Formation :

Conteneurs et CaaS : Docker, Kubernetes ...

Formateur

Brahim Hamdi
brahim.hamdi.consult@gmail.com

Juin 2024

Plan

- Présentation du formateur
- Le plan de formation
- Objectifs de la formation
- Public concerné
- Connaissances requises
- Références bibliographiques

Présentation du formateur

Brahim HAMDI

- Consultant/Formateur
- Expert DevOps & Cloud

Plan de la formation

- Le cloud, vue d’ensemble
- Les conteneurs
- Docker
- Outils d’orchestration
- Le Container as a Service
- Évolution vers le CaaS

Objectifs pédagogiques

- Présenter les fondamentaux et les technologies de containers et les raisons de leur émergence grâce à Docker
- Identifier les acteurs majeurs et les usages actuels
- Mettre en œuvre des solutions d'orchestration avec notamment Kubernetes
- Gérer les contours du nouveau modèle Containers As A Service (CaaS).

Public concerné

- Directeurs de systèmes d'informations,
- Architectes,
- Ingénieurs système et réseau,
- Chefs de projets,
- Administrateurs seniors,
- Développeurs
- ...

Connaissance requise

- Aucune

Références bibliographiques

- https://docs.docker.com/
- https://kubernetes.io/docs/
- https://docs.aws.amazon.com/
- https://learn.microsoft.com/fr-fr/azure/
- https://cloud.google.com/docs?hl=fr

Le cloud, vue d’ensemble

- Le cloud
- Les caractéristiques du cloud
- Les modèles de service
- Les modèles de déploiement
- Démonstration : AWS – EC2 & Lambda
- Cloud du Gartner

Le cloud – c’est large !

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

Le cloud - Côté économique

- Appréhender les ressources IT comme des services “fournisseur”
- Faire glisser le budget “investissement” (Capex) ver 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

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”

Caractéristiques du Cloud

Les 5 caractéristique selon NIST :

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

- 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

Les modèles de service - Modèles Cloud

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

Les modèles de service - IaaS

- Infrastructure as a Service
- Infrastructure :
  - Compute (calcul)
  - Storage (stockage)
  - Network (réseau)
- Utilisateurs cibles : administrateurs (système, stockage, réseau)

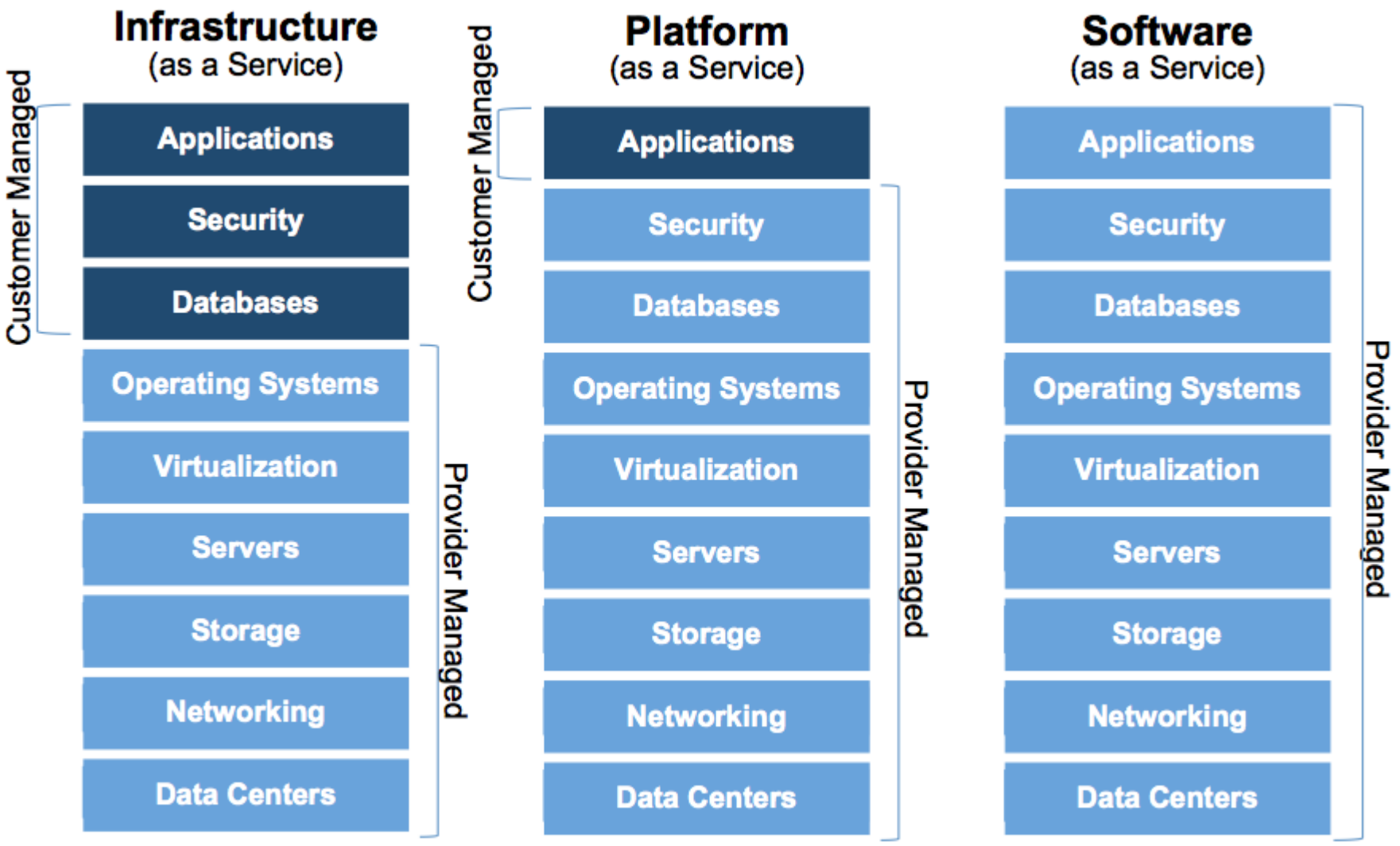
Les modèles de service - PaaS

- 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

Les modèles de service - SaaS

- Software as a Service
- Utilisateurs cibles : utilisateurs finaux

Les modèles de service - un schéma



IaaS - PaaS - SaaS (source : Wikipedia)

Les modèles de déploiement - 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

Les modèles de déploiement - 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)

Les modèles de déploiement – 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

Les modèles de déploiement - Alternatives IaaS publics à AWS

- Google Cloud Platform
- Microsoft Azure
- DigitalOcean
- Alibaba Cloud
- En France :
  - Scaleway
  - OVH
  - Outscale

Les modèles de déploiement - IaaS privé

- OpenStack
- CloudStack
- Eucalyptus
- OpenNebula

Les modèles de déploiement - OpenStack

- Naissance en 2010
- Fondation OpenStack depuis 2012...
- ...rebaptisée Open Infrastructure Foundation en 2020
- (https://openinfra.dev/)
- Écrit en Python et distribué sous licence Apache 2.0
- Soutien très large de l'industrie et contributions variées

Les modèles de déploiement - 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)

Les modèles de déploiement - Solutions de PaaS privé

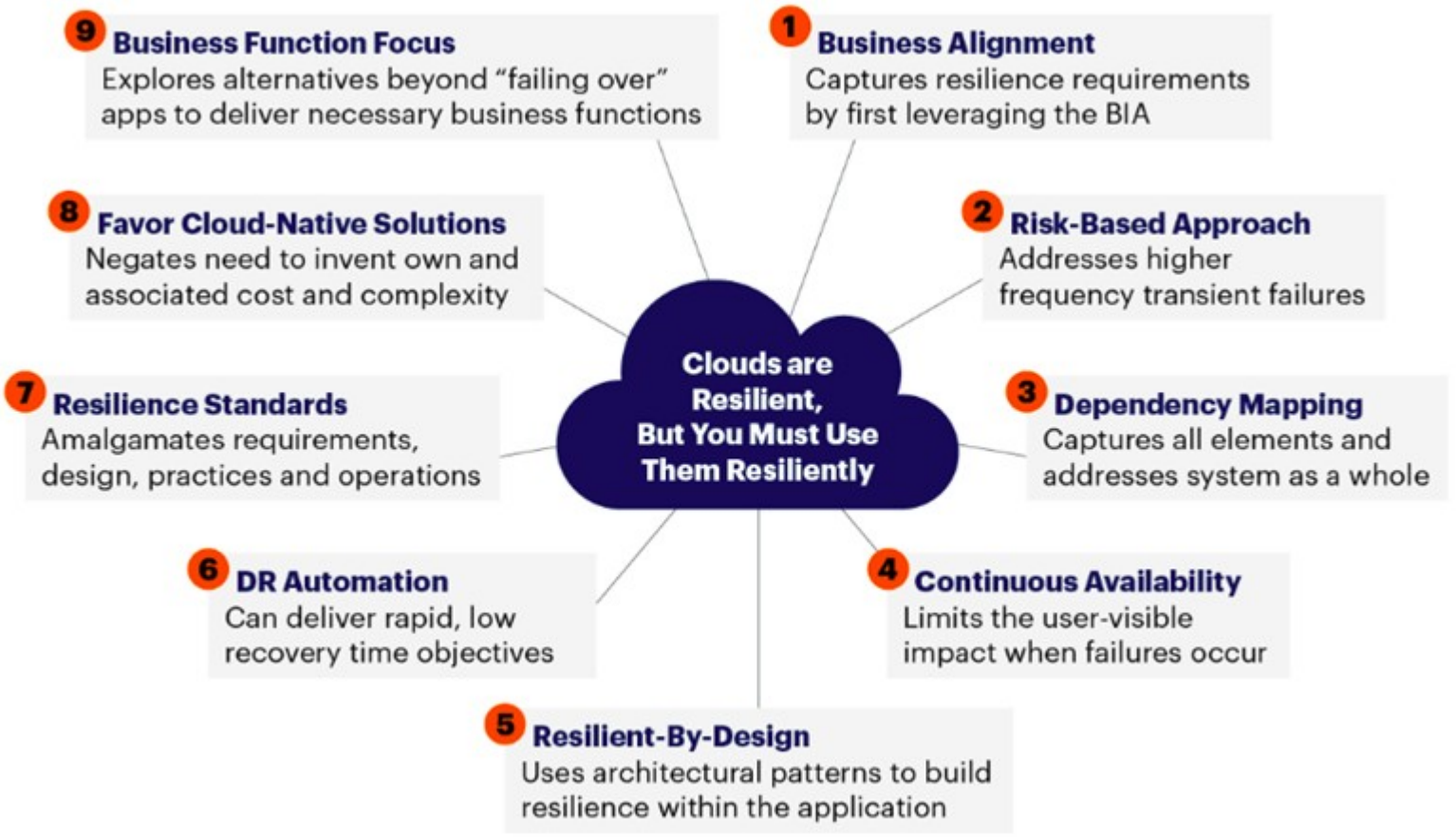
- Cloud Foundry, Fondation (https://www.cloudfoundry.org/)
- OKD, Red Hat (https://www.okd.io/)
- Solum, OpenStack (https://wiki.openstack.org/wiki/Solum)

Démonstration : AWS - Services EC2 & Lamdba

Le cloud de Gartner

- Gartner Inc. : entreprise américaine de conseil et de recherche dans le domaine des techniques avancées
- Bogues logiciels, et pas défaillances physiques, à l'origine de la quasi-totalité des pannes de cloud
- La plupart des pannes sont partielles
- Utiliser les cloud de manière résiliente
- 9 principes clés pour améliorer la résilience du cloud

Le cloud de Gartner – améliorer résilience du cloud



Le cloud de Gartner – améliorer résilience du cloud

1- Alignement sur l'activité

2- Approche basée sur les risques

3- Cartographie des dépendances

4- Disponibilité continue

5- Résilience par conception

6- Automatisation du DR

7- Normes de résilience

8- Privilégier les solutions natives du cloud

9- Concentration sur les fonctions de l'entreprise

Les conteneurs

- Le conteneur
- La Technologie
- Démonstration : Créer un conteneur avec Podman
- OS orientés conteneurs
- Conteneur Linux
- Orchestration des conteneurs

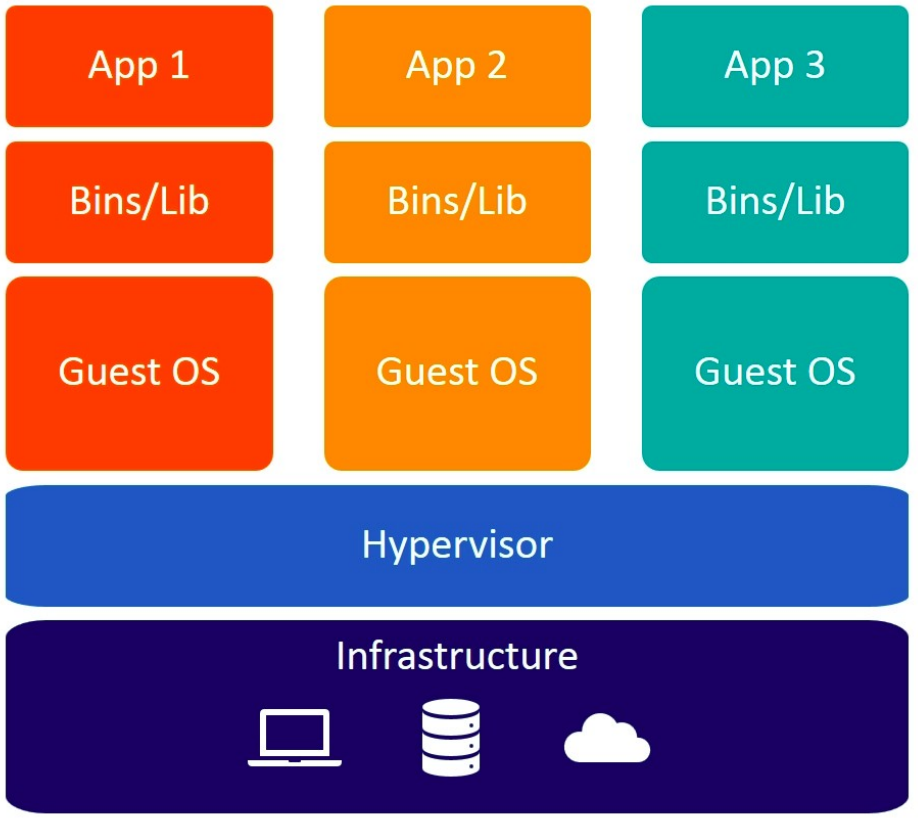
Le conteneur - avantages et inconvénients des VM

