FORMATION DOCKER

CONCERNANT CES SUPPORTS DE COURS

SUPPORTS DE COURS RÉALISÉS PAR PARTICULE

- ex Osones/Alterway
- Expertise Cloud Native
- Expertise DevOps
- Nos offres de formations: https://particule.io/trainings/
- Sources : https://github.com/particuleio/formations/
- HTML/PDF : https://particule.io/formations/

COPYRIGHT

LE CLOUD : VUE D'ENSEMBLE

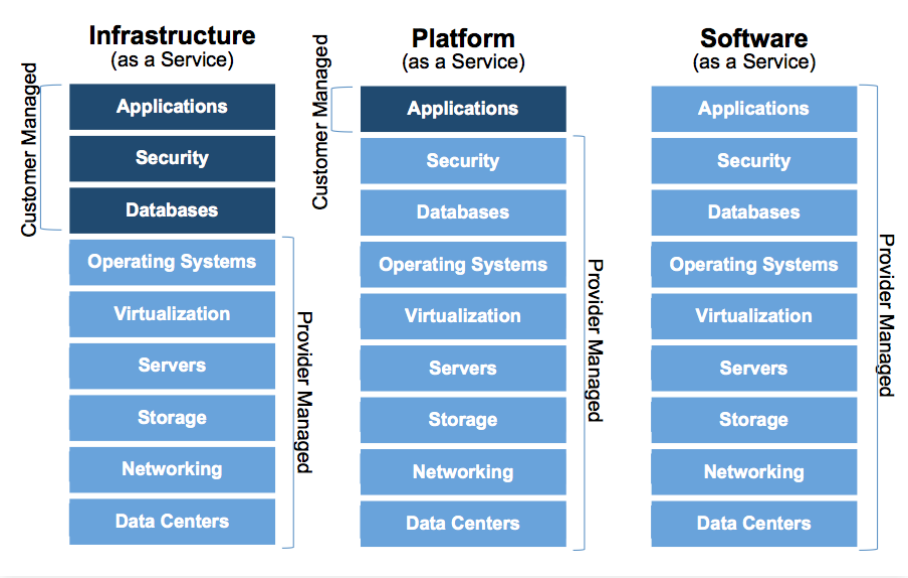
LE CLOUD, C'EST LARGE !

- Stockage/calcul distant (on oublie, cf. externalisation)
- Virtualisation++
- Abstraction du matériel (voire plus)
- Accès normalisé par des APIs
- Service et facturation à la demande
- Flexibilité, élasticité

WAAS : WHATEVER AS A SERVICE

- IaaS : Infrastructure as a Service
- PaaS : Platform as a Service
- SaaS : Software as a Service

LE CLOUD EN UN SCHÉMA



POURQUOI DU CLOUD ? CÔTÉ TECHNIQUE

- Abstraction des couches basses
- On peut tout programmer à son gré (API)
- Permet la mise en place d’architectures scalables

VIRTUALISATION DANS LE CLOUD

- Le cloud IaaS repose souvent sur la virtualisation
- Ressources compute : virtualisation
- Virtualisation complète : KVM, Xen
- Virtualisation conteneurs : OpenVZ, LXC, Docker, RKT

NOTIONS ET VOCABULAIRE IAAS

- L’instance est par définition éphémère
- Elle doit être utilisée comme ressource de calcul
- Séparer les données des instances

ORCHESTRATION DES RESSOURCES ?

- Groupement fonctionnel de ressources : micro services
- Infrastructure as Code : Définir toute une infrastructure dans un seul fichier texte de manière déclarative
- Scalabilité : passer à l'échelle son infrastructure en fonction de différentes métriques.

POSITIONNEMENT DES CONTENEURS DANS L'ÉCOSYSTÈME CLOUD ?

- Facilitent la mise en place de PaaS
- Fonctionnent sur du IaaS ou sur du bare-metal
- Simplifient la décomposition d'applications en micro services

LES CONTENEURS

DÉFINITION

- Les conteneurs fournissent un environnement isolé sur un système hôte, semblable à un chroot sous Linux ou une jail sous BSD, mais en proposant plus de fonctionnalités en matière d'isolation et de configuration. Ces fonctionnalités sont dépendantes du système hôte et notamment du kernel.

LE KERNEL LINUX

- Namespaces
- Cgroups (control groups)

LES NAMESPACES

MOUNT NAMESPACES (LINUX 2.4.19)

- Permet de créer un arbre des points de montage indépendants de celui du système hôte.

UTS NAMESPACES (LINUX 2.6.19)

- Unix Time Sharing : Permet à un conteneur de disposer de son propre nom de domaine et d'identité NIS sur laquelle certains protocoles tel que LDAP peuvent se baser.

IPC NAMESPACES (LINUX 2.6.19)

- Inter Process Communication : Permet d'isoler les bus de communication entre les processus d'un conteneur.

PID NAMESPACES (LINUX 2.6.24)

- Isole l’arbre d’exécution des processus et permet donc à chaque conteneur de disposer de son propre processus maître (PID 0) qui pourra ensuite exécuter et manager d'autres processus avec des droits illimités tout en étant un processus restreint au sein du système hôte.

USER NAMESPACES (LINUX 2.6.23-3.8)

- Permet l'isolation des utilisateurs et des groupes au sein d'un conteneur. Cela permet notamment de gérer des utilisateurs tels que l'UID 0 et GID 0, le root qui aurait des permissions absolues au sein d'un namespace mais pas au sein du système hôte.

NETWORK NAMESPACES (LINUX 2.6.29)

- Permet l'isolation des ressources associées au réseau, chaque namespace dispose de ses propres cartes réseaux, plan IP, table de routage, etc.

CGROUPS : CONTROL CROUPS

```
CGroup: /
           |--docker
           |   |--7a977a50f48f2970b6ede780d687e72c0416d9ab6e0b02030698c1633fdde95
           |   |--6807 nginx: master process ngin
           |   |   |--6847 nginx: worker proces
```

CGROUPS : LIMITATION DE RESSOURCES

- Limitation des ressources : des groupes peuvent être mis en place afin de ne pas dépasser une limite de mémoire.

CGROUPS : PRIORISATION

- Priorisation : certains groupes peuvent obtenir une plus grande part de ressources processeur ou de bande passante d'entrée-sortie.

CGROUPS : COMPTABILITÉ

- Comptabilité : permet de mesurer la quantité de ressources consommées par certains systèmes, en vue de leur facturation par exemple.

CGROUPS : ISOLATION

- Isolation : séparation par espace de nommage pour les groupes, afin qu'ils ne puissent pas voir les processus des autres, leurs connexions réseaux ou leurs fichiers.

CGROUPS : CONTRÔLE

- Contrôle : figer les groupes ou créer un point de sauvegarde et redémarrer.

DEUX PHILOSOPHIES DE CONTENEURS

- *Systeme* : simule une séquence de boot complète avec un init process ainsi que plusieurs processus (LXC, OpenVZ).
- *Process* : un conteneur exécute un ou plusieurs processus directement, en fonction de l'application conteneurisée (Docker, Rkt).

