

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

SONDES : READINESS AND LIVENESS

Permettent à Kubernetes de sonder l'état d'un pod et d'agir en conséquence

2 types de sonde : Readiness et Liveness

3 manières de sonder :

TCP : ping TCP sur un port donné

HTTP: http GET sur une url donnée

Command: Exécute une commande dans le conteneur