- Avantages :
  - Emulation bas niveau
  - Sécurité/compartimentation forte hôte/VMs et VMs/VMs
- Inconvénients
  - Usage disque important
  - Impact sur les performances

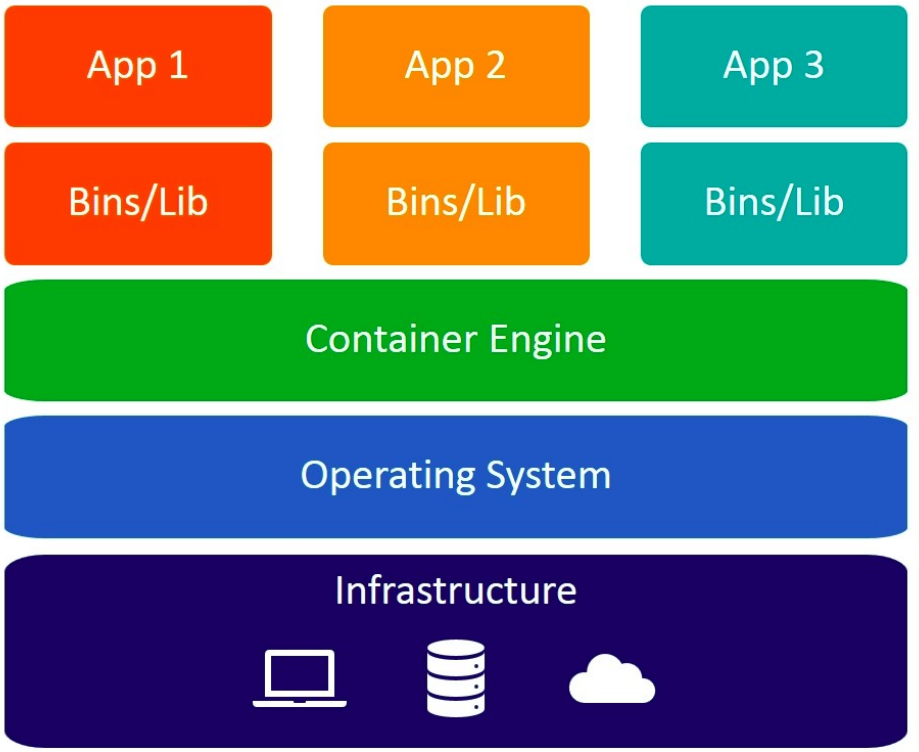
Le conteneur - c’est quoi ?

- Les mêmes idées que la virtualisation, mais sans virtualisation
- Isolation et automatisation
- Principe d’infrastructure consistante et répétable
- Peu d’*overhead* par rapport à une VM !
- Un super chroot
- Certains parlent de virtualisation ”niveau OS” ou ”légère”, isolation applicative

Le conteneur - Différences avec la VM



Virtual Machines



Containers

Le conteneur - Positionnement des conteneurs dans le 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

La technologie - Deux philosophies de conteneurs

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

La technologie - 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

La technologie - Container Runtime

- Docker
- Podman
- Rkt
- LXC

La technologie - LXC

- Ensemble d’outils en espace utilisateur
- Un conteneur est un dossier dans */var/lib/lxc*
- Petit fichier de configuration + root fs
- Les premières versions n’avaient pas de support CoW
- Les premières versions ne supportent pas le déplacement d’images

La technologie - Rocket (Rkt)

- Se prononce ‘rock-it’
- Développé par CoreOS
- Pas de daemon : intégration avec systemd
- Utilise systemd-nspawn et propose maintenant d’autres solutions (eg. KVM)
- Adresse certains problèmes de sécurité inhérents à Docker

La technologie - Docker Engine

- Open source développé par dotCloud en mars 2013.
- Fonctionne en mode *daemon* : difficulté d'intégration avec les init-process
- Utilisait la lib lxc
- Utilise désormais la libcontainer
- Daemon accessible via une API REST
- Gestion des conteneurs, images, builds, et plus

La technologie - Podman

- Développé par des ingénieurs Red Hat et des membres de la communauté Open Source
- Gère les conteneurs à l'aide de la bibliothèque *libpod*
- Architecture sans *daemon*

podman

Démonstration : créer un conteneur avec Podman

OS orientés conteneurs – OS traditionnelles

- Debian, CentOS, Ubuntu
- Supportent tous Docker
- Paquets DEB et RPM disponibles
- Paquets inutiles ou out of date
- Services par défaut/inutiles alourdissent les distributions

OS orientés conteneurs – objectifs

- 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)

OS orientés conteneurs – CoreOS : phylosophie

- 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

OS orientés conteneurs – CoreOS : Fonctionnalités

- 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

OS orientés conteneurs – RancherOS : phylosophie

- 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

OS orientés conteneurs – Fedora Atomic : phylosophie

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

OS orientés conteneurs – Projet Photon

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

Conteneur Linux - principe

- Processus isolé par des mécanismes noyaux.
- 3 éléments fondamentaux:
  - Namespaces
  - Cgroups
  - Copy-On-Write

Conteneur Linux - Les namespaces

- Apparue dans le noyau 2.4.19 en 2002, réellement utiles en 2013 dans le noyau 3.8
- Limite ce qu'un processus peut voir du système:
- un processus ne peut utiliser/impacter que ce qu'il peut voir
- Types: pid, net, mnt, uts, ipc, user
- Chaque processus est dans un namespace de chaque type

Conteneur Linux - Control Groups

- Fonctionnalité du noyau, apparue en 2008 (noyau 2.6.24)
- Des fonctionnalités du noyau
- Contrôle précis de l'allocation des ressources pour un/groupe de processus, appelés tâches:
  - Allocation
  - Monitoring
  - limite
- Types :
  - cpu, memory, network, block io, device

Conteneur Linux - Copy-On-Write

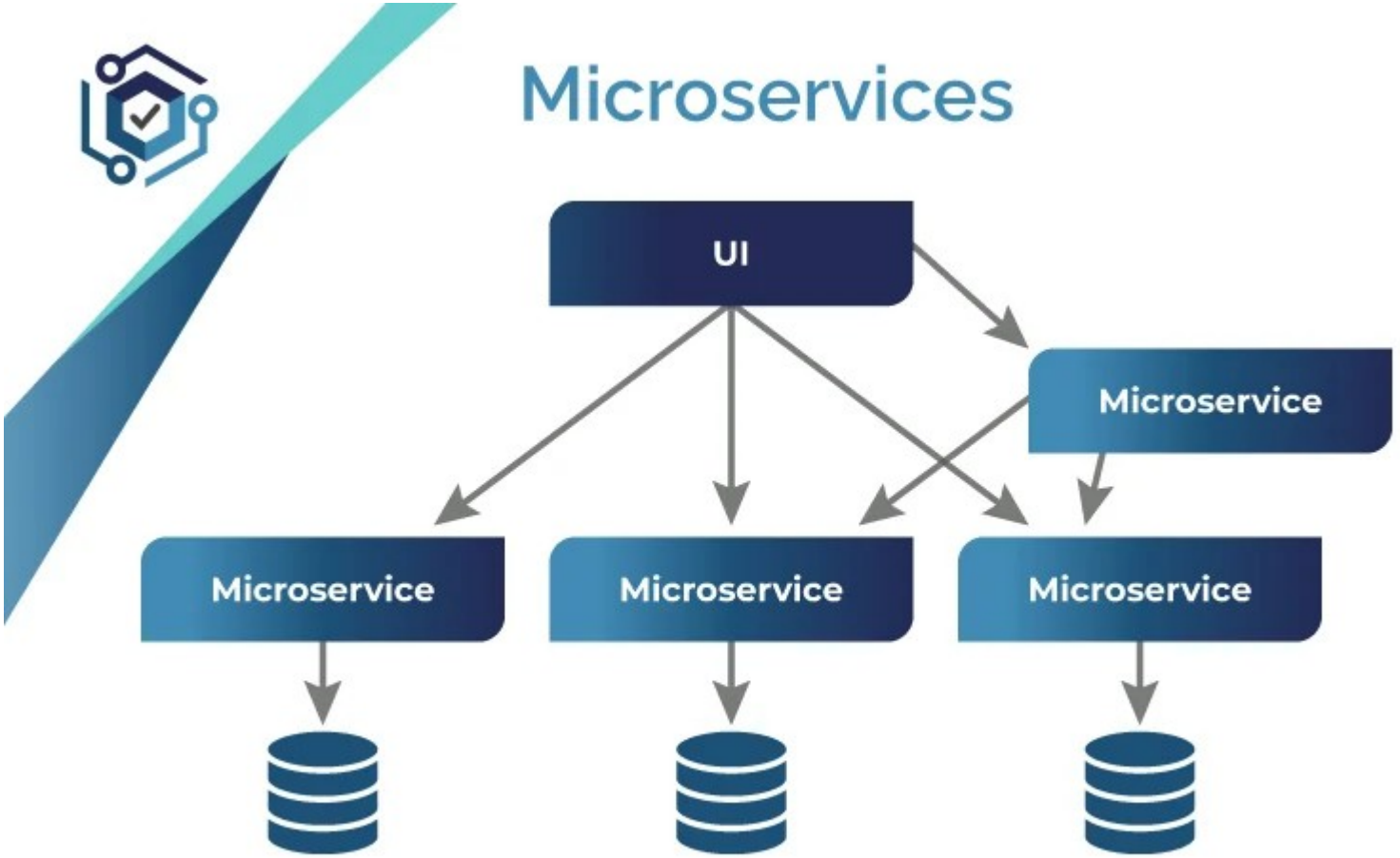
- COW : Donne un pointeur vers la même ressource plutôt que donner une copie.
- Réduit l'usage disque et le temps de création
- Plusieurs options disponibles
  - device mapper (niveau fichier)
  - btrfs, zfs (niveau fs)
  - aufs, overlay (niveau fichiers)
- Le type de stockage surveille les modifications

Orchestration des conteneurs - orchestration

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

Orchestration des conteneurs – Les microservices

- Découpage des applications en micro services



Orchestration des conteneurs - IaC

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

Orchestration des conteneurs - Scaling

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

Orchestration des conteneurs - orchestrateurs

- **Docker-compose**: surcouche à Docker, simple, capacités assez limitées.
- **Docker Swarm**: gestion de cluster de machines, scaling, intégré à Docker.
- **Mesos**: gestion de (très gros) clusters de machines, scaling, gestion fine des ressources, templates et bibliothèques de déploiement.
- **Kubernetes** : nombreuses fonctionnalités (clusters, scaling, templates, ressources, stockage), templates et bibliothèques (via helm), contrôle d’accès.