ENCORE PLUS “CLOUD” QU’UNE INSTANCE

- Partage du kernel
- Un seul processus par conteneur
- Le conteneur est encore plus éphémère que l’instance
- Le turnover des conteneurs est élevé : orchestration

CONTAINER RUNTIME

Permettent d'exécuter des conteneurs sur un système

- docker: historique
- containerd: implémentation de référence
- cri-o: implémentation Open Source développée par RedHat
- kata containers: Conteneurs dans des VMs

LES CONTENEURS: CONCLUSION

- Fonctionnalités offertes par le Kernel
- Les conteneurs engine fournissent des interfaces d'abstraction
- Plusieurs types de conteneurs pour différents besoins

LES CONCEPTS

UN ENSEMBLE DE CONCEPTS ET DE COMPOSANTS

- Layers
- Stockage
- Volumes
- Réseau
- Publication de ports
- Service Discovery

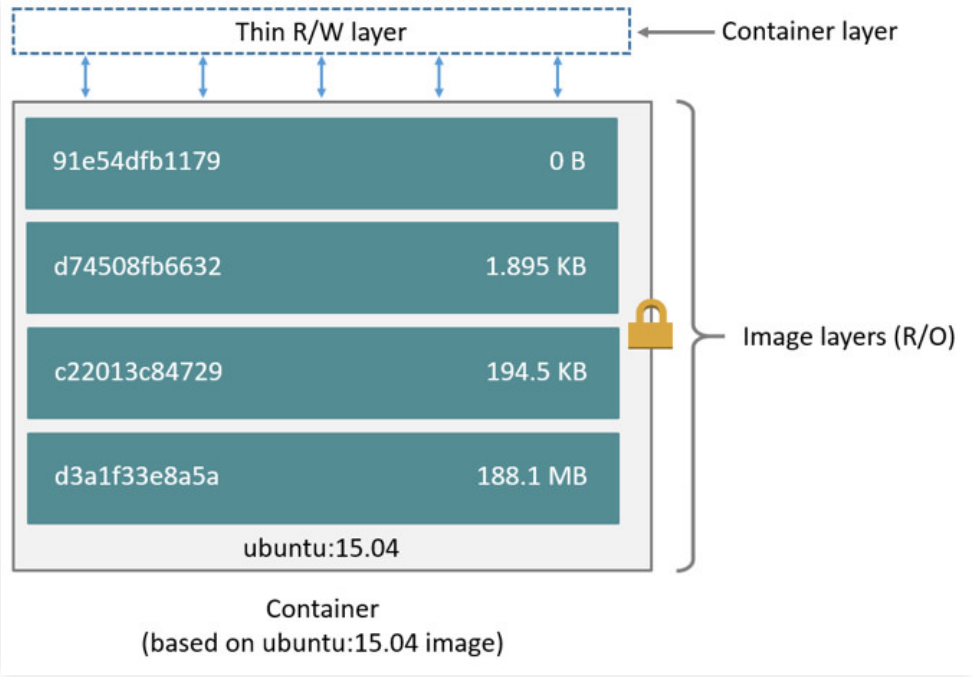
LAYERS

- Les conteneurs et leurs images sont décomposés en couches (layers)
- Les layers peuvent être réutilisés entre différents conteneurs
- Gestion optimisée de l'espace disque.

LAYERS : UNE IMAGE

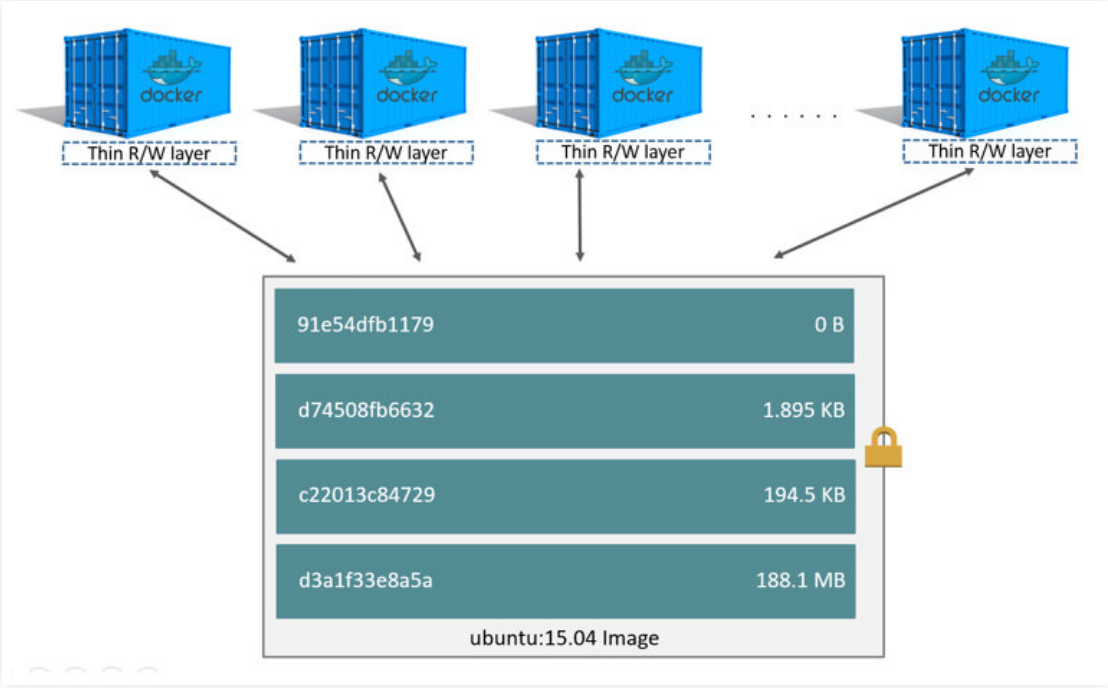
Une image se decompose en layers

LAYERS : UN CONTENEUR



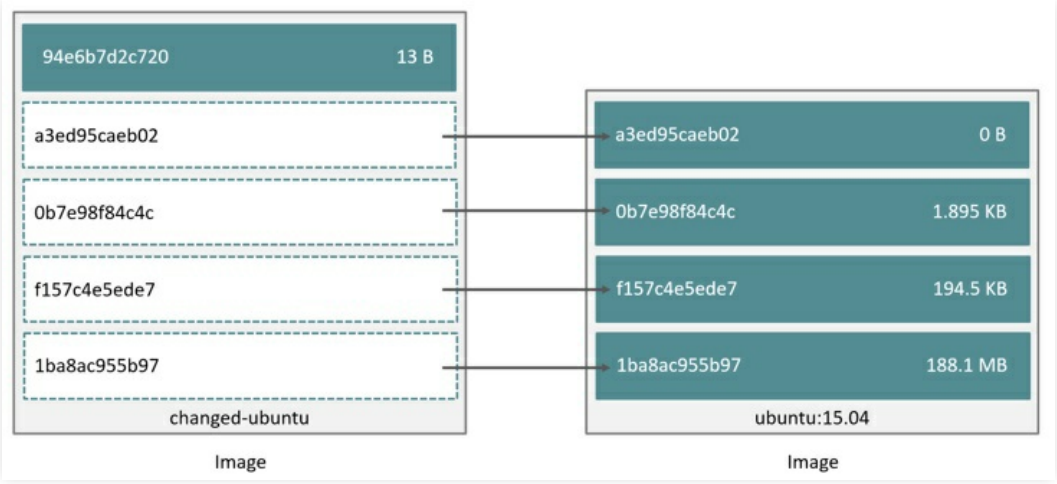
Une conteneur = une image + un layer r/w

LAYERS : PLUSIEURS CONTENEURS



Une image, plusieurs conteneurs

LAYERS : RÉPÉTITION DES LAYERS



Les layers sont indépendants de l'image

STOCKAGE

- Images Docker, données des conteneurs, volumes
- Multiples backends (extensibles via plugins):
  - AUFS
  - DeviceMapper
  - OverlayFS
  - NFS (via plugin Convoy)
  - GlusterFS (via plugin Convoy)

STOCKAGE : AUFS

- A unification filesystem
- Stable : performance écriture moyenne
- Non intégré dans le Kernel Linux (mainline)

STOCKAGE : DEVICE MAPPER

- Basé sur LVM
- Thin Provisionning et snapshot
- Intégré dans le Kernel Linux (mainline)
- Stable : performance écriture moyenne

STOCKAGE : OVERLAYFS

- Successeur de AUFS
- Performances accrues
- Relativement stable mais moins éprouvé que AUFS ou Device Mapper

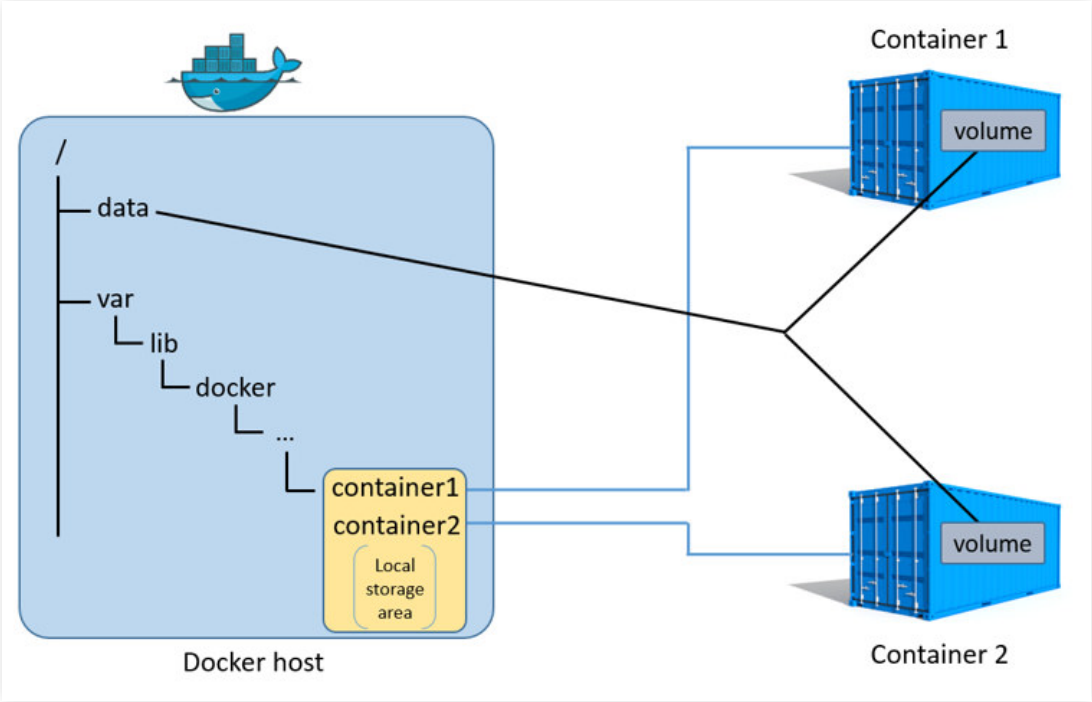
STOCKAGE : PLUGINS

- Étendre les drivers de stockages disponibles
- Utiliser des systèmes de fichier distribués (GlusterFS)
- Partager des volumes entre plusieurs hôtes docker

VOLUMES

- Assurent une persistance des données
- Indépendance du volume par rapport au conteneur et aux layers
- Deux types de volumes :
  - Conteneur : Les données sont stockées dans ce que l'on appelle un data conteneur
  - Hôte : Montage d'un dossier de l'hôte docker dans le conteneur
- Partage de volumes entre plusieurs conteneurs

VOLUMES : EXEMPLE



Un volume monté sur deux conteneurs

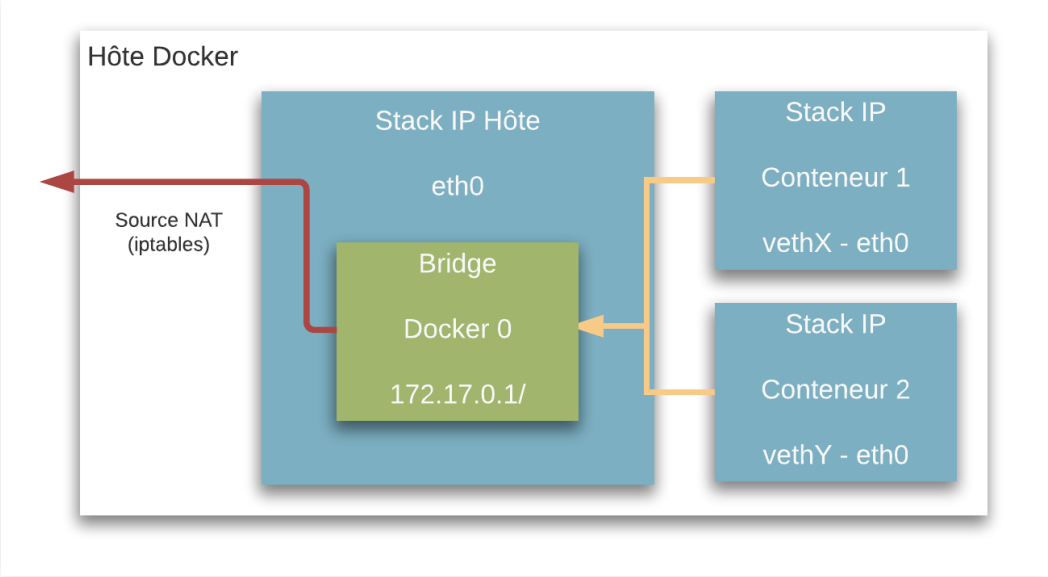
VOLUMES : PLUGINS

- Permettre le partage de volumes entre differents hôtes
- Fonctionnalité avancées : snapshot, migration, restauration
- Quelques exemples:
  - Convoy : multi-host et multi-backend (NFS, GlusterFS)
  - Flocker : migration de volumes dans un cluster

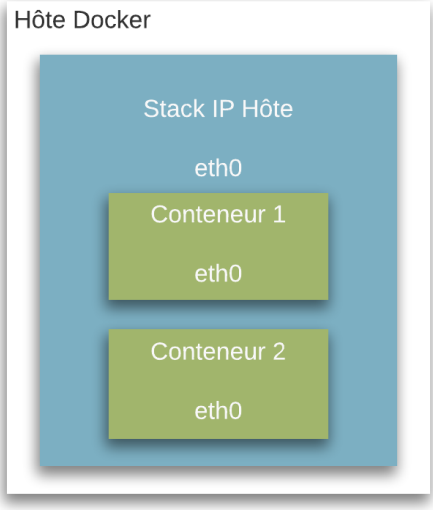
RÉSEAU : A LA BASE, PAS GRAND CHOSE...

```
NETWORK ID      NAME      DRIVER
42f7f9558b7a    bridge    bridge
6ebf7b150515    none      null
0509391a1fbd    host      host
```