Orchestration des conteneurs – k8s vs Docker swarm

	Kubernetes	Docker Swarm
Installation & Configuration	Installation difficile et compliqué	Installation très facile: Intégration dans l'environnement Docker déjà réalisée
GUI	Offre un outil intégré : taableau de bord clair et facile à utiliser.	GUI uniquement avec un logiciel supplémentaire
Scalability	Un peu lent dans la mise à l'échelle	Mise à l'échelle extrêmement rapide: 5× plus rapide que Kubernetes
Auto-Scaling	Offre une mise à l'échelle automatique.	N'offre pas une mise à l'échelle automatique.
Logging & Monitoring	Fait partie intégrante de l'application	Uniquement avec un logiciel supplémentaire

Docker

- Docker
- Layers
- Volumes
- Réseau
- Publication de ports
- Build, Ship & Run
- Démonstration : Dockeriser l’application DockerCoins

Docker - Docker Engine

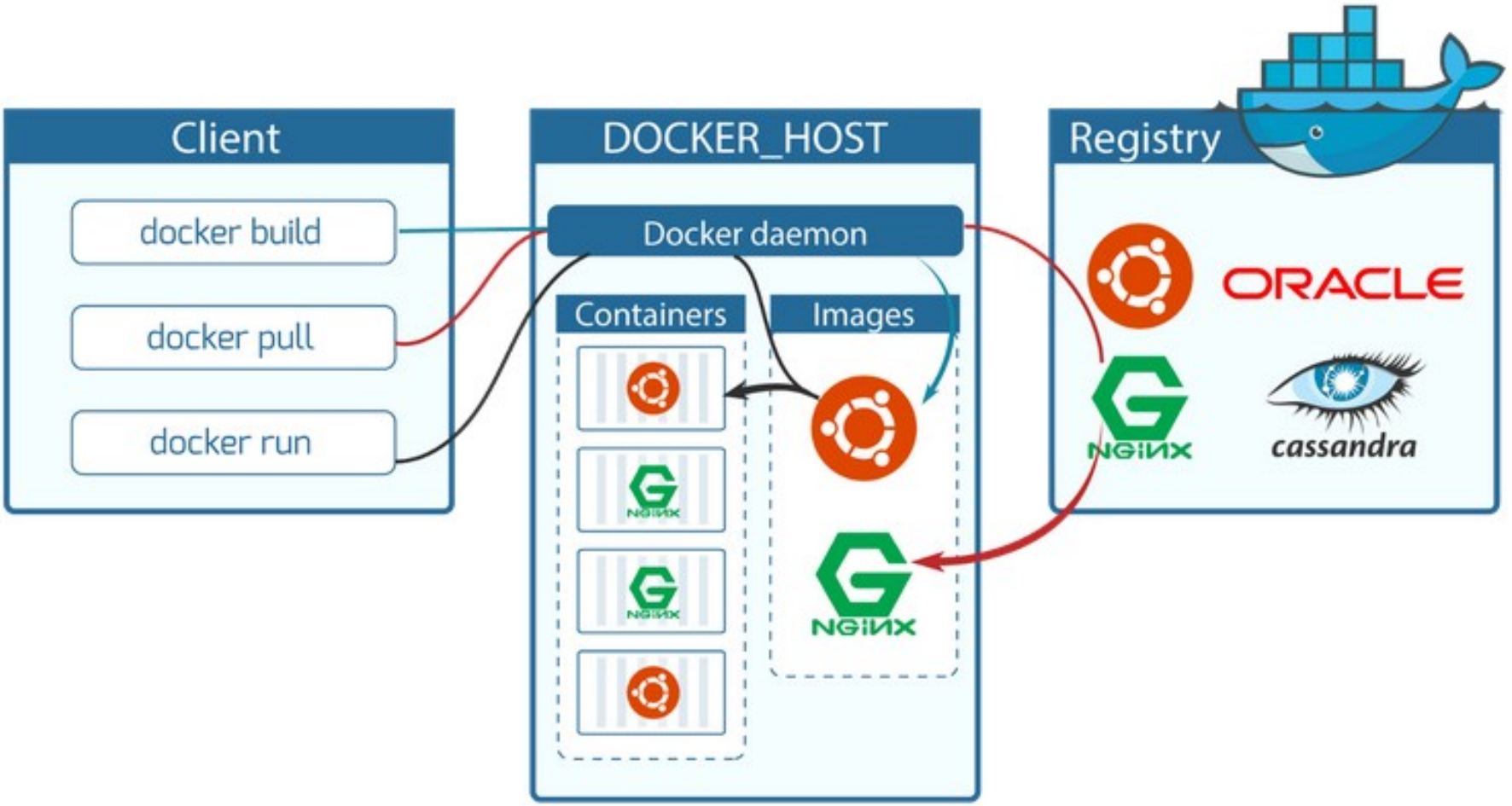
- Développé par dotCloud en mars 2013
- Développé en langage Go de Google
- Fonctionne en mode *daemon*
- Utilisait la lib *lxc*
- Utilise désormais la *libcontainer*
- *Daemon* accessible via une API REST
- Gère les conteneurs, images, builds, et plus

- 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

Docker - Exécuter des conteneurs partout

- Sur n’importe quelle plate-forme: physique, virtuel, cloud,
- Pouvoir passer de l’une à l’autre des plate-formes,
- Maturité des technologies (cgroups, namespaces, copy-on-write)

Docker – fonctionnement



Docker – Concepts et composants

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

Layers – Pourquoi ?

- 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

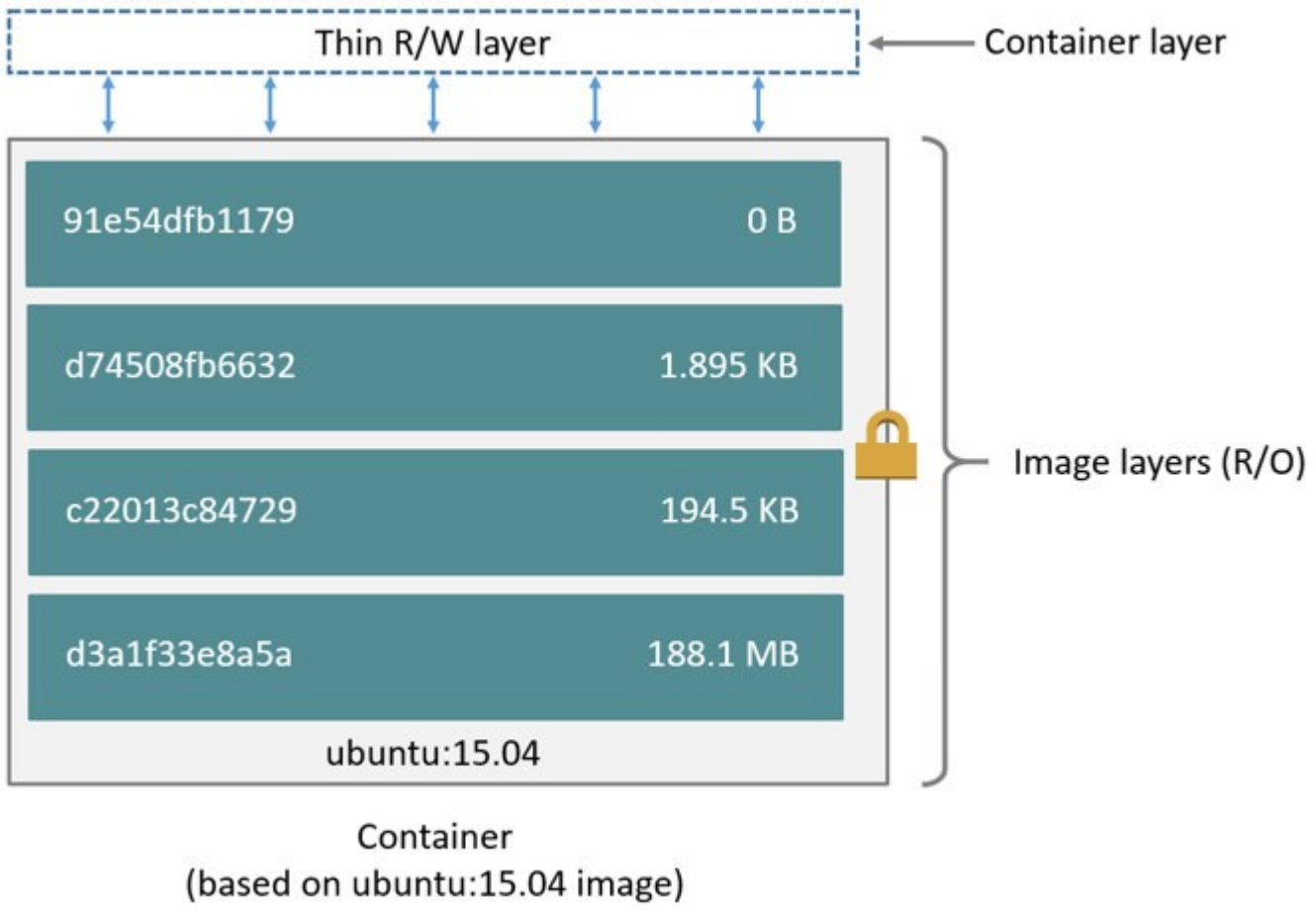
Image

- Une image se décompose en layers R/O

Layers – Dockerhub

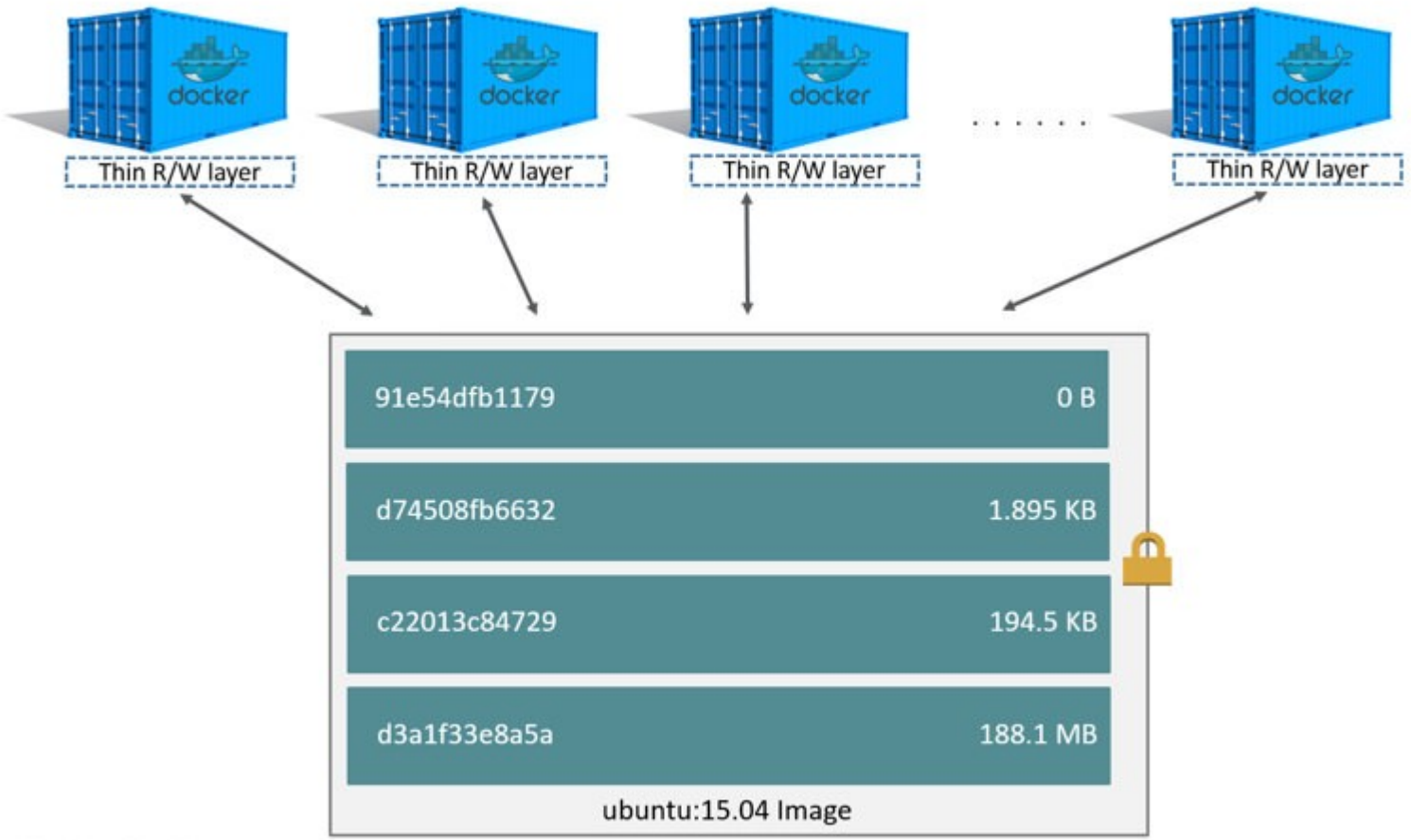
- Une *registry* pour stocker et distribuer des images docker
- DockerHub n’est qu’au *Docker registry* ce que GitHub est à git
- Pull & Push des images

Layers – Un conteneur



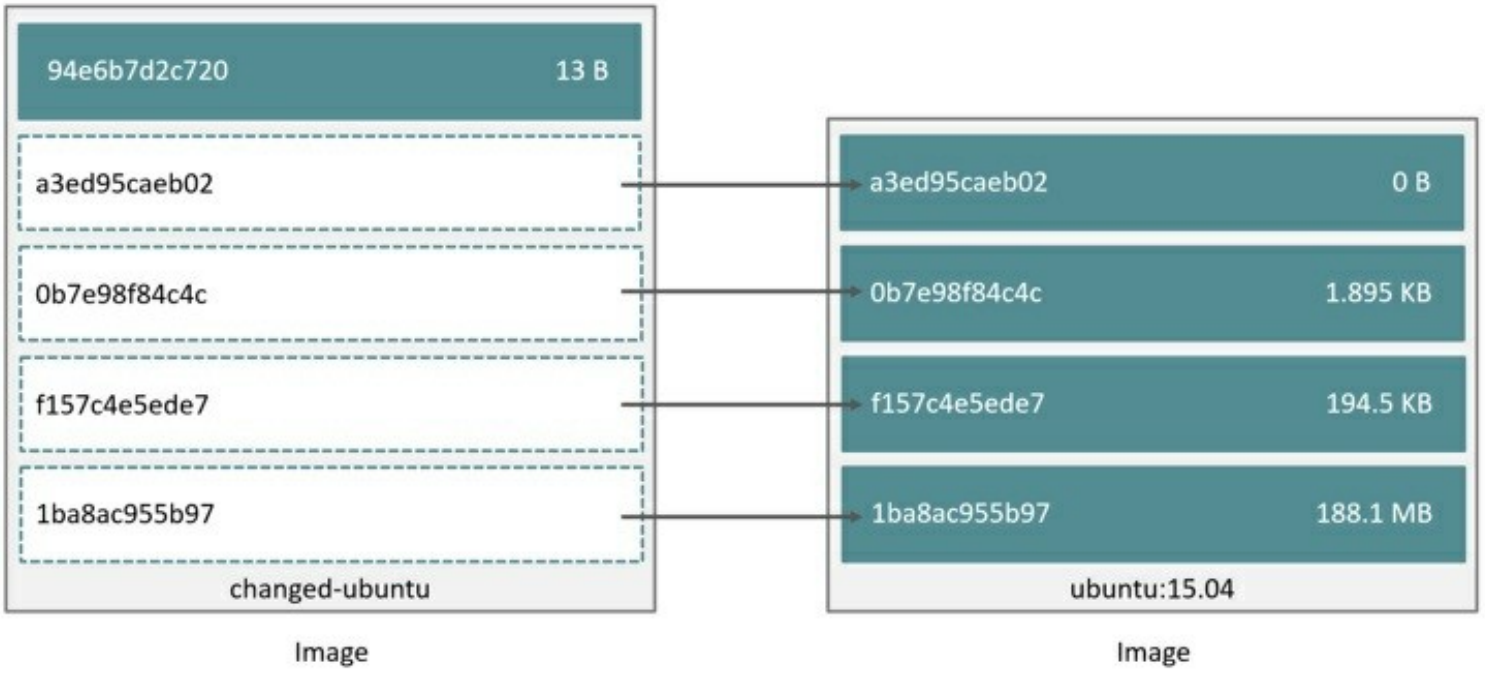
- Un conteneur = une image r/o + un layer r/w