RÉSEAU : BRIDGE



RÉSEAU : HOST


Hôte Docker
Stack IP Hôte
eth0
Conteneur 1
eth0
Conteneur 2
eth0

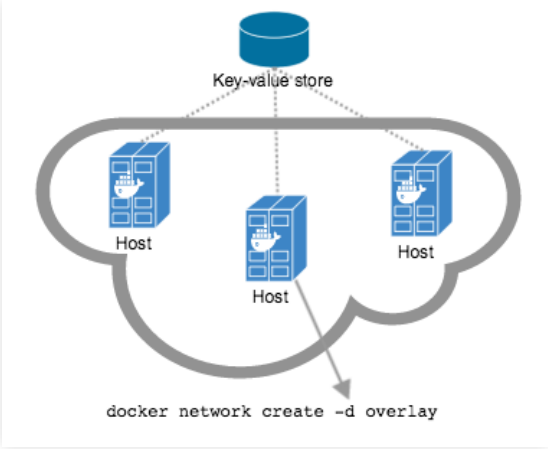
RÉSEAU : NONE

- Explicite

RÉSEAU : LES ÉVOLUTIONS

- Refactorisation des composants réseau (*libnetwork*)
- Système de plugins : multi host et multi backend (overlay network)
- Quelques exemples d'overlay :
  - Flannel : UDP et VXLAN
  - Weave : VXLAN

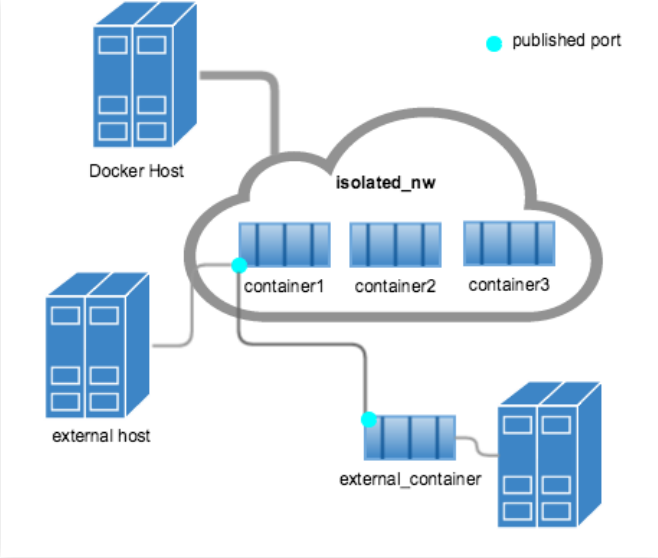
RÉSEAU : MULTIHOST OVERLAY



PUBLICATION DE PORTS

- Dans le cas d'un réseau diffèrent de l'hôte
- Les conteneurs ne sont pas accessible depuis l'extérieur
- Possibilité de publier des ports depuis l'hôte vers le conteneur (*iptables*)
- L'hôte sert de proxy au service

PUBLICATION DE PORTS



LINKS

- Les conteneurs d'un même réseau peuvent communiquer via IP
- Les liens permettent de lier deux conteneurs par nom
- Système de DNS rudimentaire (`/etc/hosts`)
- Remplacés par les *discovery services*

LES CONCEPTS: CONCLUSION

- Les composants de Docker sont modulaires
- Nombreuses possibilités offertes par défaut.
- Possibilité d'étendre les fonctionnalités via des plugins

BUILD, SHIP AND RUN !

BUILD

LE CONTENEUR ET SON IMAGE

- Flexibilité et élasticité
- Format standard de facto
- Instanciation illimitée

CONSTRUCTION D'UNE IMAGE

- Possibilité de construire son image à la main (long et source d'erreurs)
- Suivi de version et construction d'images de manière automatisée
- Utilisation de *Dockerfile* afin de garantir l'idempotence des images

DOCKERFILE

- Suite d'instruction qui définit une image
- Permet de vérifier le contenu d'une image

```
FROM alpine:3.4
MAINTAINER Particule <admin@particule.io>
RUN apk -U add nginx
EXPOSE 80 443
CMD ["nginx"]
```

DOCKERFILE : PRINCIPALES INSTRUCTIONS

- `FROM` : baseimage utilisée
- `RUN` : Commandes effectuées lors du build de l'image
- `EXPOSE` : Ports exposées lors du run (si `-P` est précisé)
- `ENV` : Variables d'environnement du conteneur à l'instanciation
- `CMD` : Commande unique lancée par le conteneur
- `ENTRYPOINT` : "Préfixe" de la commande unique lancée par le conteneur

DOCKERFILE : BEST PRACTICES

- Bien choisir sa baseimage
- Chaque commande Dockerfile génère un nouveau layer
- Comptez vos layers !

DOCKERFILE : BAD LAYERING

```
RUN apk --update add \
    git \
    tzdata \
    python \
    unrar \
    zip \
    libxslt \
    py-pip \

RUN rm -rf /var/cache/apk/*

VOLUME /config /downloads

EXPOSE 8081

CMD ["--datadir=/config", "--nolaunch"]

ENTRYPOINT ["/usr/bin/env","python2","/sickrage/SickBeard.py"]
```

DOCKERFILE : GOOD LAYERING

```
RUN apk --update add \
    git \
    tzdata \
    python \
    unrar \
    zip \
    libxslt \
    py-pip \
    && rm -rf /var/cache/apk/*

VOLUME /config /downloads

EXPOSE 8081

CMD ["--datadir=/config", "--nolaunch"]

ENTRYPOINT ["/usr/bin/env","python2","/sickrage/SickBeard.py"]
```

DOCKERFILE : DOCKERHUB

- Build automatisée d'images Docker
- Intégration GitHub / DockerHub
- Plateforme de stockage et de distribution d'images Docker

SHIP

SHIP : LES CONTENEURS SONT MANIPULABLES

- Sauvegarder un conteneur :

```
docker commit mon-conteneur backup/mon-conteneur
```

```
docker run -it backup/mon-conteneur
```

- Exporter une image :

```
docker save -o mon-image.tar backup/mon-conteneur
```

- Importer un conteneur :

```
docker import mon-image.tar backup/mon-conteneur
```

SHIP : DOCKER REGISTRY

- DockerHub n’est qu’au Docker registry ce que GitHub est à git
- Pull and Push
- Image officielle : `registry`

RUN

RUN : LANCER UN CONTENEUR

- docker run
- `-d` (detach)
- `-i` (interactive)
- `-t` (pseudo tty)

RUN : BEAUCOUP D'OPTIONS...

- `-v` /directory/host:/directory/container
- `-p` portHost:portContainer
- `-P`
- `-e` “VARIABLE=valeur”
- `--restart=always`
- `--name=mon-conteneur`

RUN : ...DONT CERTAINES UN PEU DANGEREUSES

- `--privileged` (Accès à tous les devices)
- `--pid=host` (Accès aux PID de l'host)
- `--net=host` (Accès à la stack IP de l'host)

RUN : SE “CONNECTER” À UN CONTENEUR

- docker exec
- docker attach

RUN : DÉTRUIRE UN CONTENEUR

- docker kill (SIGKILL)
- docker stop (SIGTERM puis SIGKILL)
- docker rm (détruit complètement)

BUILD, SHIP AND RUN : CONCLUSIONS

- Écosystème de gestion d'images
- Construction automatisée d'images
- Contrôle au niveau conteneurs

ÉCOSYSTÈME DOCKER

DOCKER INC.