Layers – plusieurs conteneurs



- Une image, exécutée par plusieurs conteneurs

Layers – Répétition des layers

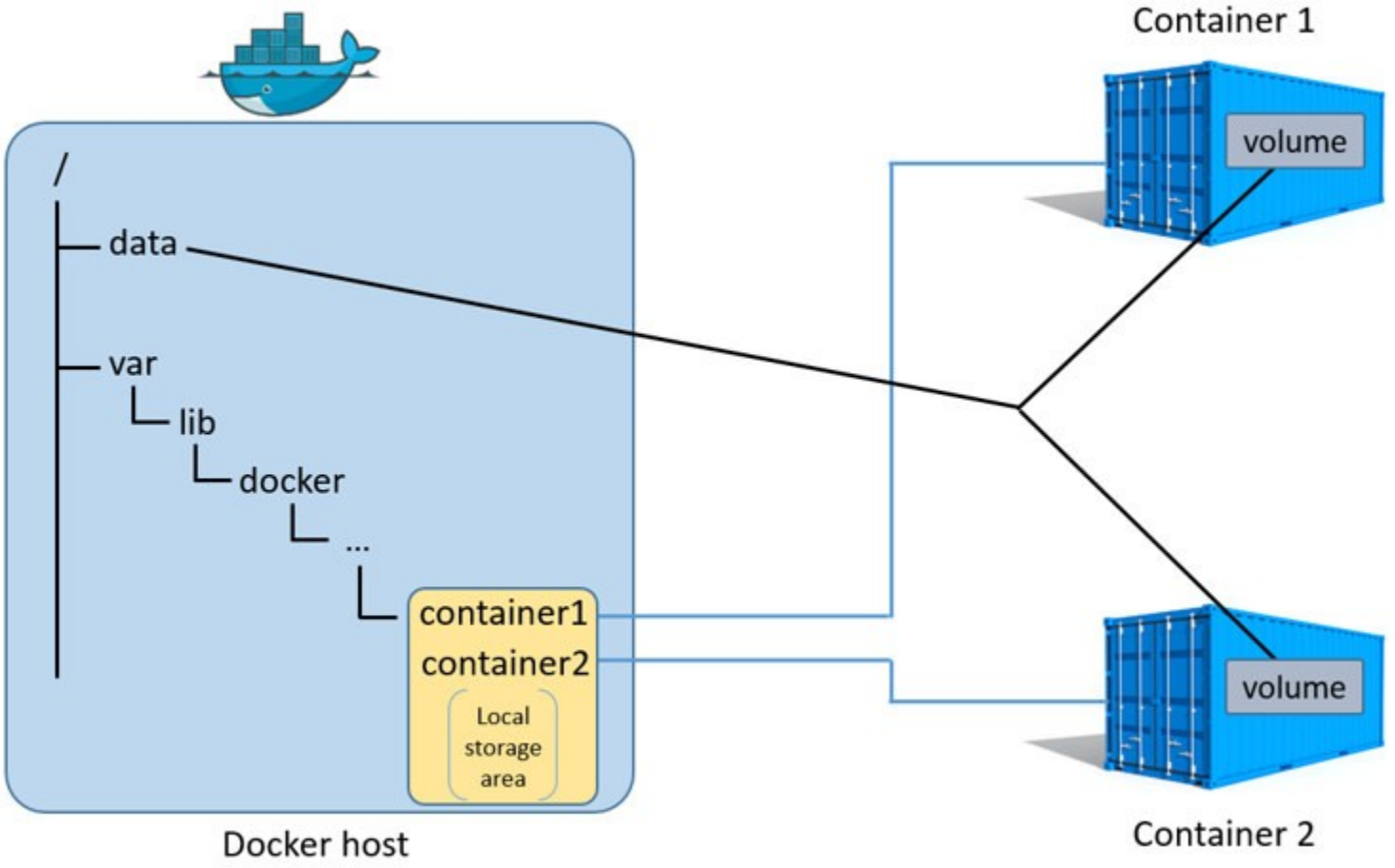


- Les layers sont indépendants de l'image

Volumes – Pourquoi ?

- Assurent une persistance des données
- Indépendance du volume par rapport au conteneur et aux layers
- 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

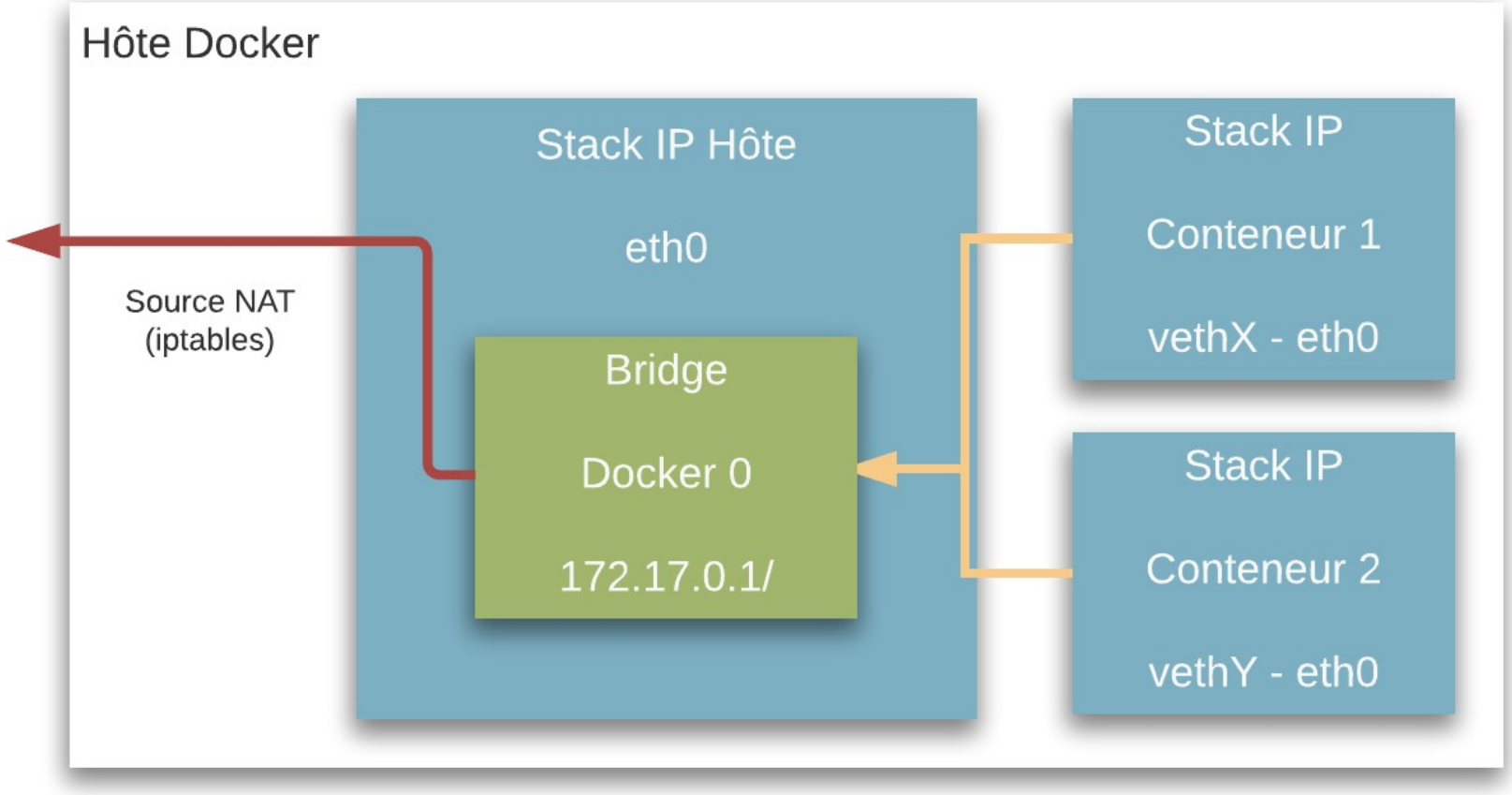
Volumes – Plugins

- 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

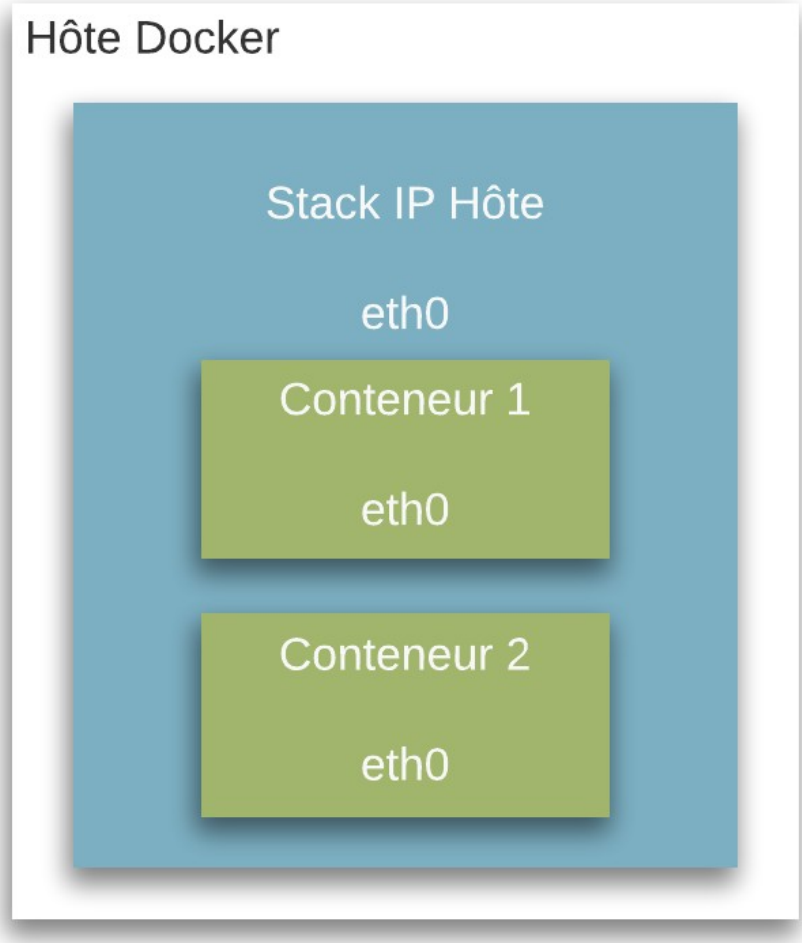
Réseau – à la base, pas grand-chose …

```
NETWORK ID     NAME      DRIVER    SCOPE
b44849e27cee   bridge    bridge    local
330ca4d7d52c   host      host      local
ab97579efbad   none      null      local
```

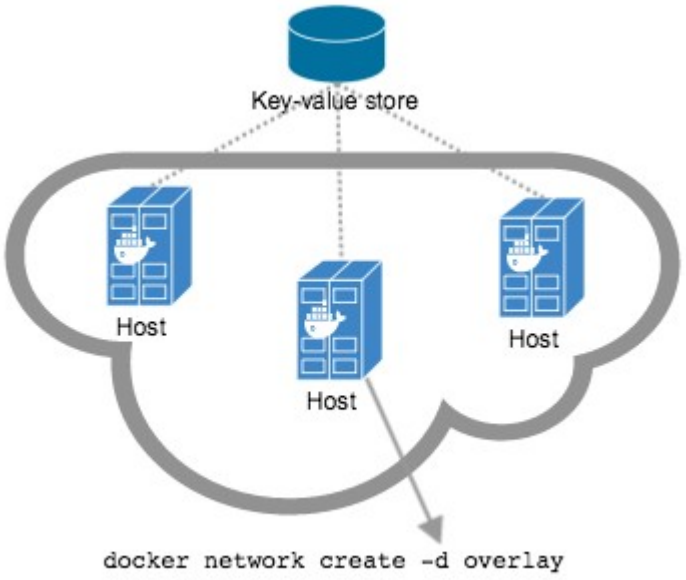
Réseau – Bridge



Réseau – Host


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

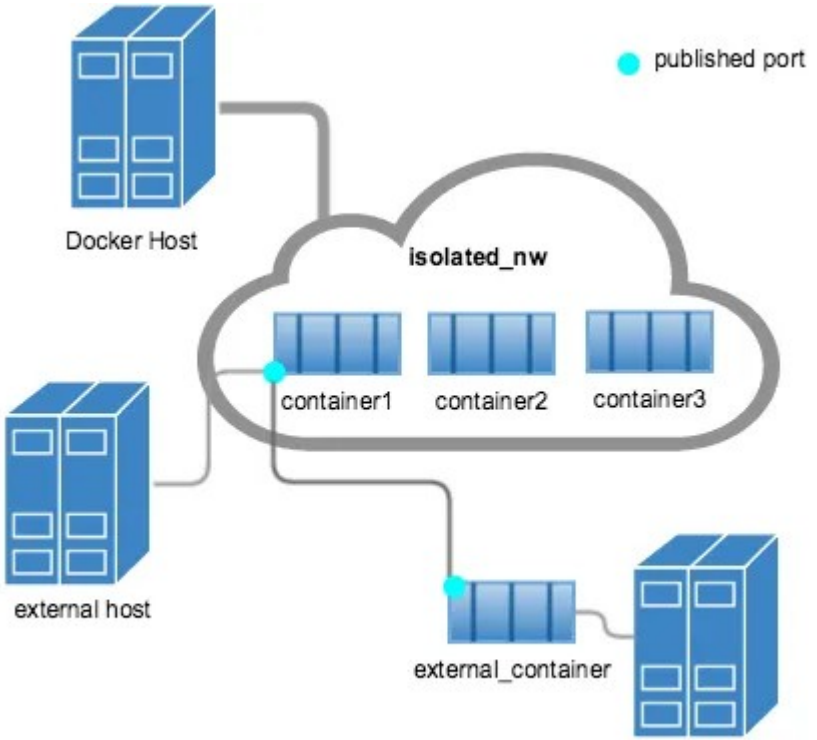
Réseau – Multihost Overlay



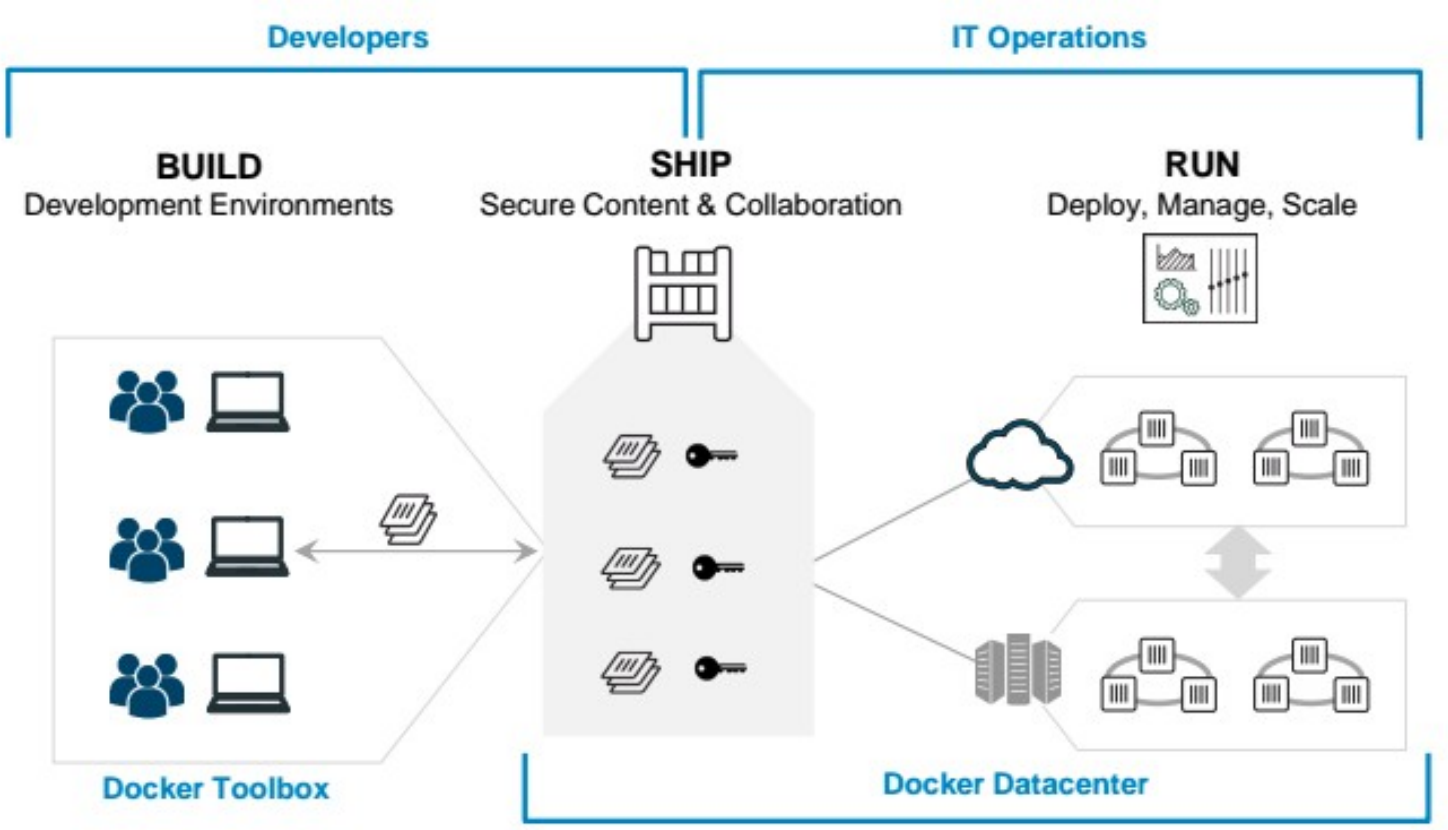
Publication de ports – Pourquoi ?

- 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 – Pourquoi ?