- Docker Inc != Docker Project
- Docker Inc est le principal développeur du Docker Engine, Compose, Machine, Kitematic, Swarm etc.
- Ces projets restent OpenSource et les contributions sont possibles

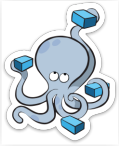
OCI

- Créé sous la Linux Fondation
- But : Créer des standards OpenSource concernant la manière de "runner" et le format des conteneurs
- Non lié à des produits commerciaux
- Non lié à des orchestrateurs ou des clients particuliers
- runC a été donné par Docker à l'OCI comme base pour leurs travaux

LES AUTRES PRODUITS DOCKER

DOCKER-COMPOSE

- Concept de stack
- Infrastructure as a code
- Scalabilité



DOCKER-COMPOSE

docker-compose.yml

```
nginx:
  image: nginx
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - "/srv/nginx/etc/sites-enabled:/etc/nginx/sites-enabled"
    - "/srv/nginx/etc/certs:/etc/nginx/certs"
    - "/srv/nginx/etc/log:/var/log/nginx"
    - "/srv/nginx/data:/var/www"
```

DOCKER-MACHINE

- "Metal" as a Service
- Provisionne des hôtes Docker
- Abstraction du Cloud Provider

DOCKER SWARM

- Clustering : Mutualisation d'hôtes Docker
- Orchestration : placement intelligent des conteneurs au sein du cluster

AUTOUR DE DOCKER

- Plugins : étendre les fonctionnalités notamment réseau et volumes



  - Systèmes de gestion de conteneurs (COE)
  - Docker as a Service :
    - Docker Cloud
    - Tutum

ÉCOSYSTÈME DOCKER : CONCLUSION

- Le projet Docker est Open Source et n'appartient plus à Docker
- Des outils permettent d'étendre les fonctionnalités de Docker
- Docker Inc. construit des offres commerciales autour de Docker

DOCKER HOSTS

LES DISTRIBUTIONS TRADITIONNELLES

- Debian, CentOS, Ubuntu
- Supportent tous Docker
- Paquets DEB et RPM disponibles

UN PEU "TOO MUCH" NON ?

- Paquets inutiles ou out of date
- Services par défaut/inutiles alourdissent les distributions
- Cycle de release figé

OS ORIENTÉS CONTENEURS

- Faire tourner un conteneur engine
- Optimisé pour les conteneurs : pas de services superflus
- Fonctionnalités avancées liées aux conteneurs (clustering, network, etc.)
- Sécurité accrue de part le minimalisme

OS ORIENTÉS CONTENEURS : EXEMPLES

- CoreOS (CoreOS)
- Atomic (Red Hat)
- RancherOS (Rancher)
- Photon (VMware)

COREOS : PHILOSOPHIE

- Trois "channels" : stable, beta, alpha
- Dual root : facilité de mise à jour
- Système de fichier en read only
- Pas de gestionnaire de paquets : tout est conteneurisé
- Forte intégration de *systemd*

COREOS : FONCTIONNALITÉS ORIENTÉES CONTENEURS

- Inclus :
  - Docker
  - Rkt
  - Etcd (base de données clé/valeur)
  - Flannel (overlay network)
  - Kubernetes Kubelet
- Permet nativement d'avoir un cluster complet
- Stable et éprouvé en production
- Idéal pour faire tourner Kubernetes (Tectonic)

COREOS : ETCD

- Système de stockage simple : clé = valeur
- Hautement disponible (quorum)
- Accessible via API

COREOS : FLANNEL

- Communication multi-hosts
- UDP ou VXLAN
- S'appuie sur un système clé/valeur comme etcd

RANCHEROS : PHILOSOPHIE

- Docker et juste Docker : Docker est le process de PID 1)
- Docker in Docker : Daemon User qui tourne dans le Daemon System
- Pas de processus tiers, pas d'init, juste Docker
- Encore en beta

FEDORA ATOMIC : PHILOSOPHIE

- Équivalent à CoreOS mais basée sur Fedora
- Utilise *systemd*
- Update Atomic (incrémentielles pour rollback)

PROJECT PHOTON

- Développé par VMware mais Open Source
- Optimisé pour vSphere
- Supporte Docker, Rkt et Pivotal Garden (Cloud Foundry)

DOCKER HOSTS : CONCLUSION

- Répond à un besoin différent des distributions classiques
- Fourni des outils et une logique adaptée aux environnements full conteneurs
- Sécurité accrue (mise à jour)

DOCKER EN PRODUCTION

OÙ DÉPLOYER ?

- Cloud public:
  - GCE
  - AWS

- Cloud privé: OpenStack

- Bare Metal

COMMENT ?

- Utilisation de COE (Container Orchestration Engine)
- Utilisation d'infrastructure as code
- Utilisation d'un *Discovery Service*

CONTAINER ORCHESTRATION ENGINE

- Gestion du cycle de vie des conteneurs/applications
- Abstraction des hôtes et des cloud providers (clustering)
- Scheduling en fonction des besoins de l'application
- Quelques exemples:
  - Docker Swarm
  - Rancher
  - Mesos
  - Kubernetes

INFRASTRUCTURE AS A CODE

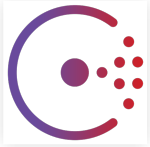
- Version Control System
- Intégration / Déploiement Continue
- Outils de templating (Heat, Terraform, Cloudformation)

DISCOVERY SERVICE

- La nature éphémère des conteneurs empêche toute configuration manuelle
- Connaître de façon automatique l'état de ses conteneurs à tout instant
- Fournir un endpoint fixe à une application susceptible de bouger au sein d'un cluster

CONSUL

- Combinaison de plusieurs services : KV Store, DNS, HTTP
- Failure detection
- Datacenter aware
- Github



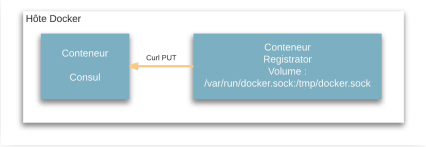
CONSUL

Exemple :

```
$ curl -X PUT -d 'docker' http://localhost:8500/v1/kv/container/key1
true
$ curl http://localhost:8500/v1/kv/container/key1 | jq .
[
  {
    "CreateIndex": 5,
    "ModifyIndex": 5,
    "LockIndex": 0,
    "Key": "container/key1",
    "Flags": 0,
    "Value": "ZG9ja2Vy"
  }
]
$ echo ZG9ja2Vy | base64 -d
docker
```

CONSUL

- L'enregistrement des nouveaux conteneurs peut être automatisé
- *Registrator* est un process écoutant le daemon Docker et enregistrant les évènements



RANCHER

- Permet de provisionner et mutualiser des hôtes Docker sur différents Cloud Provider
- Fournit des fonctionnalité de COE :
  - Cattle : COE développé par Rancher
  - Kubernetes : COE développé par Google
- Bon compromis entre simplicité et fonctionnalités comparé à Mesos ou Kubernetes
- Encore jeune, adapté aux environnement de taille moyenne (problèmes de passage à l'échelle)

APACHE MESOS / MARATHON

- Mesos : Gestion et orchestration de systèmes distribués
- A la base orienté Big Data (Hadoop, Elasticsearch...) et étendu aux conteneurs via Marathon
- Marathon : Scheduler pour Apache Mesos orienté conteneur
- Multi Cloud/Datacenter
- Adapté aux gros environnements, éprouvé jusque 10000 nodes

KUBERNETES

- COE développé par Google, devenu open source en 2014
- Utilisé par Google pour la gestion de leurs conteneurs
- Adapté à tout type d'environnements
- Devenu populaire en très peu de temps

QUELQUES AUTRES

- Hashicorp Nomad
- Amazon ECS
- Docker Cloud
- Docker UCP (Universal Control Plane)

DOCKER EN PRODUCTION : CONCLUSION

MONITORER UNE INFRASTRUCTURE DOCKER

QUOI MONITORER ?

- L'infrastructure
- Les conteneurs

MONITORER SON INFRASTRUCTURE

- Problématique classique

Monitorer des serveur est une problématique résolu depuis de nombreuses années par des outils devenus des standards :

- Nagios
- Shinken
- Centreon
- Icinga

INTÉRÊT ?

- Un des principes du cloud computing est d'abstraire les couches basses
- Docker accentue cette pratique
- Est-ce intéressant de monitorer son infra ?

Oui bien sûr ;)

MONITORER SES CONTENEURS

- Monitorer les services fournis par les conteneurs
- Monitorer l'état d'un cluster
- Monitorer un conteneur spécifique

PROBLÉMATIQUES DES CONTENEURS

- Les conteneurs sont des boîtes noires
- Les conteneurs sont dynamiques
- La scalabilité induite par les conteneurs devient un problème

MONITORER LES SERVICES

- La problématique est différente pour chaque service
- Concernant les web services (grande majorité), le temps de réponse du service et la disponibilité du service sont de bons indicatifs
- Problème adressé par certains services discovery pour conserver une vision du système cohérente (ex : Consul

MONITORER L'ÉTAT D'UN CLUSTER

- Dépends grandement de la technologie employée
- Le cluster se situe t-il au niveau de l'host Docker ou des conteneurs eux mêmes ?

MONITORER, OK MAIS DEPUIS OÙ ?

- Commandes exécutées dans le container
- Agents à l'intérieur du conteneur
- Agents à l'extérieur du conteneur

LES OUTILS DE MONITORING

- Docker stats
- CAdvisor
- Datadog
- Sysdig
- Prometheus

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

- Exemple de label :

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

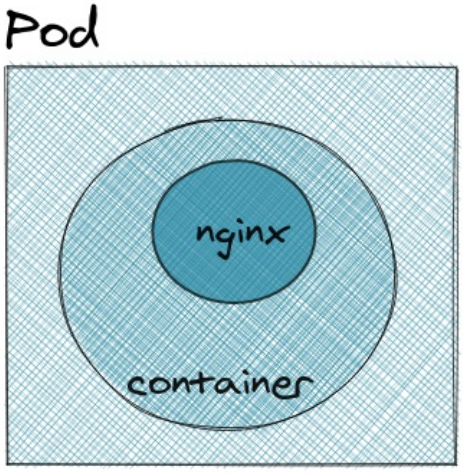
KUBERNETES : POD

- 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 : POD

Les Pods sont définis en YAML comme les fichiers `docker-compose` :

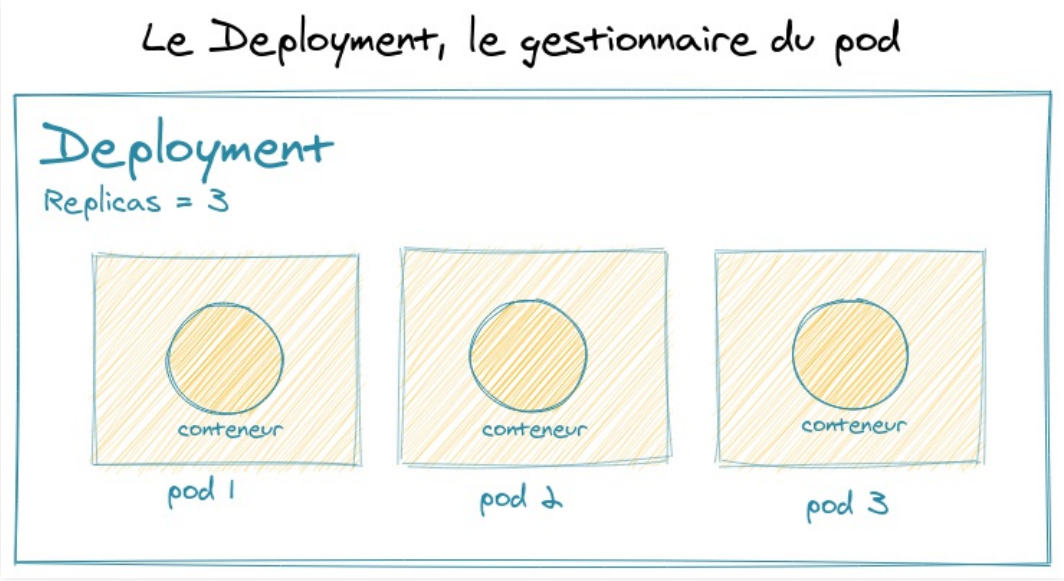
```
---
apiVersion: v1
kind: Pod
metadata:
  name: mon_pod
spec:
  containers:
    - name: conteneur
      image: nginx:latest
```



KUBERNETES : DEPLOYMENT

- Permet d'assurer le fonctionnement d'un ensemble de Pods
- Version, Update et Rollback
- Anciennement appelés Replication Controllers

KUBERNETES : DEPLOYMENT



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

```
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: CRON JOB

- 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

```
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 : NETWORKING

KUBERNETES : NETWORK PLUGINS

- Kubernetes n'implémente pas de solution de gestion de réseau par défaut
- Le réseau est implémenté par des solutions tierces :
- Calico : IPinIP + BGP
- Cilium : eBPF
- Weave : VXLAN
- Bien d'autres

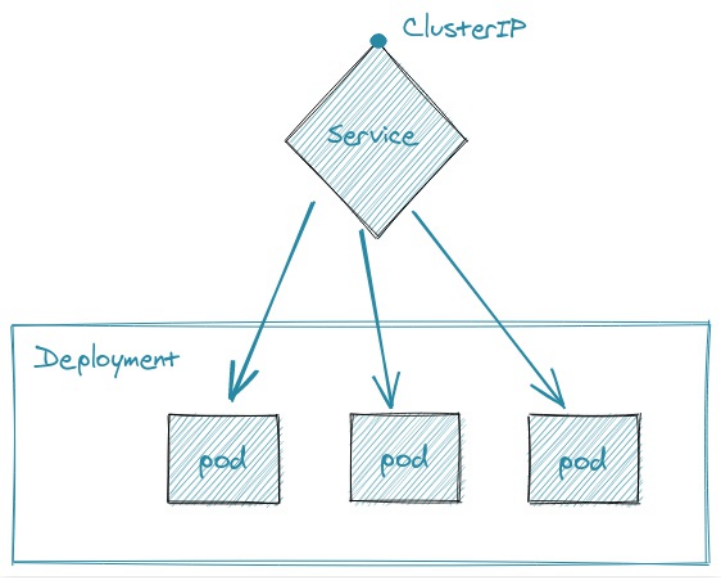
KUBERNETES : CNI

- Container Network Interface
- Projet dans la CNCF
- Standard pour la gestion du réseau en environnement conteneurisé
- Les solutions précédentes s'appuient sur CNI