Build, Ship & Run – concept



Build, Ship & Run – Dockerfile

- Construction d’une image manuellement ou automatisée
- Dockerfile : Suite d'instructions 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"]
```

Build, Ship & Run – Dockerfile : principale instructions

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

Build, Ship & Run – Ship

- Sauvegarder un conteneur
- Exporter/Importer une image
- pull/push image de/vers *registry*

Build, Ship & Run – Ship : exemples

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

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

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

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

Build, Ship & Run – « Runner » un conteneur

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

- Exemple :

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

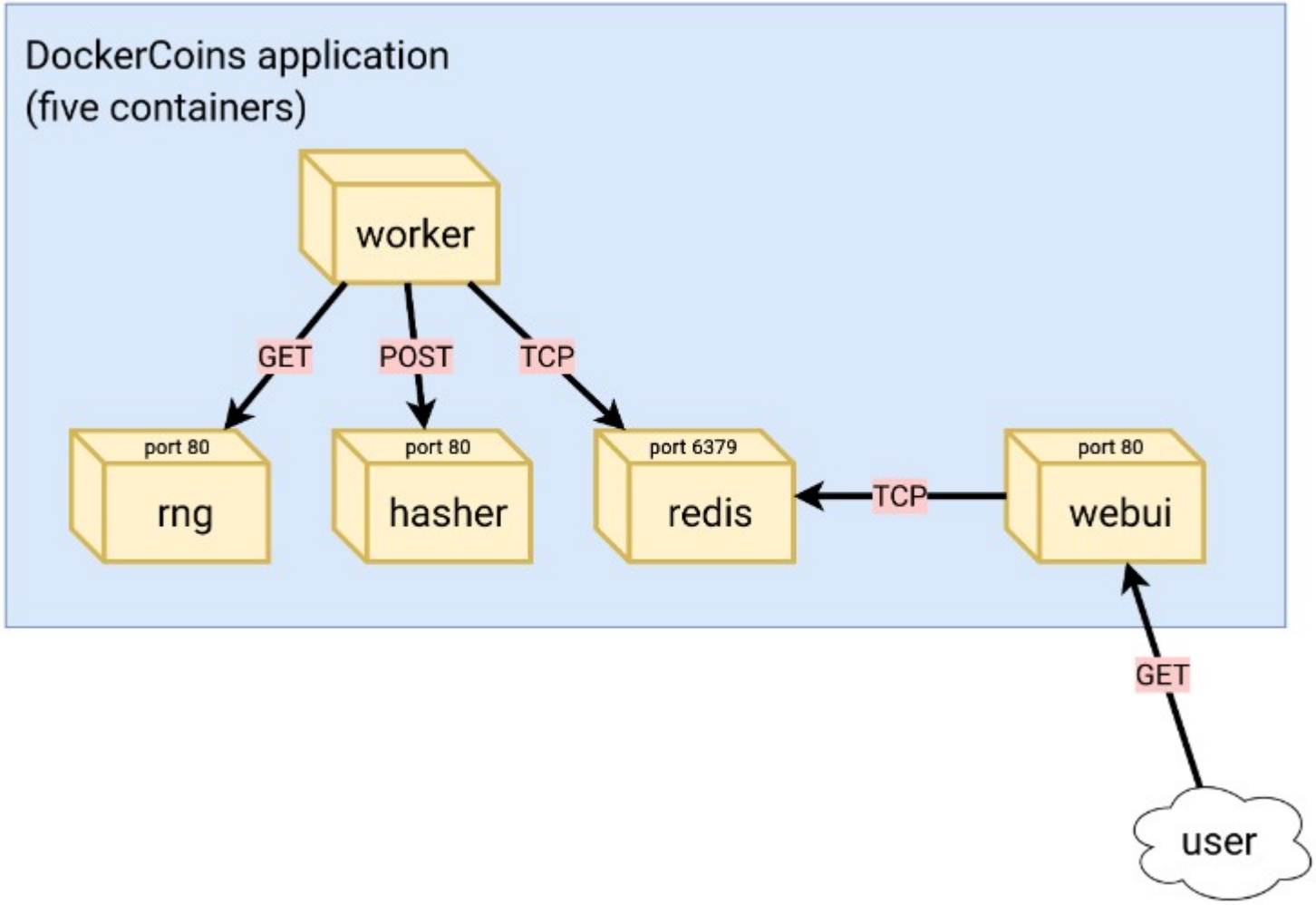
Build, Ship & Run – Se connecter au conteneur

- docker exec ...
- docker attach ...

Build, Ship & Run – Détruire le conteneur

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

Démonstration: Dockeriser l'application DockerCoins



Outils d’orchestration des conteneurs

- Orchestration Docker
- Démonstration : Déployer DockerCoins avec Docker Compose
- Démonstration : Déployer DockerCoins sur un Cluster Docker Swarm
- Projet Kubernetes
- Concepts et Objets Kubernetes
- Démonstration : Déployer et Orchestrer DockerCoins sur un cluster k8s
- Kubernetes vs Docker Swarm

Orchestration Docker - Axes d’orchestration

- Réseau
- Docker Compose
- Docker Swarm

Orchestration Docker – Docker Compose

- **Compose** permet d'éviter de gérer individuellement des conteneurs qui forment les différents services de votre application.
- Outil qui définit et exécute des applications multi-conteneurs
- Utilise un fichier de configuration dans lequel vous définissez les services de l'application
- A l'aide d'une simple commande, vous contrôlez le cycle de vie de tous les conteneurs qui exécutent les différents services de l'application.

Orchestration Docker – Utilisation de docker compose

- Définir l'environnement de l'application pour qu'il soit possible de la générer de n'importe où.
  - à l'aide de Dockerfile
  - à l'aide d'image officielle
- Définir les services dans un fichier *docker-compose.yml*
  - pour les exécuter et les isoler.
- Exécuter la commande *docker-compose* qui se chargera d'exécuter l'ensemble de l'application

Démonstration : Déployer DockerCoins avec Docker Compose

Orchestration Docker – Docker Swarm

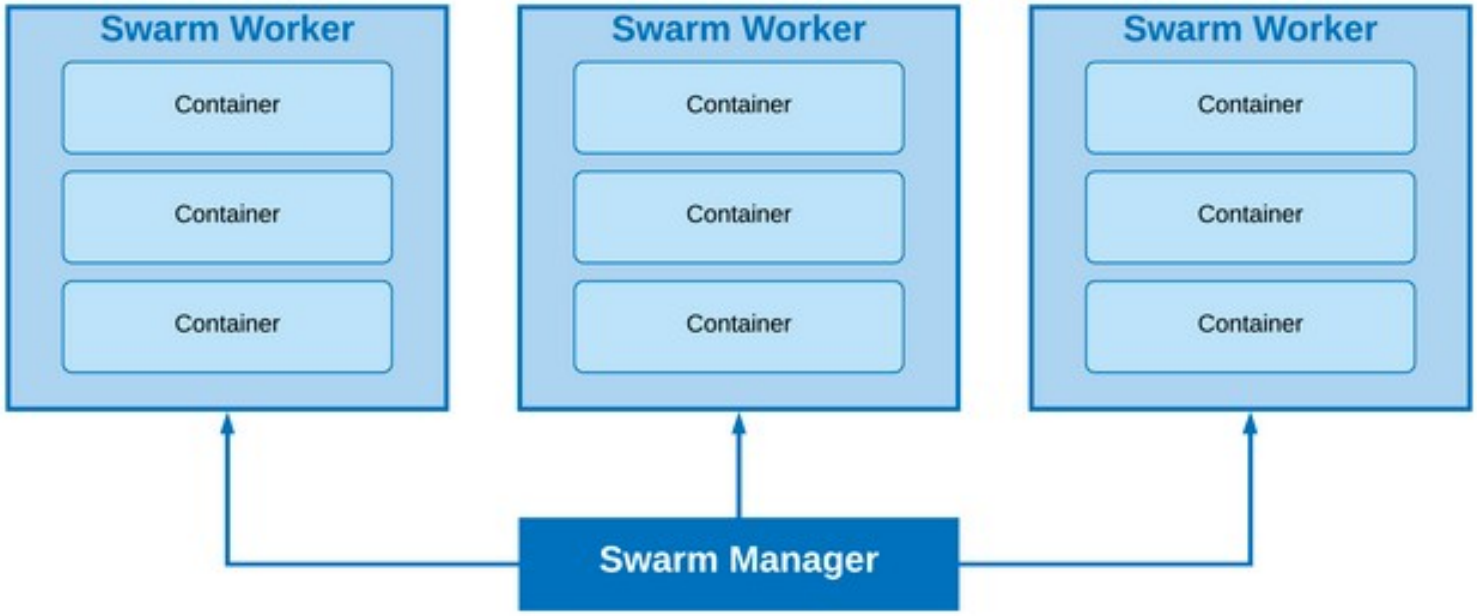
- Docker Engine 1.12 inclut le mode swarm pour nativement gérer un cluster de Docker Engines qu'on nomme un swarm (essaim).
- On utilise la CLI Docker pour :
  - créer un swarm,
  - déployer des services d'application sur un swarm,
  - gérer le comportement du swarm.

Orchestration Docker – Mode Swarm

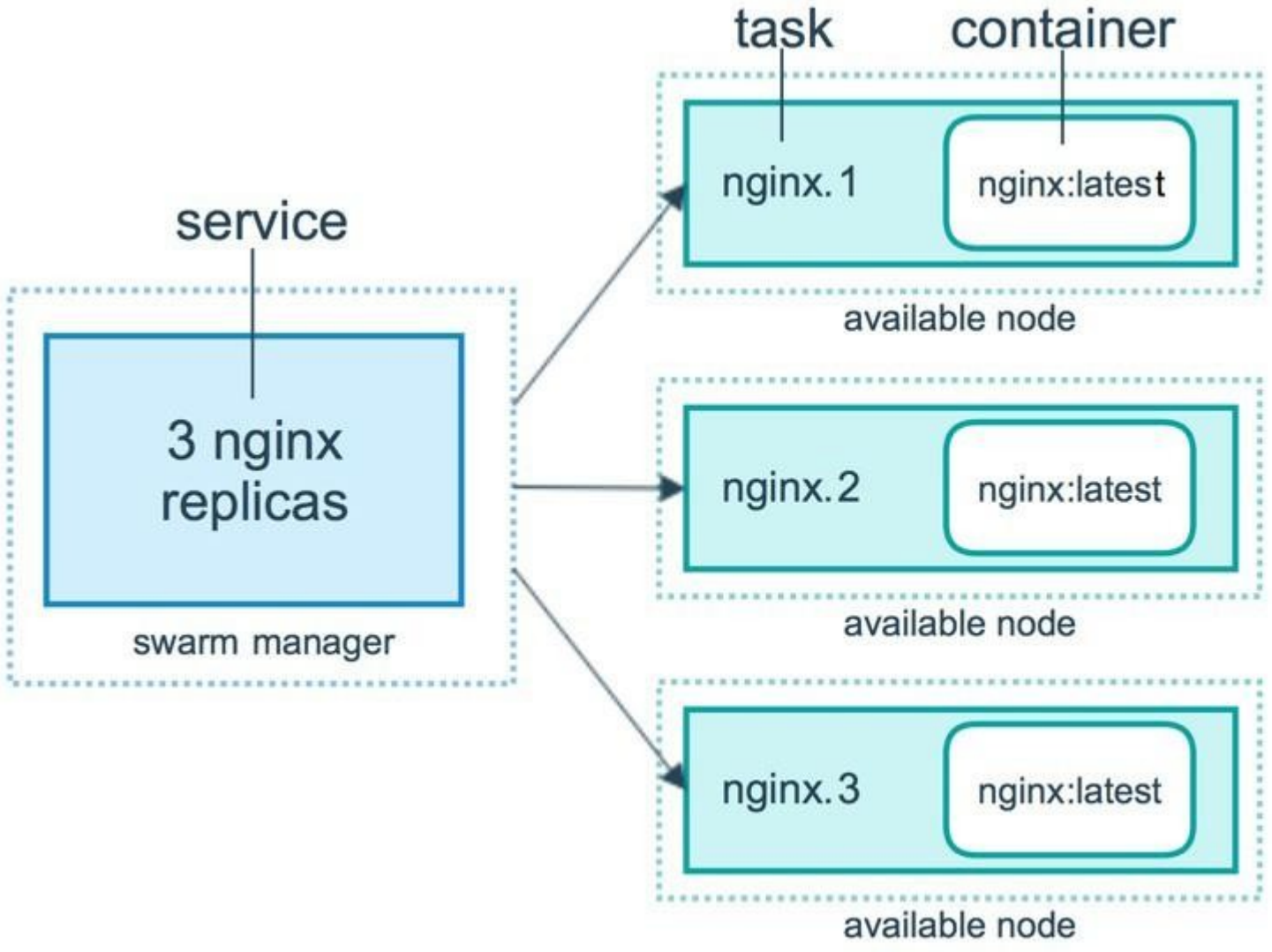
- Les fonctionnalités de gestion et orchestration de cluster incluses dans le Docker Engine.
- Un swarm (essaim) est un cluster de Docker Engines sur lequel vous déployez des services.
- La CLI Docker inclut la gestion des nœuds d'un swarm
  - ajout de noeuds,
  - déploiement de services,
  - gestion de l'orchestration des services

Orchestration Docker – Notion de nœud

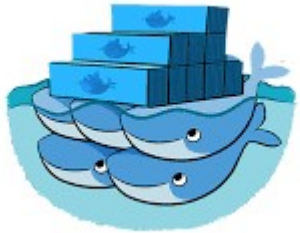
- Un nœud est une instance *Docker Engine* participant à un swarm.
- 2 types de nœuds :
  - *Manager*
  - *Worker*



Orchestration Docker – services et tâches



Démonstration : Déployer DockerCoins sur un cluster Docker Swarm



- Développé par Google, devenu open source en 2014
- Adapté à tout type d'environnement
- Devenu populaire en très peu de temps
- Premier projet de la CNCF

kubernetes

- *The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes.*

Projet Kubernetes – Cycle de développement

- Chaque *release* a son propre planning, pour exemple :

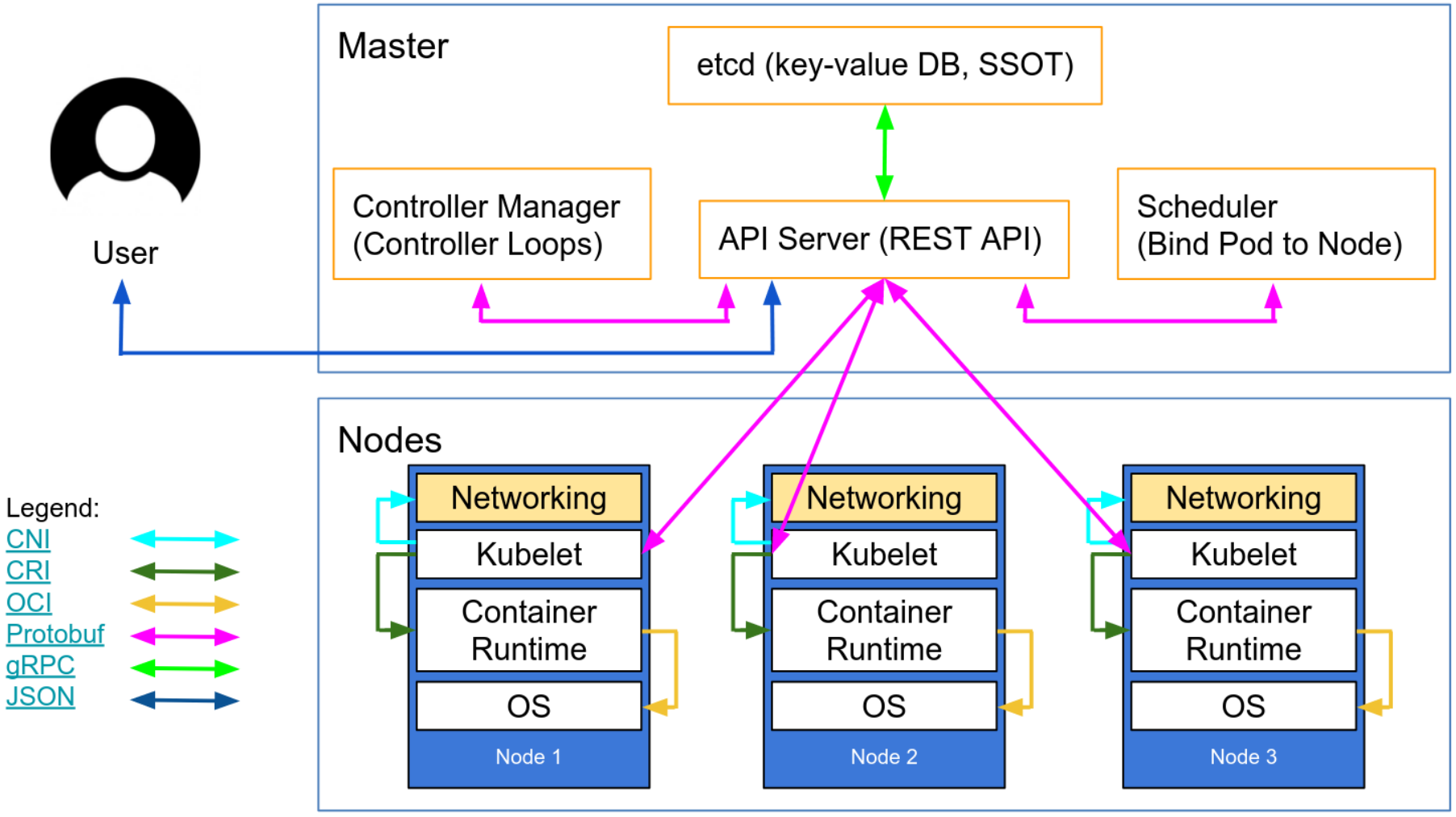
  https://github.com/kubernetes/sig-release/tree/master/releases/release-1.30#timeline

- Chaque cycle de développement dure 12 semaines et peut être étendu si nécessaire

Projet Kubernetes – KubeCon

- La CNCF organise trois KubeCon par an :
- Amérique du Nord : 12 – 15 Novembre 2024 à Salt Lake
- Europe : 19 – 22 Mars 2024 à Paris
- Chine : 21 – 23 Août 2024 à Hong Kong

Projet Kubernetes – Le cluster



Concepts & Objets Kubernetes – API RESOURCES

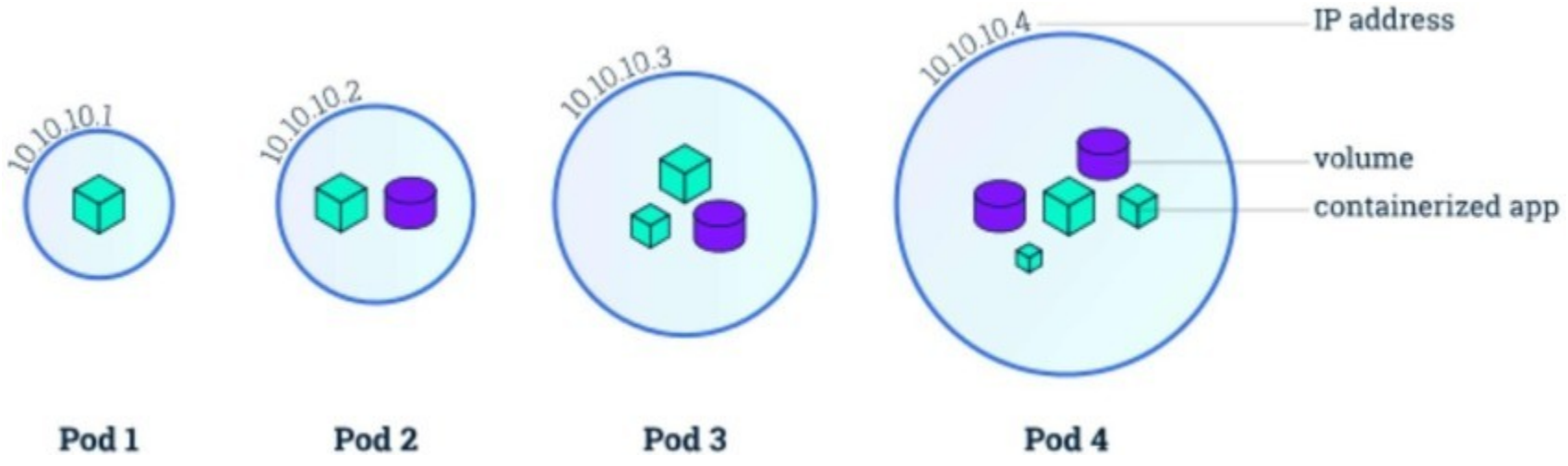
- Namespaces
- Pods
- Services
- Deployments
- DaemonSets
- StatefulSets
- Etc ...

Concepts & Objets 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

Concepts & Objets Kubernetes – Pod

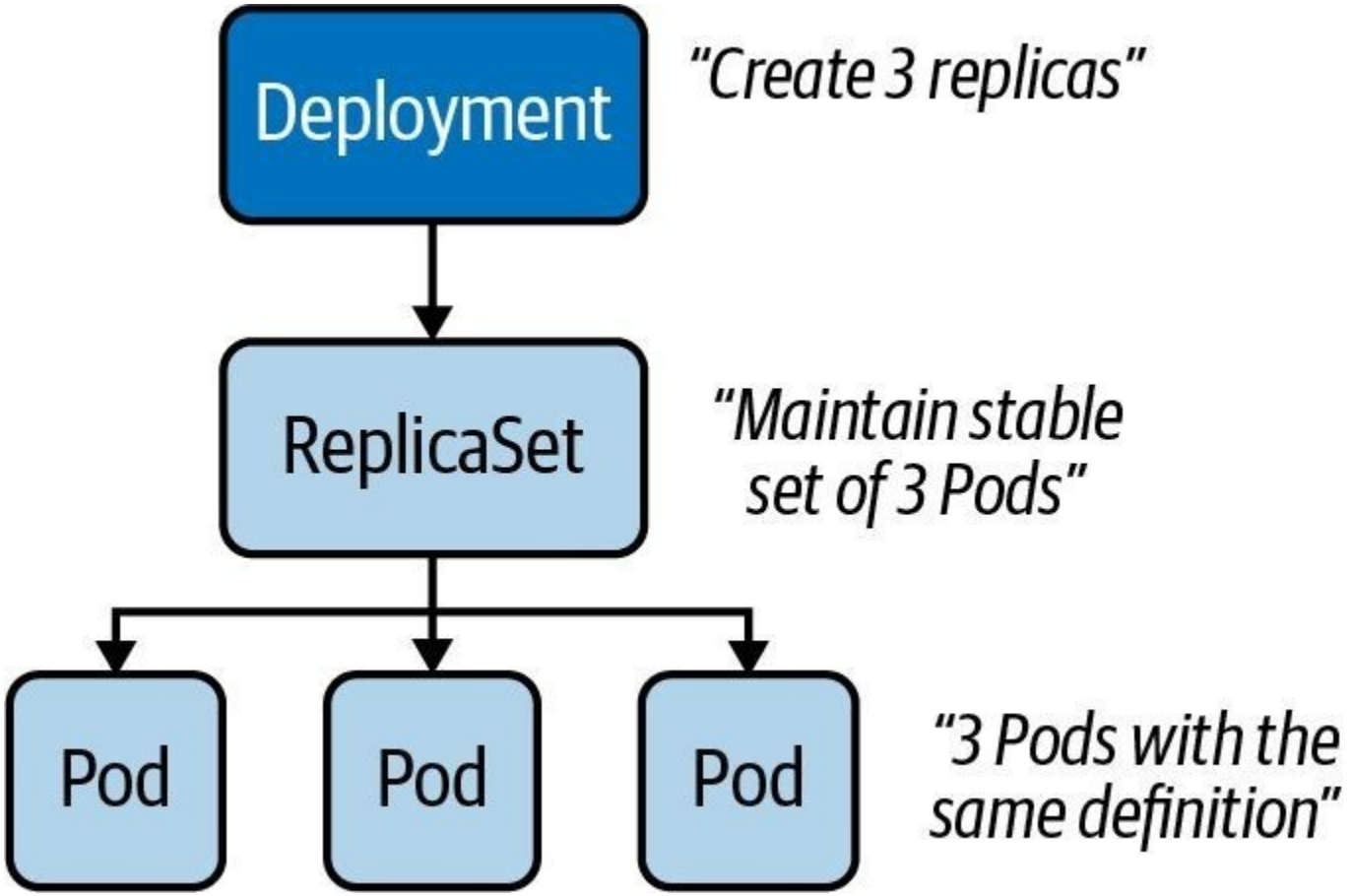
- Représente la plus petite unité déployable et exécutable
- Brique de base qui encapsule un ou plusieurs conteneurs
  - ayant une seul et unique @IP.
  - Il a son propre réseau et volume(s) interne(s).



Concepts & Objets Kubernetes – Deployment

- Contrôle et pilote des ReplicaSets et des pods.
- Il ajoute des fonctionnalités aux ReplicaSet comme :
  - Déployer (rollout) des réplicas de Pod
  - Déclarer un nouvel état des pods (nouvelle version)
  - Revenir à une ancienne version du Deployment (rollback)
  - Mettre à l’échelle le Deployment (pour gérer les montées en charge par exemple)
  - Etc ...

Concepts & Objets Kubernetes – Deployment



Concepts & Objets Kubernetes – DaemonSet

- Assure que tous les nœuds 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

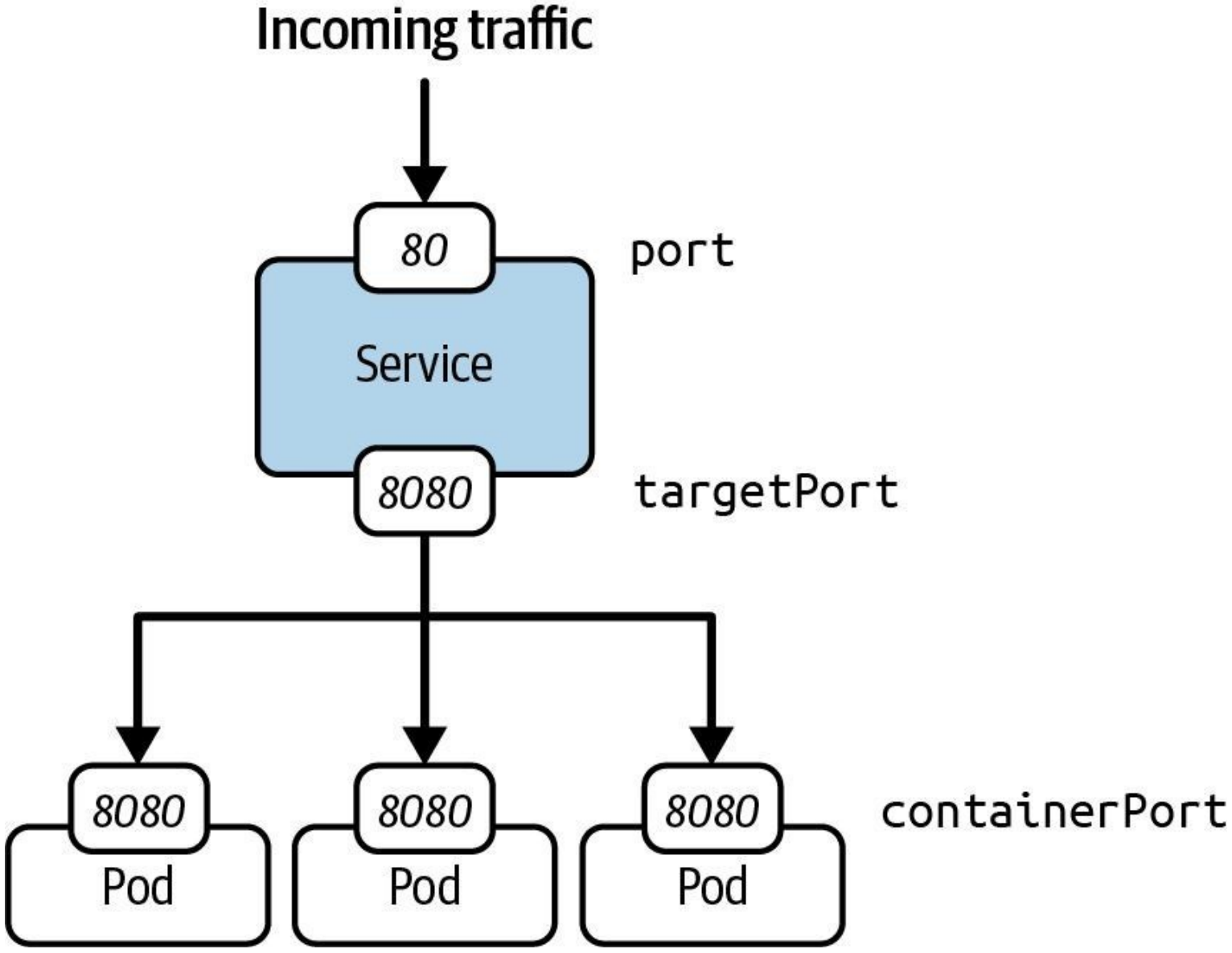
Concepts & Objets Kubernetes – StatefulSet

- Similaire au Deployment, mais *Stateful*
  - Deployment et DeamonSet sont *Stateless*
- 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

Concepts & Objets Kubernetes – Service

- 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

Concepts & Objets Kubernetes – Service



Démonstration : Déployer et orchestrer DockerCoins sur un cluster k8s

Kubernetes vs Docker Swarm – Terminologies

	Docker swarm	Kubernetes
Controller	Manger	Master
Slave	Worker	Node worker
Workload Definition	Service	Deployment
Deployment Unit	Task	Pod
Scale-out Definition	Replicas	Replica Set
Service Discovery	DNS	DNS
Load Balancing	Ingress	Service
Port	PublishedPort	Endpoint
Storage	Volumes	Persistent Volumes / Claims
Network	Overlay	Flat Networking Space

Kubernetes vs Docker Swarm – Fonctionnalités

Features	Docker Swarm	Kubernetes
Installation & Cluster configuration	Installation very simple, but cluster not very strong	Insttallation complicated ; but once setup, the cluster is very strong
GUI	No GUI	GUI is the Kubernetes Dashboard
Scalability	Highly scalable & scales faster than kubernetes	Highly scalable & scales faste
Auto-Scaling	Can not do auto-scaling	Can do auto-scaling
Load Balancing	Does auto load balancing of trafic between containers in the cluster	Manual intervention needed for load balancing trafic between different containers in different Pods
Rolling Updates & Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks	Can deploy Rolling updates, & does automatic Rollbacks