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



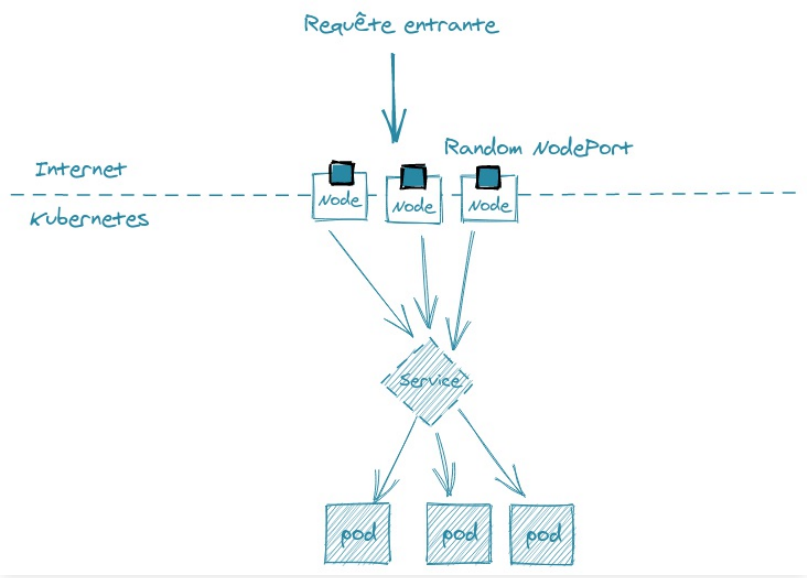
KUBERNETES : SERVICES CLUSTERIP

- Exemple de Service (on remarque la sélection sur le label et le mode d'exposition):

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    app: guestbook
```

KUBERNETES : SERVICE NODEPORT

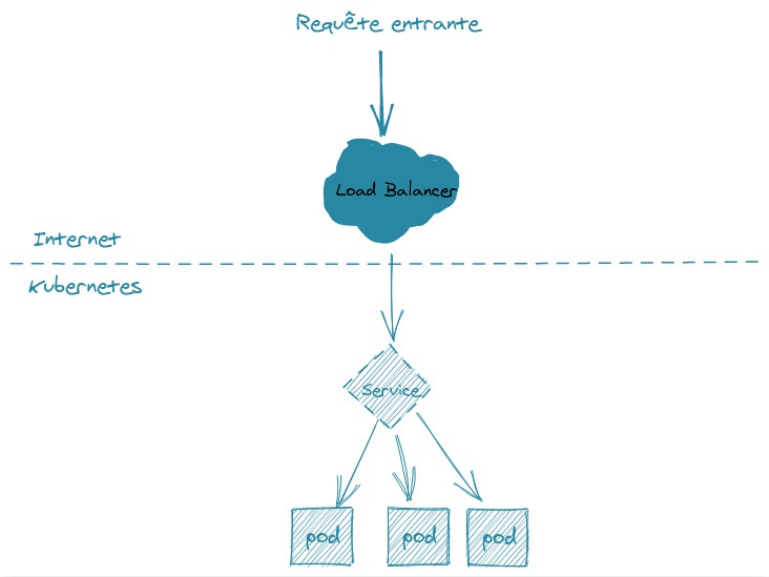
`NodePort` : chaque noeud du cluster ouvre un port statique et redirige le trafic vers le port indiqué



KUBERNETES : SERVICE LOADBALANCER

- `LoadBalancer` : expose le Service en externe en utilisant le loadbalancer d'un cloud provider
  - AWS ELB/ALB/NLB
  - GCP LoadBalancer
  - Azure Balancer
  - OpenStack Octavia

KUBERNETES : SERVICE LOADBALANCER



KUBERNETES : SERVICES

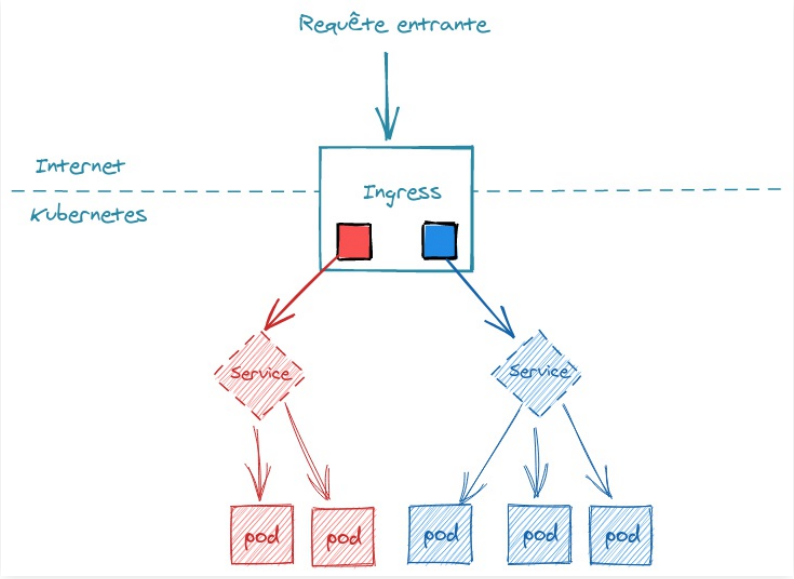
Il est aussi possible de mapper un Service avec un nom de domaine en spécifiant le paramètre `spec.externalName`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

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 : INGRESS

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: particule
spec:
  rules:
  - host: blog.particule.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
```

KUBERNETES : INGRESS CONTROLLER

Pour utiliser un `Ingress`, il faut un Ingress Controller. Un `Ingress` permet de configurer une règle de reverse proxy sur l'Ingress Controller.

- Nginx Controller : https://github.com/kubernetes/ingress-nginx
- Traefik : https://github.com/containous/traefik
- Istio : https://github.com/istio/istio
- Linkerd : https://github.com/linkerd/linkerd
- Contour : https://www.github.com/heptio/contour/

KUBERNETES : ARCHITECTURE

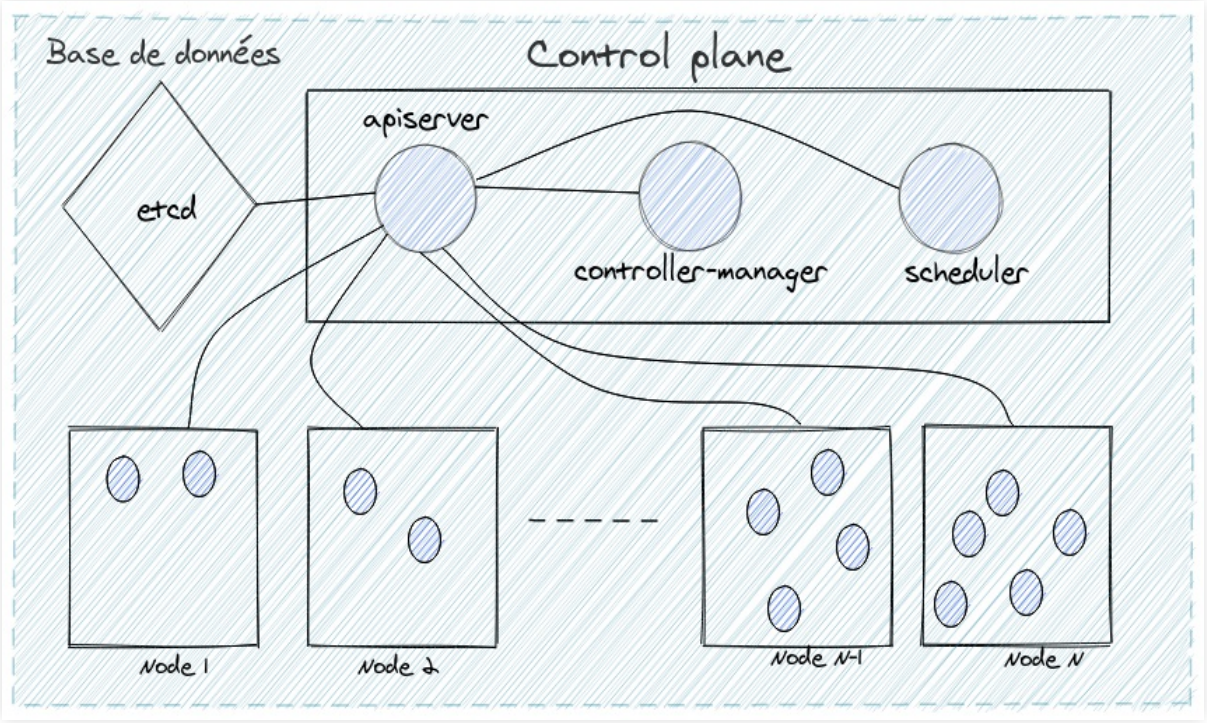
KUBERNETES : COMPOSANTS

- 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

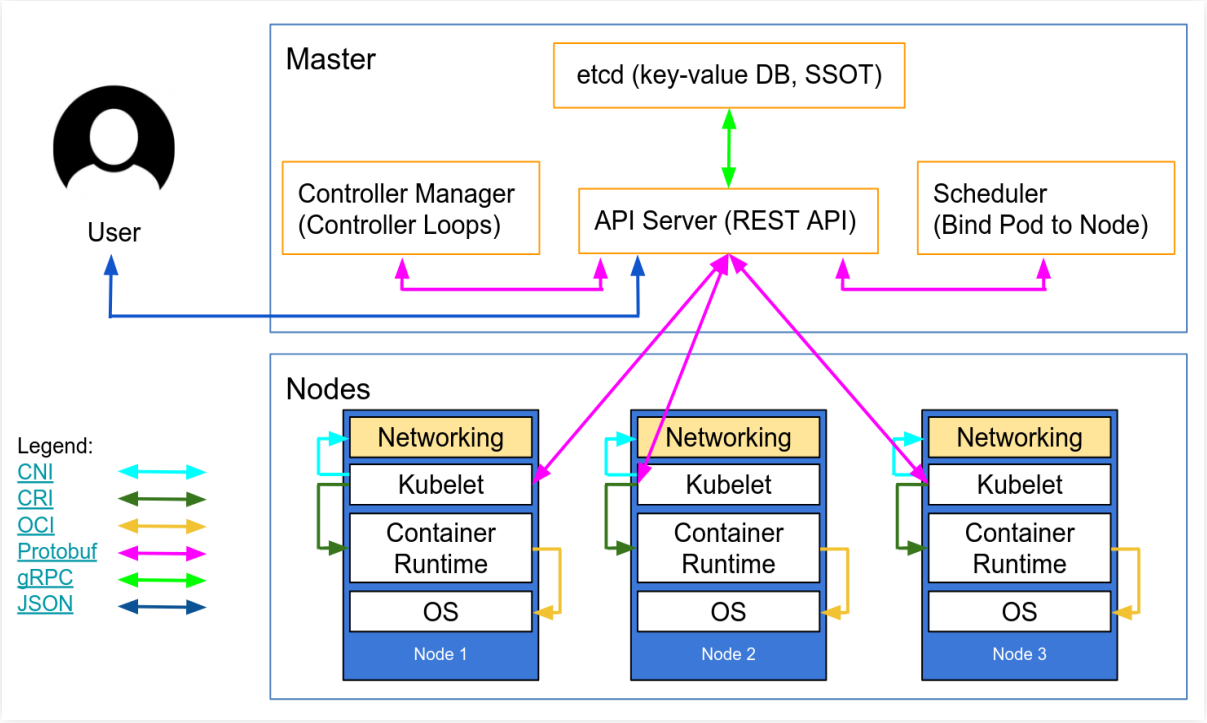
KUBERNETES : COMPOSANTS DU CONTROL PLANE

- 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 : 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
- Projet intégré à la CNCF

KUBERNETES : KUBE-APISERVER

- 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 à 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 serviceaccounts 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

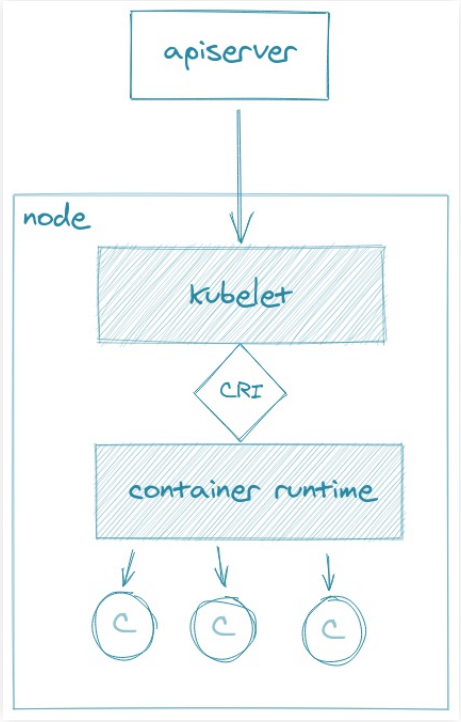
KUBERNETES : KUBELET

- Service principal de Kubernetes
- Permet à Kubernetes de s'auto configurer :
  - Surveille un dossier contenant les *manifests* (fichiers YAML des différents composant de Kubernetes).
  - Applique les modifications si besoin (upgrade, rollback).
- Surveille l'état des services du cluster via l'API server (*kube-apiserver*).

KUBERNETES : KUBELET

- Assure la communication entre les nodes et l'apiserver
- En charge de créer les conteneurs au travers de l'interface Container Runtime Interface (CRI)
- Peut fonctionner avec différentes container runtimes

KUBERNETES : KUBELET



KUBERNETES: NETWORK

Kubernetes n'implémente pas de solution réseau par défaut, mais s'appuie sur des solutions tierces qui implémentent les fonctionnalités suivantes :

- Chaque pods reçoit sa propre adresse IP
- Les pods peuvent communiquer directement sans NAT

OPENSHIFT : INTRODUCTION

OPENSHIFT

- Solution de PaaS développée par Red Hat
- Deux version :
  - Open Source : OpenShift Origin
  - Entreprise : OpenShift Container Platform
- Se base sur Kubernetes en tant qu'orchestrateur de conteneurs

OPENSHIFT VS KUBERNETES : DIFFÉRENCES ET AJOUTS

- Pas d'Ingress : Router
- Build et Images Stream : Création d'images et pipeline de CI/CD
- Templates : Permet de definir et templatiser facilement un ensemble d'Objet OpenShift
- OpenShift SDN ou Nuage Network pour le réseau

OPENSHIFT : ROUTER

- Quasiment identique à Ingress mais implémentés avant Kubernetes
- Fournissent les même fonctionnalités
- Deux implementations :
  - HAProxy
  - F5 Big IP

OPENSHIFT : BUILD

- Permet de construire et reproduire des images d'application
- Docker builds : via Dockerfile
- Source-to-Image (S2I) builds : système de build propre à OpenShift qui produit une image Docker d'une application depuis les source
- Pipeline builds : via Jenkins

OPENSHIFT : IMAGE STREAM

- Similaires à un dépôt DockerHub
- Rassemble des images similaires identifiées par des tags
- Permet de garder un historique des différentes versions

OPENSHIFT : CONCLUSION

CONCLUSION