Data Volumes	Can share storage volumes with any other container	Can share storage volumes only with other containers in same Pod
Logging & Monitoring	3rd party tools like ELK should be used	In-built tools for logging & monitoring

Le Container-as-a-Service

- Le CaaS
- CaaS vs. PaaS vs. IaaS
- Architecture
- Exemples pratiques
- Fournisseurs du CaaS
- Démonstration : AWS – ECS & EKS
- Bénéfices
- Limites

Le CaaS - Qu’est-ce que c’est

- Containers as-a-service
- Service basé sur le cloud
- Donne aux entreprises le moyen de gérer leurs
  - Applications virtualisées
  - Clusters
  - Conteneurs
- Permet de faciliter et d’accélérer les déploiements.

Le CaaS - Fonctionnement

- Le prestataire de CaaS héberge un moteur d'orchestration qui exécute les conteneurs d'une entreprise et maintient l'infrastructure partagée par ces derniers.
- Les utilisateurs peuvent accéder à ce service via une virtualisation par conteneurs, un appel API ou une interface de portail web.
- Au lieu d'une machine virtuelle (VM) ou d'un système hôte bare metal, le service est proposé via un conteneur,
  - Ce qui améliore son évolutivité et accélère son déploiement.

Le CaaS - Circonstances d’utilisation

- Le CaaS convient particulièrement bien aux déploiements de micro-applications
- Déploiements presque instantanés.
- Intégrer l’évolutivité automatique et la gestion de l’orchestration
- Réduire le temps que le personnel informatique consacre à chaque déploiement.

Le CaaS - Pourquoi est-il important ?

- En raison de ce qu'il permet/interdit aux équipes de développement de logiciels et aux départements informatiques de faire.
- Avant, les équipes devaient prêter attention à l'infrastructure sous-jacente supportant l'exécution des conteneurs.
- Une ressource dédiée a été chargée de superviser et de gérer les machines du cloud et les systèmes de routage du réseau.

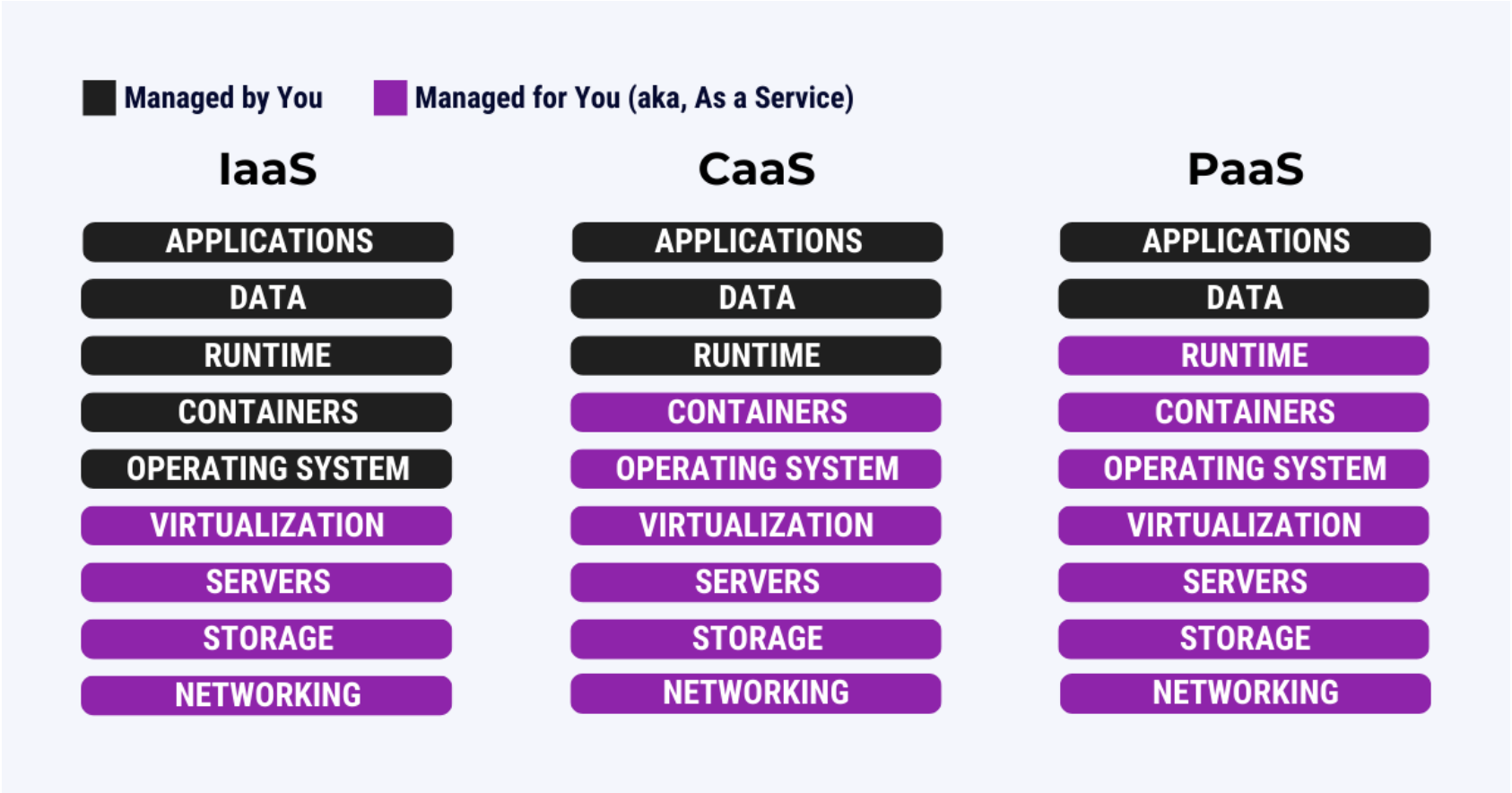
Le CaaS - Pourquoi est-il important ?

- L’avènement du CaaS a délesté ces ressources de ces tâches et a permis aux équipes d’utiliser le temps ainsi économisé pour créer et tester l’infrastructure des conteneurs avant de déployer ceux-ci.
- De plus, le DevOps peut se concentrer sur la pensée créative nécessaire pour concevoir des solutions adaptées aux besoins des clients.
  - Il devient ainsi possible de proposer plus rapidement de nouvelles fonctionnalités en réponse au demandes des clients

CaaS - vs. PaaS vs. IaaS

- Le CaaS se situe entre IaaS et PaaS
- Facilite la mise en place de PaaS
- Fonctionne sur du IaaS ou sur du bare-metal
- Simplifient la décomposition d’applications en microservices.
- Les développeurs peuvent utiliser des conteneurs sans avoir à se soucier de l'ensemble de l'infrastructure.
  - Ils sont ainsi en mesure de se concentrer sur les aspects applicatifs.

CaaS - vs. PaaS vs. IaaS



- L'architecture CaaS est composée de plusieurs couches :

1. Couche d’infrastructure : ressources physiques ou virtuelles qui composent la plateforme CaaS (Calcul, stockage, réseau, etc ...)

2. Couche d'orchestration de conteneurs : gestion du cycle de vie des conteneurs (provisionnement, mise à l'échelle et planification des conteneurs). Elle prend en charge les outils d'orchestration de conteneurs (Kubernetes, Docker Swarm, Apache Mesos, etc ...)

3. Couche de conteneurisation : regroupe les applications et les dépendances dans des conteneurs légers et portables. Les conteneurs créés à l'aide d'outils de conteneurisation tels que Docker et stockés/distribués à l'aide de registres de conteneurs tels que Docker Hub.

4. Couche de services de plateforme : décrit les services supplémentaires essentiels au déploiement et à la gestion des conteneurs (équilibrage de charge, découverte de services, journalisation, ...). Services généralement proposés par des fournisseurs CaaS et accessibles via des interfaces de programmation d'applications (API) ou des consoles Web.

5. Couche d'application : contient des applications conteneurisées déployées sur la plateforme CaaS. Les applications sont développées à l'aide de divers langages et frameworks de programmation et sont ensuite regroupées sous forme d'images Docker pour les déployer sur la plateforme CaaS.

Exemples pratiques

- Déploiement de microservices
  - particulièrement adapté à la mise à disposition des applications à microservices.
- Applications de machine learning et services web nécessitant une mise à l'échelle
- Exploitation d'applications cloud natives

Fournisseurs du CaaS

- AWS : Elastic Container Service (ECS), Elastic Kubernetes Service (EKS) et AWS Fargate
- Microsoft Azure : Azure Container Apps, Azure Container Instances, Azure Kubernetes Service (AKS)
- Google Cloud Platform : Google Kubernetes Engine (GKE)
- Autres : Cloud Kubernetes Service d'IBM, Red Hat OpenShift et Container Engine for Kubernetes (OKE) d'Oracle.

Démonstration : AWS – ECS & EKS

Bénéfices – Déploiement transparent

- Automatiser le processus de déploiement
- Simplifier la tâche d'emballage, de distribution, de sécurisation et de lancement des conteneurs
- Déployer des applications avec seulement quelques commandes,
- réduire l'intervention manuelle
- minimiser le risque d'erreur humaine.

Bénéfices – Scalabilité

- Mettre à l'échelle de manière transparente les applications vers le haut ou vers le bas en fonction de la demande
- Mécanismes de répartition automatique de la charge horizontale et de mise à l'échelle verticale
- Répondre aux pics de trafic ou à la croissance rapide sans avoir besoin d'une planification d'infrastructure étendue.

Bénéfices – Portabilité

- Permet la création d'applications natives du cloud qui s'exécutent sur divers fournisseurs de cloud ou environnements sur site.
- Permettre aux entreprises de choisir à tout moment l'infrastructure la mieux adaptée à leurs besoins

Bénéfices – Optimisation du coût

- CaaS peut réduire les coûts d'exploitation de l'ingénierie en diminuant les ressources nécessaires pour gérer un déploiement.
- Optimiser l'utilisation des ressources en empaquetant efficacement plusieurs conteneurs sur une seule ressource cloud (CPU, RAM, stockage).
- Contribuer aux économies de coûts
- Les organisations peuvent obtenir plus avec moins de ressources.

Bénéfices – Accélération DevOps

- Les CaaS favorisent une culture DevOps en encourageant la collaboration entre les équipes de développement et d'exploitation
- Les processus de déploiement et de gestion rationalisés :
  - encouragent l'itération rapide,
  - allègent les charges de travail
  - facilitent les pratiques d'intégration continue et de déploiement continu (CI/CD).

Limites – la sécurité

- De nouveaux défis de sécurité, à cause de la complexité même du conteneur type.
- Les images de conteneur incluent des bibliothèques système et d’autres fichiers et systèmes de fichiers dépendants
  - les vulnérabilités sont fréquentes et les conteneurs ont ainsi acquis une réputation d’insécurité.
- Développer et gérer les conteneurs avec la même préoccupation de sécurité que pour n’importe quel environnement informatique.

Limites – orchestration à définir

- Vous devez choisir votre outil d’orchestration pour les conteneurs (parmi lesquels Kubernetes, Docker Swarm, Amazon EKS, Rancher, Microsoft AKS…)
- La configuration et la gestion de l’orchestrateur et de ses composants peuvent être difficiles à maîtriser.

Limites – Stockage et protection des données

- Les données des conteneurs doivent être sauvegardées et stockées pour protéger les entreprises des risques (pannes, pertes de données…) qui peuvent survenir lors de la migration et du déploiement de nouvelles applications.

Evolution vers le CaaS

- Préparation
- Collecte d’informations
- Préparation des plans
- Étapes
- Les Best Practices

Préparation - Objectifs de la migration

- Évaluer les objectifs à atteindre grâce à la migration
  - accélérer le lancement d’un business
  - économiser les coûts
  - investissements attendus
  - Etc ...

Préparation – les besoins

- Identifier les différents besoins nécessaires pour réussir la migration vers le CaaS :
  - Compétences techniques :
    - Déterminer si l’entreprise doit recruter ou former le personnel technique
    - La migration des conteneurs peut nécessiter la prise en charge de plusieurs technologies, telles que conteneurs, Kubernetes, réseau, outils CI\CD, etc.

Préparation – les besoins

- Gouvernance :
  - Déterminer quelles équipes participeront au projet de migration
  - désigner le responsable de chaque équipe
  - identifier le décideur final pour la chaîne de décision du projet de migration.
  - Évaluer l’efficacité des outils de gestion de projet et de
  - Déterminer la façon de mesurer les résultats du projet
- Sécurité et conformité :
  - Déterminer les exigences de la sécurité des infrastructures, telles que les mesures de réseau et de cryptage.

Collecte d’informations - Informations générales

- La collecte d’informations s'effectue principalement à travers des questionnaires et des entretiens, pour formuler une compréhension globale des objectifs migratoires.
- Parmi les questions :
  - Quels applications à migrer ?
  - Quand commencer ? Quelle est la durée ? Y a-t-il une date limite?
  - Quel est le nombre de personnes responsables de la migration ? Quels types de compétences possèdent-ils ? De quels modules sont-ils responsables ?
  - Quel est le coût de la main d'œuvre ? Coût de la migration ? Coût de duplication de l’environnement ?

Collecte d’informations - Informations techniques

- Quelle infrastructure utilisée
- Quelle est l'utilisation réelle en matière de ressources de calcul, de stockage et de réseau ?
- L'application est-elle stateful ou stateless ?
- Quelles sont les dépendances entre les applications ?
- Existe-t-il des informations spécifiques à l’infrastructure des conteneurs utilisée ? (Exemples dans le diapo suivant)

Collecte d’informations - Informations spécifiques

- Version Docker
- Outil d'orchestration ? K8S\Mésos …
- Version de l’outil d'orchestration ?
- La version K8s est-elle compatible avec l'application ?
- Déployer sur quelle plateforme ? Rancher\OpenShift\ACK\GKE\AKS\ Kubeadm
- Y a-t-il un nœud Windows ?
- Quel plug-in ? CNI, CSI, Cluster Autoscaler, Ingress, etc.
- Quel type de stockage ? Volumes, CSI, EFS, etc.
- Quel type de réseau ? Flanelle/Calico/McVlan

Collecte d’informations - Informations spécifiques

- Quels outils de collecte de journaux ? Filebeat, Flume, Fluentd, etc.
- Quel outil de monitoring ? Prometheus/Nagios/Zabbix
- Quel type de Dashboard ? Kibana/Grafana
- Quelles métriques à surveiller ?
- Quelle solution de pipeline CI/CD ? Jenkins/GitLab-CI/GitOps
- Quel outil de package ? Helm chart / Kustomize
- Quelles plateformes de code source ? GitHub/GitLab/SVN
- Quel référentiel d'images ? DockerHub, registre privé

Préparation des plans

- Préparer les 4 plans suivants :
  - Plan de migration: Définir les étapes de la migration
  - Plan de test : Test unitaire, d’intégration, ...
  - Plan de basculement : Décrivant le processus de basculement en détail.
  - Plan de Rollback : Décrire le processus de revenir en arrière si le basculement de migration échoue

Étapes - Migration

- Créer l'environnement : Créer l'environnement de base et les opérations associées suivant le plan de migration.
- Utiliser des processus ou des outils automatisés pour la migration : La migration peut être effectuée à l'aide d'outils CI\CD ou d'autres outils de migration d'applications conteneurs,
- Checklist : suivez la Checklist pour confirmer si la migration a été
- terminé avant de basculer.

Étapes - Validation

- Chaque application doit passer par une série de tests spécifiques
  - Vérification fonctionnelle
  - Vérification des performances
  - Reprise après sinistre
- Une fois que tous les éléments de test ont réussi, vous pouvez passer à la phase de basculement.

Étapes - Basculement

- Appliquer le plan de basculement
- Basculez le flux vers le nouveau système et observer le fonctionnement.
- Exécuter le plan de Rollback en cas d’anomalies

- Prioriser la sécurité :
  - Protéger les environnements conteneurisés tout au long du cycle de vie des applications, du développement et du déploiement à la phase d'exécution de cette application.
  - Intégrer des outils de sécurité, ce qui peut contribuer à améliorer la sécurité de la plateforme CaaS.
- Surveiller les conteneurs :
  - Se concentrer davantage sur la surveillance orientée services et spécifique aux conteneurs que sur la surveillance basée sur l'hôte.
  - Les plates-formes CaaS isolent les conteneurs de l'infrastructure sous-jacente: surveiller uniquement les conteneurs au niveau du service, réduisant ainsi le besoin de surveiller les hôtes physiques.

- Mettre en œuvre un stockage compatible avec les conteneurs
  - Nécessite des plates-formes de stockage compatibles avec les conteneurs.
  - Les orchestrateurs de conteneurs peuvent se connecter aux fournisseurs de stockage à la volée et provisionner des volumes de stockage en fonction des besoins.
- Focus sur la mise en réseau de conteneurs
  - éviter la configuration réseau manuelle des environnements conteneurisés.
  - se concentrer sur l’automatisation du réseau

- Gérer le cycle de vie des conteneurs
  - Les conteneurs sont éphémères ; il est donc important de gérer leur cycle de vie pour éviter le gaspillage des ressources de calcul.
  - La gestion du cycle de vie doit être liée au processus d'intégration continue/déploiement continu (CI/CD).

- Implémenter l’orchestration des conteneurs
  - L'orchestration est cruciale lorsqu'il s'agit de déployer efficacement des conteneurs.
  - Cette couche communique avec les applications pour garantir que les conteneurs fonctionnent comme prévu et maintiennent les SLA requis.
  - Les orchestrateurs assument la responsabilité de positionner les conteneurs sur les hôtes appropriés dans un cluster, de provisionner les ressources réseau et de maintenir les applications en fonctionnement.
  - Kubernetes est l'une de ces normes connues pour la planification et l'orchestration de conteneurs.